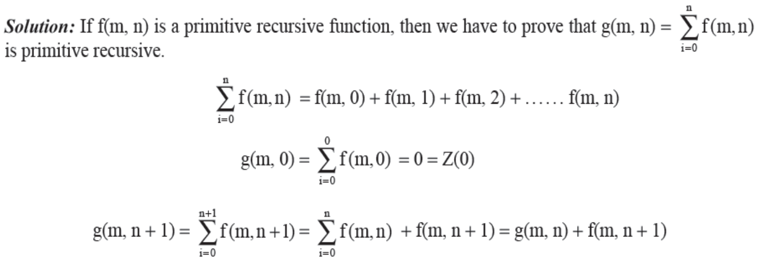
Solution:
$$fact(0) = 0! = 1 = s(0)$$
$$fact(x+1) = x \times fact(x) = mult(s(x), fact(x))$$

As the multiplication function is primitive recursive, fact is also primitive recursive.

Example 11.7 Prove that the bounded addition of the primitive recursive function is also primitive.



Solution: If f(m, n) is a primitive recursive function, then we have to prove that $g(m, n) = \sum_{i=0}^{n} f(m,n)$ is primitive recursive.

$$\sum_{i=0}^{n} f(m,n) = f(m, 0) + f(m, 1) + f(m, 2) + \ldots\ldots f(m, n)$$

$$g(m, 0) = \sum_{i=0}^{0} f(m,0) = 0 = Z(0)$$

$$g(m, n+1) = \sum_{i=0}^{n+1} f(m,n+1) = \sum_{i=0}^{n} f(m,n) + f(m, n+1) = g(m, n) + f(m, n+1)$$

Thus, it is also primitive recursive.

(It can also be proved in the same way that the bounded product of primitive recursive functions is also primitive recursive.)

11.4 Gödel Number

In the late 19th century, two properties of a formal system were the cause of brainstorming for the logicians and mathematicians. The properties were completeness and consistency. Completeness is defi ned as the possibility to prove or disprove any proposition that can be expressed in the system. On the other hand, consistency is the impossibility to both prove and disprove a proposition in a system. Using the axioms of set theory, many mathematicians try to prove these properties of a formal system.

A British philosopher and logician Bertrand Arthur William Russell in 1901 discovered a paradox known as the Russell's paradox.

11.4.1 Russell's Paradox

S is defined as the set of all sets that are not members of themselves. Is S a member of itself? From this paradox, two conditions appear for S.

1. If S is an element of S, then S is a member of itself and should not be in S.
2. If S is not an element of S, then S is not a member of itself, and should be in S.

In this context, Kurt Gödel, an Australian (later American), proposed a theorem in 1931 known as the Gödel Incompleteness Theorem. The theorem states that 'if a formal theorem is consistent, then it is impossible to prove within the system that it is inconsistent.'

The concept of the Gödel number was used by Gödel in proving his famous incompleteness theorem. The Gödel numbering is a function that assigns each symbol and a well-formed formula of some formal language to a unique natural number. It can also be used as an encoding scheme that assigns a unique number to each symbol of a mathematical notation. The Gödel numbering encodes a sequence of numbers in a single value. The scheme of converting a sequence of natural numbers uses the factorization of a natural number into a product of prime numbers.

The Gödel number for a sequence $x_0, x_1, ...x_n - 1$ for n natural numbers is defined as

$$pn(0)_0^x \times pn(1)_1^x \times pn(2)_2^x \times ......pn(n))_n^x = 2_0^x \times 3_1^x \times 5_2^x \times ......pn(n)_n^x$$

where pn(n) is the nth prime number.

As an example, the sequence $2, 1, 0, 3$ is encoded as $2^2 3^1 5^0 7^3 = 4116$. The sequence $1, 0, 2, 1$ is encoded as $2^1 3^0 5^2 7^1 = 350$.

Decoding a Gödel number G to obtain the sequence is the reverse process. The steps are

- Prime factorize the given number.
- Arrange the obtained primes in sequence, starting from the least and raise appropriate power with the number of times they appear. If any prime number in between two does not appear, raise the power as ‘0’.

Decoding a Gödel number G to obtain the sequence is the reverse process. The steps are

- Prime factorize the given number.
- Arrange the obtained primes in sequence, starting from the least and raise appropriate power with the number of times they appear. If any prime number in between two does not appear, raise the power as ‘0’.

As an example, take a number 10780. The prime factorization of the number results in $2 \times 2 \times 5 \times 7 \times 7 \times 11 = 2^2 \times 5^1 \times 7^2 \times 11^1$.

In the sequence ‘3’ is missing, though 3 is a prime number in between 2 and 5. Thus, the modifi ed sequence with power raised to each prime is $2^2 \times 3^0 \times 5^1 \times 7^2 \times 11^1$.

Hence, the sequence is $(2, 0, 1, 2, 1)$.

The function for constructing a Gödel number is not one to one. Two different sequences may have the same Gödel number. These two sequences are different with the number of ‘0’ at last. As an example, the sequences $(2, 0, 1)$ and $(2, 0, 1, 0, 0)$ have the same Gödel number.

11.5 Ackermann's Function

The German mathematician Wilhelm Friedrich Ackermann discovered a function which proves that all total computable functions are not primitive recursive. This function is named after him and known as the Ackermann's function.

For two non-negative integers x and y, the Ackermann's function is defi ned as

11.5 Ackermann's Function

The German mathematician Wilhelm Friedrich Ackermann discovered a function which proves that all total computable functions are not primitive recursive. This function is named after him and known as the Ackermann's function.

For two non-negative integers x and y, the Ackermann's function is defi ned as

$$A(x,y)=\begin{cases} y+1 & \text{if } x=0 \\ A((x-1),1) & \text{if } x>0 \text{ and } y=0 \\ A((x-1),A(x,(y-1)) & \text{if } x>0 \text{ and } y>0 \end{cases}$$

For every non-negative integer x and y, the function A(x, y) can be computed. So, it is a total function.

$A(x,0) = A((x-1),1) = A((x-2), A((x-1),(1-1))) = A((x-2), A((x-1),0))$

(this will continue recursively).

Here, we are not getting a zero function, which proves that it is not primitive recursive.

Example: *Compute A(1, 3).*

$$\begin{aligned} A(1,3) &= A(0, A(1,2)) As x, y > 0 \\ &= A(0, A(0, A(1,1))) \end{aligned}$$

$$
\begin{aligned}
&= A(0, A(0, A(0, A(0, 1)))) \\
&= A(0, A(0, A(0, 2))) \\
&= A(0, A(0, 3)) \\
&= A(0, 4) \\
&= 5
\end{aligned}
$$

Example: *Compute A(2, 2).*

$$
\begin{aligned}
A(2, 2) &= A(1, A(2, 1)) \\
&= A(1, A(1, A(2, 0))) \\
&= A(1, A(1, A(1, 1))) \\
&= A(1, A(1, A(0, A(1, 0)))) \\
&= A(1, A(1, A(0, A(0, 1)))) \\
&= A(1, A(1, A(0, 2))) \\
&= A(1, A(1, 3))
\end{aligned}
$$

$$
\begin{aligned}
&= A(1, A(0, A(1, 2))) \\
&= A(1, A(0, A(0, A(1, 1)))) \\
&= A(1, A(0, A(0, A(0, A(1, 0))))) \\
&= A(1, A(0, A(0, A(0, A(0, 1))))) \\
&= A(1, A(0, A(0, A(0, 2)))) \\
&= A(1, A(0, A(0, 3))))) \\
&= A(1, A(0, 4)) \\
&= A(1, 5) \\
&= A(0, A(1, 4)) \\
&= A(0, A(0, A(1, 3))) \\
&= A(0, A(0, A(0, A(1, 2)))) \\
&= A(0, A(0, A(0, A(0, A(1, 1))))) \\
&= A(0, A(0, A(0, A(0, A(0, A(1, 0)))))) \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= A(0, 6) \\
&= 7
\end{aligned}
$$

11.6 Minimalization

Suppose there is a function f(x) which has a least value of x which makes f(x) = 0. Now, it is told to fi nd that least value. We shall consider x = 0, 1, 2, ... and calculate f(x) correspondingly. Where f(x) = 0 is achieved, that value of x is the least value. The process will stop within a fi nite amount of time after a fi nite number of steps. Let g(x) be a function which computes the process of fi nding the least value of x such that f(x) = 0. So, g is computable. It is said that g is produced from f by minimization.

As an example, let $f(x, y) = 2x + y - 5$, where x and y are both positive integers (I).

This is a total function as for every positive integer value of x and y, there is a value of $f(x, y)$.

We have to find the range of x for which $y \in I$.

Make $f(x, y) = 0, i.e., 2x + y - 5 = 0$

$\rightarrow y = 5 - 2x$

If $x > 2$, then there is no value of y such that $y \in I$.
Consider this function as g(x, y). We can write

$$g(x, y) = 5 - 2x \qquad \text{for } x \leq 2$$
$$= \text{Undefined} \qquad \text{for } x > 2.$$

Let g be a $k + 1$ argument function. The unbounded minimalization of g is a k argument function f such that for every arguments $n_1, n_2, ...n_k$

$f(n_1, n_2, ...n_k) =$ minimum value i such that $g(n_1, n_2, ...n_k, i)$ exist.

$= \text{Undefined otherwise.}$

But if such an f(x) is given for which it is not known whether there exists an x for which f(x) = 0, then the process of testing may never terminate. This is called unbounded minimalization. If g is produced by unbounded minimization, then it is not effectively computable as there is no surety of termination.

If the searching is set to an upper limit, then the function g can be made computable. The search process will find a value which makes f(x) = 0 and the function g will return that value if such a value is found. Otherwise, g will return 0. This is called bounded minimization as a boundary of search process is specified.

Bounded minimalization is defi ned by the search operator over the natural numbers $\leq y$

$$f(x_1, ..., x_n, y) = \mu^y z[p(x_1, ..., x_n, z)]$$

It defines a function f which returns the least value of z satisfying $p(x_1, ..., x_n, z)$ or returns the bound. More precisely, it can be written as

But if such an f(x) is given for which it is not known whether there exists an x for which f(x) = 0, then the process of testing may never terminate. This is called unbounded minimalization. If g is produced by unbounded minimization, then it is not effectively computable as there is no surety of termination.

If the searching is set to an upper limit, then the function g can be made computable. The search process will find a value which makes f(x) = 0 and the function g will return that value if such a value is found. Otherwise, g will return 0. This is called bounded minimization as a boundary of search process is specified.

Bounded minimalization is defi ned by the search operator over the natural numbers $\leq y$

$$f(x_1, \ldots, x_n, y) = \mu^y z[p(x_1, \ldots, x_n, z)]$$

It defines a function f which returns the least value of z satisfying $p(x_1, \ldots, x_n, z)$ or returns the bound. More precisely, it can be written as

$$f(x_1, \ldots, x_n, y) = \begin{cases} \min z \text{ subject to } p(x_1, \ldots, x_n, z) = 1 \text{ and } 0 \leq z \leq y \\ y + 1 \text{ otherwise} \end{cases}$$

11.7 μ Recursive

A function is said to be μ recursive if it can be constructed from the initial functions and by a finite operation of:

- Composition
- Primitive recursion
- Unbounded minimalization (μ operation)

  This type of function was first proposed by Gödel and Stephen Kleene.

  It can be proved that 'a function is computable by a Turing machine if it is μ recursive'.

11.7.1 Properties of a μ Recursive Function

- The zero function, successor function, and projection function are μ recursive.
- If f is a m variable μ recursive function and g is a n variable μ recursive function, then $f(g_1, g_2, ...g_n)$ is also μ recursive.
- If f and g are n and $n+2$ variable μ recursive functions, then a new function, say h constructed from f and g by primitive recursion, is μ recursive.