Some useful definitions:

(a) Adding the minimization operator μ to primitive recursion in effect adds computation with open-ended searches ("do until" loops) to fixed-depth searches ("for" loops). So that's what gives us access to full-power, unrestricted computation.

Yes, the difference is just whether there is a given upper bound. Let $P(x)$ be a predicate on natural numbers:

- with the unbounded μ operator, $(\mu x, P(x))$ is the smallest x such that $P(x)$ is true.
- with the bounded μ operator, $(\mu x_{x<z}, P(x))$ is the smallest x less than z such that $P(x)$ is true.

The point is that the unbounded μ lets you define all the computable functions, but it does searches in infinite sets, resulting in algorithms which may not terminate. The bounded μ has less expressive power, belonging to the primitive recursive functions, but the "upside" is that the algorithms behind primitive recursive functions always terminate.

> > $f(x) = (\mu i)[g(x, i)],$
>
> where $g(u, v) = 0$ if $u = v^2$, and $g(u, v) = 1$ if $u \neq v^2$. Then, g is a total function, but f is not. If x is not a perfect square, then the value of $f(x)$ is undefined. (If x is a square, the value of $f(x)$ is defined and equals $\sqrt{x}$.)

In this example, we have that $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$.

The function $f(x)$ means : "the least number i such that $x = i^2$".

For $x = 1$ we have that $i = 1$, and thus $f(1) = 1$.

For $x = 2$, we have no i, and thus $f(2)$ is undefined. The same for $x = 3$.

For $x = 4$, instead, we have that $i = 2$ and thus $f(4) = 2$.

And so on.

I think you're confused about definitions. What you represented above is *bounded* minimization where we look for the least number z *below* a given bound k such that $f(n, z) = 0$. Such bounded minimization can be defined without any extra gadgets, using only primitive recursion.

The so-called μ operation performs *unbounded* search. It is not something we can define using primitive recursion. It is a *new* primitive operation which we add to primitive recursive functions to obtain recursive functions.

That is, if f is a function of several arguments then $\mu(f)$ is a *partial* function which satisfies

$$\mu(f)(\vec{n}) = k \iff f(\vec{n}, k) = 0 \land \forall j < k, f(n, k) \neq 0.$$

If there is no such k then $\mu(f)(\vec{n})$ is undefined.

Every primitive recursive function is total. However, μ allows us to create non-total functions, for instance $\mu(f)(1)$, where $f(n, k) = 1$, is everywhere undefined. Therefore, μ is not something that can be defined using primitive recursion.

There are various mechanisms that exceed the power of primitive recursion which allow us to define μ. One such is a *general recursion*, and another is a *fixed-point operator*.

> A partial function $f\ : N \to N$ is μ-recursive if it can be defined from basic primitive recursive functions by
>
> - composition,
> - primitive recursion,
> - unbounded minimization.
>
> Unbounded minimization can be applied to unsafe predicates. The function $\mu i\ p(\overline{n}, i)$ is undefined when there is no i such that $p(\overline{n}, i) = 1$.

6.4 The μ-recursive functions

Unbounded minimization :

$$\mu i\ q(\overline{n}, i) = \begin{cases} \text{the smallest } i \text{ such that } q(\overline{n}, i) = 1 \\ 0 \text{ if such an } i \text{ does not exist} \end{cases}$$

A predicate $q(\overline{n}, i)$ is said to be *safe* if

$$\forall \overline{n}\ \exists i\ q(\overline{n}, i) = 1.$$

The μ-recursive functions and predicates are those obtained from the basic primitive recursive functions by :

- composition, primitive recursion, and
- unbounded minimization of safe predicates (safe unbounded minimization).

6.4 Beyond primitive recursive functions

Theorem
There exist computable functions that are not primitive recursive.

A	0	1	2	$\ldots$	j	$\ldots$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$\ldots$	$f_0(j)$	$\ldots$
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$\ldots$	$f_1(j)$	$\ldots$
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$\ldots$	$f_2(j)$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
f_i	$f_i(0)$	$f_i(1)$	$f_i(2)$	$\ldots$	$f_i(j)$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$

$$g(n) = f_n(n) + 1 = A[n, n] + 1.$$

is not primitive recursive, but is computable.