Primitive Recursive Functions

Rickesh Bedia
1313253

The Significance of Primitive Recursive Functions

Primitive recursion is a natural way of defining a computation. Therefore once primitive recursion has been mastered, one might wonder: is that all there is in computation?

Well, it turns out it's not. First, you are missing undefined values. Therefore, you can extend primitive recursion to partial primitive recursive functions.

Still, there are some (complete) computable functions that are not primitive recursive, I.e the Ackermann function. This is why the Ackermann function is important. It turns out that *recursion* is *primitive recursion* + *minimization*, and while it cannot be proven, it seems, that's all there is to computation.

Primitive recursive functions are important, as they are

1.
2. Simple statements about the consequences of Manipulation Rules
3.
4. The base for the definition of recursion

1. Introduction

Primitive recursion is a procedure that defines the value of a function at an argument, *n,* by using its value at the previous argument, $n - 1$.

We will define primitive recursive functions as maps of the form

f:

Definition 1.1:
There are three types of basic functions:

- Constants Zero ,
- Projections ,
- for every , every,
- Successor Function ,

-

Defintion 1.2: Composition
Given functions , for , , we define a new function by composition as follows:

This can also be written as due to the special case when n=1:

Defintion 1.3: Primitive Recursion
Givenand , we define a new function
by
for all , where with

Defintion 1.4:
A function is primitive recursive if it can be constructed from the basic functions by applying compostion and primitive recursion

Example 1.5:We can now use the successor function, *S*, to define addition, multiplication, exponential and factorial

Addition: where the function is of the form
Example:

Multiplication: where the function is of the form
Example:

Exponential:where the function is of the form
Example:

Predecessor:

Subtraction:where the function is of the form
Example:

Factorial: where the function is of the form
Example:

1.1 Primitive Recursion with Parameters

Usually when defining a function of many variables by primitve recursion, all variables except one are fixed. primitive recursion with parametres allows us to use substitutions for these variables, which can then be reducible by primitive recursion.
Example 1.5: *Towers of Brahma*:

In the great temple Benares, beneath the dome which marks the centre of the worlds, rests a brass-plate in which are fixed diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly the priests transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When the sixty- four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish.

For the recursive solution, we want to move n disks from needle 1 to needle 3. Therefore, first we must move n – 1 disks from needle 1 to needle 2, then move the remaining disk from needle 1 to needle 3, and then finally move n - 1 disks from needle 2 to needle 3 I.e:

where 1 represents needle 1, 2 represents needle 2 and 3 represents needle 3.

Let be the number of moves for n disks
Therfore

which is the least number of moves needed to solve the problem.

In the case of *Towers of Brahma,* it would take moves for the world to vanish. If a correct move was made every second, this would take approximately 585 billion years to complete the task.

Example 1.6: One of the most famous examples of a primitive recursive functions is the problem set by Leonardo de Pisa, in his 1202 paper Liber adaci:

How many pairs of rabbits can be bred in one year from one pair?

The problem was set under the conditions:

-
- A wall surrounds the rabbits, therefore none can escape or be killed
-
- A pair of rabbits breed every month
-
- A rabbit breeds in the second month after it was born

This sequence gives the Fibonnaci numbers:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, …..

Defintion 1.7: Fibonacci Sequence

1.2 Double Recursion

Primitive Recursion is used to define functions of many variables by only keeping all but one variable fixed. Double recursion allows recursion to happen on two variables. This is more general and is reducible in many cases, but not all, to primitive recursion.

Theorem 1.8: The Ackermann's function is a double reccursive function which is not primitive recursive:

Proof:

We need to check two possibilities (i) a composition of primitive recursive functions and (ii) a recursive chain of primitive recursive functions grows faster than any primitive recursive function. We shall show that no such fuction exists, hence the Ackermann function is not a primitive recursive function.

First we shall prove (i):
Let s.t. where are the first parameters of the Ackermann function. Then letthen we shall show first that
Let (x times on the right hand side)

To aid our proof we will show two further observations:

1.
2. where both are postive
3.
4.

We know that (1) follows from . To see this, assume x = 3, so we have . (1) follows by induction
For (2) we will use the defintion of the Ackermann function and consider . This completes the proof of (2), since the simple case where the number od Ackermann compostions is 0.

With these two facts, we can finish the proof. Looking at our initial inequalities:

but by our first observation we can see that

which completes the proof as we have found a value of the Ackermann function which is strictly greater than the compostion of any two primitve recursive functions.

Now we shall prove (ii):
Assume f(x) is defined by the recursive equation:
Assume g is an increasing primitive recursive function, we know that there is an Ackermann function larger than g and we shall show by induction that for m>0.
We know that f(0) is a constant so from earlier observations we have an Ackermann function greater than f(0)
Assume the observation holds for all case
Let and observe
Then, since g is increasing, but the induction hypothesis tell us:

by substituation and out choice of m, we have:
which we know by defintion of the Ackermann function and substitution that:
Thus we can conclude that

This proof tells that the Ackermann's function grows extremely fast, faster than any primitive recursive function and the Ackermann function is not primitive recursive because all primitve recursive functions are in fact smaller than the Ackermann function

2. Recursion Theorem

The recursion theorem allows us to compute values of functions which are solutions to recursive equations. We will use a procedure similar to the classical method of computing approximations to real numbers which are solutions to algebraic equations. Therefore, we can implicitly define real numbers by circular definitions involving the original number itself. We need circular definitions in order to use recursive functions

Example 2.1: Consider:

x can now be thought of as a fixed point of the function

such that

We have two different methods of making *x* explicit

1.
2. We can write the equation in the form
3.
4.
5. Solving the equation gives
6.
7.
8. However, this only works for simple functions and the solutions are no longer circular, but still implicit.
9.
10. We can perform repeated substitutions of the *x* on the right hand side of the original equation to obtaina continuous fraction
11.
12.
13. The infinite expression is built up as a limit of finite expressions, that provide approximations to the solution.

14.
15. If we write the approximation as
16.
17. then
18.
    1.
19.
20. This means f is the Fibonacci sequence, and the approximations are given by the ratio of terms
21.
    1.
22.

This keeps the result both circular and implicit so we are able to use recursive functions

2.1 Differentiable Functions

We want to compute numerical approximations to

Our first approximation can be computed by dissecting a rectangle of edges 1 and 2, with are 2, into two equal squares. We then cut one square into two rectangles of edges 1 and ½. We can place these rectangles along side the other square, creating a square of edge 3/2, with area greater than 2 by a smaller square of edge ½, producing an error of ¼.

1. 2. 3. 4.

Our second approximation can be computed from the square of edge 3/2. Two rectangular strips of area 1/8 and short edge giving a square of edge , with area differing from 2 by a small square of edge 1/12, producing an error of 1/144

This procedure can be iterated. Given an approximation
where

-
- is the error of the approximation

-
- is the area of each of the rectangular strips
-
- is the short edge of the rectangular strip

If we write , then and
and the recursive formula can be written as , this is Newton's Formula used to approximate when a function is equal to zero.

Newton's formula can also be found by looking at a function *f* for an increment *h* s.t
I.e

We can ignore the , the error term, to get
which can be rewritten as

In general, for any function *f,* the increment is given by Taylor's Formula

2.3 Contractions

Example 2.2: Royce presented a paradoxical metaphor

Imagine a portion of the territory of England has been perfectly levelled, and a cartographer traces a map of England. The work is perfect. There is no particular of the territory of England, small as it can be, that has not been recorded in the map. Everything has its own correspondence. The map, then, must contain a map of the map, that must contain a map of the map of the map, and so on to infinity.

To solve this metaphor, we use a proof by contradiction that perfect maps are impossible. However, from a mathematical viewpoint, a perfect map that contains a copy of itself is not a contradiction, instead a contraction. For a function *f* to be a contraction mapping:

Banch proved that a contraction has a unique fixed point by the iteration:

By the triangle inequality:

Therefore the sequence converges to a point , which implies, as *f* is continuous, by assumption ,the sequence also converges to which must be equal to . Therefore is a fixed point of *f*. If we take as a fixed point as well,
As $K < 1$, it implies and is a unique fixed point of *f*.

If we now go back to the example, there must be a point in the territory that coincides with its image on the map. Therefore, a perfect map is not the whole territory, but a single point.

3. Kleene's Second Recursion Theorem

Notation: If *e* is a number, then {*e*} is the partial function corresponding to *e.*

Theorem 3.1: For any computable function *F* there exists an *e* such that $\{e\} \simeq \{Fe\}$

Note, *F* can be any total function. Let *F* be the successor function, which is computable. The theorem now states that there are two successive natural numbers *e* and *e*+1 such that $\{e\} \simeq \{e+1\}$

For the proof we will us the fact that the function can be self-applied, $\{e\}(e)$. This self-application is computable, therefore there is a function h such that $\{h(e)\} \simeq \{\{e\}(e)\}$.

Proof:
Assume that *F* is a total computable function. For any *x*, let $h(x)$ be a function such that $\{h(x)\} \simeq \{\{x\}(x)\}$. Therfore a total and computable function *h* exists.
Let there be a *k* such that $\{k\}(x) \simeq F(h(x))$. This *k* exists as *F* and *h* are computable.
We claim $e=h(k)$ solves our problem.
We know $\{h(x)\} \simeq \{\{x\}(x)\}$ and $e=h(k)$, so $\{e\}(x) \simeq \{h(k)\}(x) \simeq \{\{k\}(k)\}(x)$
But as $\{k\}(x) \simeq F(h(x))$, $\{\{k\}(k)\}(x) \simeq \{F(h(k))\}(x) \simeq \{F(e)\}(x)$

The Second Recursion Theorem allows us to find explicit solutions to recursive equations, defining programs of recursive functions by circular defintions.

The procedure is comparable to the classical methods of finding explicit defintions for functions defined by recursive equations. Consider the implicit definition of the Fibonacci sequence:

We can use De Moivre's method of genrating functions to make *f* explicit

By computing

we notice that most terms cancel as so we get:

We can obtain the explicit form of *f*, by expanding the right-hand side into a power series and comparing it to *F*(*x*):

The Second Recursion Theorem turns recursive programs which define functions by recursive calls, into programs for the same functions without any recursive calls.
A recursive call is when the function calls itself directly or indirectly.

Conclusion

At the start of this essay we began we very simple examples of recursive functions, addition, multiplication, exponential etc and we used these recursive functions to prove the Ackermann Function, showing how complex Analysis ideas such as Differentiation and Contraction Mappings can be reduced to Recursive Functions.
Finally we proved Kleene's Second Recursion Theorem. This theorem in particular is important as it can be used to create self-reproducing and self-recognising programs. The most important however is it's applications to the Turing machine, closing them under recursion which Turing showed using non recursive construction.

References

Curry, H. (1942). *The Inconsistency of Certain Formal Logics*. 7th ed. Journal of Symbolic Logic, pp.115–117.

Fibonacci's Liber abaci: a translation into modern English of Leonardo Pisano's book of calculation. (2004). *Choice Reviews Online*, 41(09).

Kleene, S. (1935). A Theory of Positive Integers in Formal Logic. Part I. *American Journal of Mathematics*, 57(1), pp.153–173, 219–244.

Kleene, S. (1938). On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(04), pp.150-155.

Peter, R. (1951). *Recursive Function*. Budapest: Akademiai Kiado, p.30.

Péter, R. and Kleene, S. (1937). λ-Definability and Recursiveness. *The Journal of Symbolic Logic*, 2(01), pp.340–353.

Royce, J. (1901). *The World and The Individual*. 2nd ed. New York: Macmillian.
Russell, B. (1938). *Principles of mathematics*. New York: W.W. Norton & Co.
Singh, P. (1985). *The Socalled Fibonacci Numbers in Ancient and Medieval India*. Historia Mathematica, pp.229–244.
Tourlakis, G. (2012). *Theory of computation*. Hoboken, N.J.: Wiley.