A number register machine consists of:

- finite number of registers \( R_0, \ldots, R_n \) (each of which can hold a natural number)

- program given by a finite list of instructions \( L_0, \ldots, L_m \)

There are three types of instructions:

1. \( R_k := R_k + 1 \) (This instruction adds one to contents of \( R_k \))

2. HALT (This instruction terminates the program.)

3. IF \( R_k \neq 0 \) THEN \( R_k := R_k - 1 \) and \( L_i \) ELSE \( L_j \)
   (This instruction checks if the contents of \( R_k \) is 0
   IF \( R_k \neq 0 \), then it subtracts 1 from \( R_k \) and goes to
   instruction \( L_i \). IF \( R_k = 0 \), then it leaves \( R_k = 0 
   and goes to instruction \( L_j \).)

Convention: If there is no ELSE clause, then if \( R_k = 0 \)
just go to the next instruction.
If a register machine $M$ has registers $R_0, \ldots, R_n$ and instructions $L_0, \ldots, L_m$ we can start it with whatever initial values in $R_0, \ldots, R_n$ (from $M$) we want. We call these initial values the input. If $M$ eventually reaches a HALT instruction, then we view the contents of register $R_0$ as the output of $M$ on the given input.

So, given $M$ with $R_0, \ldots, R_n$ and $L_0, \ldots, L_m$ we can define a function for each $k \leq n$ by

$$f^k_M : \mathbb{N}^k \to \mathbb{N}$$

by $f^k_M (n_0, n_1, \ldots, n_{k-1})$ is calculated by starting $M$ with $R_0 = n_0, R_1 = n_1, \ldots, R_{k-1} = n_{k-1}, R_k = 0, \ldots, R_n = 0$ and setting $f^k_M (n_0, n_1, \ldots, n_{k-1}) = \text{output of } M$ when it reaches HALT instruction with these inputs.
Example 1. Let $M$ have one register $R_0$ and program

$L_0$. $R_0 = R_0 + 1$

$L_1$. If $R_0 \neq 0$ then $R_0 = R_0 - 1$ and $L_0$.  
$L_2$. HALT.

No matter what we start with in $R_0$, this program never halts!

Lesson: The functions $f^k_M : \mathbb{N}^k \to \mathbb{N}$ defined by register machines are partial functions. This means their domain is

$$\text{domain}(f^k_M) \subseteq \mathbb{N}^k$$

and may be a strict subset.
Example 2: Let $M$ have 2 registers $R_0$ and $R_1$.

$L_0$. $R_0 = R_0 + 1$
$L_1$. If $R_1 \neq 0$, then $R_1 = R_1 - 1$ and $L_0$.
$L_2$. If $R_0 \neq 0$, then $R_0 = R_0 - 1$ and $L_3$
$L_3$. HALT.

Think about starting this program with

<table>
<thead>
<tr>
<th>$R_0$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$m$</td>
</tr>
</tbody>
</table>

and convince yourself it will eventually halt with $R_0 = n + m$.

Therefore, $f^2_M : \mathbb{N}^2 \rightarrow \mathbb{N}$ is $f^2_M(n,m) = n + m$.

Def: A partial function $g : \mathbb{N}^k \rightarrow \mathbb{N}$ is \underline{\text{partial register machine computable}} $\iff$ there is a register machine $M$ such that $g = f^k_M$. If $g$ is total, we call it \underline{\text{register machine computable}}.

So, addition is register machine computable.
Exercise 1. Prove that

\[ f(n,m) = \begin{cases} 
  n - m & \text{if } m \leq n \\
  0 & \text{otherwise}
\end{cases} \]

is register machine computable. We denote \( f(n,m) \) by \( n \cdot m \). (So \( n \cdot m = n - m \) if \( m \leq n \) and \( n \cdot m = 0 \) if \( m > n \).)

In presenting other examples we will often use "pseudo-instructions" that we have already defined program for. We write these as \( L^* \).

Example 3. Let \( M \) have 4 registers and the following program. (Think of \( M \) starting with \( R_0 = n, R_1 = m, R_2 = 0 \) and \( R_3 = 0 \) so it gives function \( f^*_M : \mathbb{N}^2 \rightarrow \mathbb{N} \).)

\[
\begin{array}{|c|c|}
\hline
R_0 & n \\
\hline
R_1 & m \\
\hline
R_2 & 0 \\
\hline
R_3 & 0 \\
\hline
\end{array}
\]

\( L_0: \) Add \( R_0 \) to \( R_2 \) \hspace{1cm} \text{[really this means to]} \hspace{1cm} \text{[use our program above]} \]

\( L_1: \) If \( R_1 \neq 0 \), then \( R_1 = R_1 - 1 \) and \( L_2 \) else \( L_6 \)

\( L_2: \) Add \( R_2 \) to both \( R_0 \) and \( R_3 \)

\( L_3: \) Add \( R_3 \) to \( R_2 \)

\( L_4: \) \( R_3 = R_3 + 1 \)

\( L_5: \) If \( R_3 \neq 0 \) then \( R_3 = R_3 - 1 \) and \( L_4 \).

\( L_6: \) HALT.
You should convince yourself that using $M$ we get

$$f^2_M(n,m) = n \cdot m$$

**Recall:** When we use our "adding program" from before, when we "Add $R_0$ to $R_1$," we are left with $R_0 = 0$ and $R_1 := R_1 + R_0$!

**Exercise 2:** Show that $g(n,m) = n^m$ is register machine computable.

**Example 4** The composition of register machine computable functions is register machine computable.

Suppose $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ are partial register machine computable.

$g$ computed by $\hat{M}$ with $k$ registers and $l$ instructions.

We want machine to compute $g(f(n))$
We build $M^*$ with $L+K+1$ registers and the following program:

\[
\begin{array}{c|c}
R_0 & n \\
\vdots & 0 \\
R_{i-1} & 0 \\
R_i & 0 \\
R_{i+1} & 0 \\
\vdots & \vdots \\
R_{i+k-1} & 0 \\
R_{i+k} & 0
\end{array}
\]

We will use for "f" part

We will use for "g" part

← We will use for "Go to" statement.

$L_0^* : R_{i+k} = R_{i+k} + 2$ [This is really 2 instructions!]

$L_k^* : \{ \text{put in instructions from } M \text{ for } f \text{ (but relabelled as } L_1, \ldots, L_j \text{ instead of } L_0, \ldots, L_{j-1}) \text{ and replace each HALT by "If } R_{i+k} + 0 \text{ then } R_{i+k} = R_{i+k-1} \text{ and } L_{j+1} " \}.$

$L_{j+1}^* : \text{Add } R_0 \text{ to } R_{i+k}.$

$L_{j+2}^* : \{ \text{put in instructions from } M \text{ for } g \text{ (but relabelled as } L_{j+2}, \ldots, L_{j+k+2} \text{ instead of } L_0, \ldots, L_{k-1} \text{ and } R_0, \ldots, R_{k-1}) \text{. Replace HALT by "If } R_{i+k} + 0 \text{ then } R_{i+k} = R_{i+k-1} \text{ and } L^*_{j+k+3} " \}.$
Exercise 3. Write a program that will make the contents of $R_i = 0$ and then HALT (or just go to next instruction).
Write a program that will make $R_j = 0$ for all $j \geq i$ and then HALT (or go to the next instruction).

Exercise 4. Show that for each $k \geq 1$ and $1 \leq i \leq k$ the function $\Pi^k_i : \mathbb{N}^k \to \mathbb{N}$ given by

$$
\Pi^k_i (n_1, n_2, \ldots, n_k) = n_i
$$

is register machine computable.
Def: Fix $k \geq 0$ and suppose $g: \mathbb{N}^k \to \mathbb{N}$ and $h: \mathbb{N}^{k+2} \to \mathbb{N}$. We say $f: \mathbb{N}^{k+1} \to \mathbb{N}$ is defined by primitive recursion from $g$ and $h$ if
\[
  f(\overline{x}, 0) = g(\overline{x}) \quad \text{for } \overline{x} \in \mathbb{N}^k
\]
and
\[
  f(\overline{x}, y+1) = h(\overline{x}, y, f(\overline{x}, y))
\]

Think of this process as defining $f$ "by induction" on the last variable using $g$, $h$. If $k=0$ in this definition, then $g: \mathbb{N}^0 \to \mathbb{N}$ just means $g$ is a constant (i.e., is just an element of $\mathbb{N}$).
Prim. Rec. Ex. 1 We can define addition by primitive recursion from \( g(x) = x \) and \( h(x, y, z) = z + 1 \).

Why? \( f(x, 0) = g(x) = x = x + 0 \)

\[
f(x, y+1) = h(x, y, f(x, y)) = h(x, y, x+y) \quad \text{by induction hypothesis}
\]

\[
= (x+y)+1 = (x+(y+1)).
\]

So by induction on \( y \) we just showed \( f(x, y) = x+y \).

Exercise 5 Show multiplication is defined by primitive recursion using \( g(x) = 0 \) and \( h(x, y, z) = x \cdot z \).

Show the exponentiation is defined by prim. rec. using \( g(x) = 1 \) and \( h(x, y, z) = x \cdot z \).
Our next goal is to show that if \( g: \mathbb{N}^k \rightarrow \mathbb{N} \) and \( h: \mathbb{N}^{k+2} \rightarrow \mathbb{N} \) are register machine computable, then so is \( f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \) defined by primitive recursion from \( g, h \). To do this, we introduce some new "pseudo-instructions":

\[ \text{Add } R_i \text{ to } R_i + R_j \]

\[ \text{Erase } R_i \] This means set \( R_i = 0 \) as in previous exercise.

Notice we can do this with instruction of form:

\[ L_j: \text{ IF } R_j \neq 0 \text{ then } R_i = R_{i-1} \text{ and } L_j \text{ else } L_{j+1} \]

\[ \text{Move } R_i \text{ to } R_j \] This means Erase \( R_j \) and use our old "odd" program to add \( R_j \) to \( R_i \). In the end we have

\[ \text{New } R_j = \text{ Old } R_i \]

\[ \text{New } R_i = 0 \]
Copy $R_i$ to $R_j$

To do this we use an auxiliary register $R_*$ (where assume $* = \text{big index not currently using to store anything important}$):

1. $L_e^*$: Erase $R_j$ and $R_*$.
2. $L_{e+1}^*$: Move $R_i$ to both $R_j$ and $R_*$.
3. $L_{e+2}^*$: Move $R_*$ to $R_i$.

So we get: New $R_j = \text{Old } R_i$,

New $R_i = \text{Old } R_i$,

New $R_* = 0$.

Compare $R_i, R_j$; If $R_i \neq R_j$ then $L_m$ else $L_n$.

For this pseudo-instruction we will use 4 auxiliary registers $R_*, R_{*+1}, R_{*+2}, R_{*+3}$. 
Our instruction to do compare operation look like:

\[ L_e^* : \text{Copy } R_i \text{ to } R_x \text{ and } R_{x+1}^* \]
\[ L_{e+1}^* : \text{Copy } R_j \text{ to } R_{x+2} \text{ and } R_{x+3}^* \]

\[ \begin{align*}
\text{At this point we have:} & \\
R_x &= R_i \\
R_{x+1} &= R_i \\
R_{x+2} &= R_j \\
R_{x+3} &= R_j
\end{align*} \]

\[ L_{e+2}^* : \text{Use our subtraction program} \]
\[ \text{to get } R_x^* := R_x - R_{x+2} \]

\[ L_{e+3}^* : \text{Use our subtraction program} \]
\[ \text{to get } R_{x+3}^* := R_{x+3} - R_{x+2} \]

\[ \begin{align*}
\text{At this point we have:} & \\
R_x &= R_i \div R_j \\
R_{x+2} &= R_j \div R_i
\end{align*} \]

\[ L_{e+4}^* : \text{If } R_x > 0 \text{ then } R_x := R_x - 1 \text{ and } L_m \text{ else } L_{e+5}^* \]

\[ L_{e+5}^* : \text{If } R_{x+3} > 0 \text{ then } R_{x+3} := R_{x+3} - 1 \text{ and } L_m \text{ else } L_m \]

\[ \underline{\text{Notice: If either } R_i \neq R_j \text{ or } R_j \neq R_i \text{ is NOT } 0, \text{ then}} \]
\[ \underline{\text{we go to } L_m} \]

\[ \underline{\text{But if } R_i \div R_j = 0 \text{ (so } j \geq i) \text{ and } R_j \div R_i = 0 \text{ (so } i \geq j) \]
\[ \underline{\text{then } R_i = R_j \text{ and we go to } L_m}} \]
We use one auxiliary node $R_*$

\[ L^*_L : R_* = R_* + 1 \]

\[ L^*_L+1 : \text{If } R_* \neq 0 \text{ then } R_* = R_* - 1 \text{ and } L^*_L. \]

Finally we can return to our goal: Suppose $g : \mathbb{N} \to \mathbb{N}$ computed by $M$ with $i$ registers

$h : \mathbb{N}^3 \to \mathbb{N}$ computed by $\hat{M}$ with $j$ registers

and $L = \max \{ i, j^3 + 2 + \}$

where $? = \# \text{ of aux. registers we need.}$

We give a register machine with $L$ registers to calculate $f : \mathbb{N}^2 \to \mathbb{N}$ given by primitive recursion on $g, h$. To calculate $f(n, m)$ we are going to calculate $f(n, 0) = g(n)$, then $f(n, 1) = h(n, 0, f(n, 0))$, then $f(n, 2) = h(n, 1, f(n, 1))$, $\cdots$ until we reach $f(n, m)$. 
To give you an intuitive idea of the use of our registers:

- We use $R_i$, $R_{i-1}$ to do our "g" and "h" calculation. (OK since $i \geq \max(j, j_3)$).

- We use $R_e$ to hold $n$
  
  $R_{e+1}$ to hold $m$

And $R_{e+2}$ to be a counter so that when $R_{e+2} = n$

It means we are currently calculating $F(n, r)$.

We increment $R_{e+1}$ beginning with 0 and when we get to having $R_{e+1} = R_e$ we know we are finally calculating $F(n, m)$ since $R_e = m$. Throughout our calculation,

$$\begin{align*}
R_{e+2} & \leq R_{e+1} \\
(i.e. \ R_{e+2} \ \text{will count up from 0 to } R_{e+1})
\end{align*}$$

So... our original setup is

- $R_0 = n$
- $R_1 = m$
- $R_2 = 0$ for all $g > 1$. 


\( L_0 \): Copy \( R_0 \) to \( R_1 \) 

At this point \( R_0 = R_e = n \).

\( L_1 \): Move \( R_1 \) to \( R_{e+1} \)

\( R_1 = 0, R_2 = 0, \ldots, R_{e-1} = 0, R_{e+1} = m \) and \( R_{e+2} = 0 \).

\( L_2^* \): Use \( M \) to calculate \( g(n) \)

Using \( R_0, \ldots, R_{e-1} \). Replace

HALTS in \( M \) by \( L_3^* \)

When this ends we have

\( R_0 = g(n) = f(n, 0) \)
\( R_e = n \)
\( R_{e+1} = m \) and \( R_{e+2} = 0 \)

\( L_3^* \): Compose \( R_{e+1} \) and \( R_{e+2} \).

If \( R_{e+1} = R_{e+2} \), then \( L_4^* \), else \( L_5^* \) (and remember this means \( R_{e+2} < R_{e+1} = m \)).

\( L_4^* \): HALT.

\( L_5^* \): Erase \( R_1, \ldots, R_{e-1} \).

\( L_6^* \): Move \( R_0 \) to \( R_2 \) [So by induction, \( R_2 \) now contains \( f(n, R_{e+2}) \)]

\( L_7^* \): Copy \( R_2 \) to \( R_0 \) [So \( R_0 = R_e = n \)]

\( L_8^* \): Copy \( R_{e+2} \) to \( R_1 \).

\( L_9^* \): \( R_{e+2} = R_{e+2} + 1 \)

\( L_{10}^* \): Use \( M \) to calculate \( h(n, R_e, R_{e+2}) f(n, R_{e+2}) \)
Which we are set up to do using \( R_0, \ldots, R_{k-1} \).

Since \( R_0 = n \) and \( R_1 = R_{1,2} \), \( R_2 = f(n, \text{old } R_{1,2}) \).

Replace HALT in \( A \) by Goto \( L_3 \).

This completes the program. You should convince yourself it is correct! This result opens up many new register machine computable functions.

**Example:** Bounded Sums: If \( f(x) \) is register machine computable, then so is \( \sum_{y \leq x} f(y) \).

Why? Define \( \sum_{y \leq x} f(y) \) from \( f \) by primitive recursion!

\[
\sum_{y \leq 0} f(y) = f(0) \\
\sum_{y \leq x+1} f(y) = \sum_{y \leq x} f(x) + f(x+1)
\]
Exercise 7. Show register machine computable functions are closed under bounded products. That is, if $f(x)$ is register machine computable, so is $\forall y \leq x \exists ! f(y)$.

The μ operation

Def: If $f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ is a partial function, then $\mu y f(x,y)$ [where $|x| = k$] is defined by:

$\mu y f(x,y) = \text{the least } y \in \mathbb{N} \text{ such that } f(x,y) = 0 \text{ and } f(x,z) \text{ is defined but not } = 0 \text{ for all } z < y$. 
Exercise: Prove that if $f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ is partial register machine computable then $\mu y f(y)$ is partial register machine computable. (Just worry about $k=1$ case.)

A partial function

\[ f: \mathbb{N}^{k} \rightarrow \mathbb{N} \text{ is partial register machine computable } \iff \text{ there is a register machine } M \text{ such that } f_{M}^{k} = f. \]

Gödel used a different model of computation.

Def: The partial recursive functions are the smallest class of partial functions $f: \mathbb{N}^{k} \rightarrow \mathbb{N}$ (for $k \geq 1$) such that
It contains

1. \( I(x) = x \)
2. \( S(x) = x + 1 \)
3. \( \Pi_{i}^{K}(x) = x_i \) for all \( i \leq K \)

and is closed under the operations of

4. composition
5. primitive recursion

and (6) \( \mu \)-operator.

Notice: We have proved that

\[
\text{partial recursive} \subseteq \text{partial register machine computable}
\]

In fact, these collections are equal. In fact, every other model of computation proposed so far leads to this same class of functions! This is taken as evidence for Church's Thesis.
Church's Thesis: If $f: \mathbb{N}^k \rightarrow \mathbb{N}$ is an intuitively computable function, then $f$ is actually register machine computable. By intuitively computable, I mean you can come up with a general algorithm to compute it.

Although books and articles frequently appeal to Church's Thesis, this is essentially always out of laziness. That is, I do not know of any case when you cannot actually show the function is register machine computable with a bit of work.

Because all of these models of computation coincide, I will use term "computable" instead of "register machine computable".
**Def:** A relation $X \subseteq \mathbb{N}^k$ is called **computable** (or register machine computable) $\iff$ there is a computable function $f: \mathbb{N}^k \rightarrow \mathbb{N}$ such that for all $\bar{x} \in \mathbb{N}^k$

$$f(\bar{x}) = 1 \iff \bar{x} \in X$$
$$f(\bar{x}) = 0 \iff \bar{x} \notin X.

**Example** $X = \{ (n,m) \mid n = m^3 \leq n^2 \}$ is computable.

Use following program to calculate $f$:

**Start:** $R_0 = n$ and $R_1 = m$

$L_0$. Compare $R_0$ and $R_1 \iff$ if $R_0 + R_1$, then $L_1^*$ else $L_3^*$

$L_1^*$. Erase $R_0$

$L_2^*$. HALT

$L_3^*$. Erase $R_0$

$L_4^*$. $R_0 = R_0 + 1$

$L_5^*$. HALT.
Exercise: Show \( X = \{(n,m) \mid n < m \} \subseteq \mathbb{N}^2 \) is computable.

Our final goal in these notes is to show a much broader class of relations is computable. To do this, fix the following symbol set

\[ L = \{+, \cdot, E, 0, 1, <, \} \]

and the standard model of \( \mathbb{N} \) in which

\( +^\mathbb{N} = \text{plus}, \cdot^\mathbb{N} = \text{times}, E^\mathbb{N} = \text{exponentiation} \)

That is \( E^\mathbb{N}(n,m) = n^m \). I will usually drop the "\( \mathbb{N} \)" superscript.

Recall: \(+, \cdot, E\) are all computable.

Our next goal is to show that all sets defined by special types of formulas (called bounded quantifier formulas) in \( \mathbb{N} \) are computable.
Think about a term \( t \) in this language with variables \( x_0, x_1, \ldots, x_{k-1} \). Let \( \bar{x} \) be these variables and I will write \( t(\bar{x}) \) to indicate the variables occurring in \( t \).

If \( \bar{n} \in \text{IN}^k \), then \( t(\bar{n}) \) is the element of \( \text{IN} \) we get by plugging in \( n_0, n_1, \ldots, n_{k-1} \) for \( x_0, x_1, \ldots, x_{k-1} \) and evaluating. Since \( t(\bar{x}) \) is built from \( 0, 1, x_0, \ldots, x_{k-1} \) using \( +, \times, \cdot \) and \( E \) and since these functions are all computable and the computable functions are closed under composition, we have that the function

\[
\bar{n} \mapsto t(\bar{n})
\]

is computable.
Consider an atomic \( \Sigma \)-formula \( q(x) \). It has the form \( t_0(x) = t_1(x) \) or \( t_0(x) < t_1(x) \).

Claim: \( X = \{ \bar{n} \mid N \models q(\bar{n}) \} \) is computable.

Why? Suppose \( q(x) \) is \( t_0(x) = t_1(x) \),

Then \( f(\bar{n}) = t_0(\bar{n}) \) is computable

\( g(\bar{n}) = t_1(\bar{n}) \) is computable

So to tell if \( \bar{n} \in X \):

- Calculate \( g(\bar{n}) \) and \( f(\bar{n}) \)
- Compare \( g(\bar{n}) \) and \( f(\bar{n}) \). If = output 1,
  If \( \neq \) output 0.

Same idea for \( t_0(x) < t_1(x) \).
Fix $L$-formulas $\varphi(x)$ and $\psi(x)$ such that
\[ X = \{ \bar{\eta} \mid N \models \varphi(\bar{\eta}) \} \]
\[ Y = \{ \bar{\eta} \mid N \models \psi(\bar{\eta}) \} \] are computable.

Then \[ X \cap Y = \{ \bar{\eta} \mid N \models (\varphi \land \psi)(\bar{\eta}) \} \] is computable.

Why? Suppose $f(x)$, $g(x)$ are computable so that
\[ \bar{\eta} \in X \iff f(\bar{\eta}) = 1 \; ; \; \bar{\eta} \in X \iff f(\bar{\eta}) = 0 \]
\[ \bar{\eta} \in Y \iff g(\bar{\eta}) = 1 \; ; \; \bar{\eta} \in Y \iff g(\bar{\eta}) = 0 \]

Let $h(x) = f(x) \cdot g(x)$. Then
\[ \bar{\eta} \in X \cap Y \iff f(\bar{\eta}) = g(\bar{\eta}) = 1 \iff h(\bar{\eta}) = 1 \]
\[ \bar{\eta} \in X \cap Y \iff f(\bar{\eta}) = 0 \text{ or } g(\bar{\eta}) = 0 \iff h(\bar{\eta}) = 0 \]

The same idea works for
\[ X \cup Y = \{ \bar{\eta} \mid N \models (\varphi \lor \psi)(\bar{\eta}) \} \]
\[ \overline{X} = \{ \bar{\eta} \mid N \not\models \varphi(\bar{\eta}) \} \]
If \( t \) is a term and \( \varphi(x) \) is a formula, we define the \underline{bounded quantifiers} formula as:

\[
\exists x < t \varphi(x) \sim \exists x (x < t \land \varphi(x))
\]

\[
\forall x < t \varphi(x) \sim \forall x (x < t \rightarrow \varphi(x)).
\]

\[
\text{Notice: } \mathcal{N} = \exists x < t \varphi(x) \Rightarrow \mathcal{N} = \varphi(n) \text{ for some } n < t^{\text{th}}
\]

\[
\mathcal{N} = \forall x < t \varphi(x) \Rightarrow \mathcal{N} = \varphi(n) \text{ for all } n < t^{\text{th}}.
\]

Suppose \( \varphi(x,y) \) is a formula such that

\[
\{ (n,m) \mid \mathcal{N} = \varphi(n,m) \} \text{ is computable.}
\]

Then for any term \( t(x,y) \) the set

\[
\{ (n,m) \mid \mathcal{N} = \forall x < t(n,m) \varphi(x,m) \} \text{ is computable.}
\]
Why? Let \( f(x,y) \) be computable function such that

\[
\text{INF} = \varphi(n,m) \Leftrightarrow f(n,m) = 1 \quad \text{and} \quad \text{NAF} \varphi(n,m) \Leftrightarrow f(n,m) = 0,
\]

Let \( g(n,m) = t(n,m) \) be computable (via Step 1).

Let \( h(x,y) = \prod_{x < z} f(x,y) \) be bounded predicate which we know is computable.

Now by composition, \( \hat{f}(x,y) = h(g(x,y), y) \) is computable and

\[
\hat{f}(n,m) = \prod_{x < g(n,m)} f(x,m)
\]

and you can check:

\[
\text{INF} \equiv \forall x < t(n,m) \varphi(x,m) \Leftrightarrow \text{for all } x < g(n,m) \ f(x,m) = 1
\]

\[
\Leftrightarrow \prod_{x < g(n,m)} f(x,m) = 1 \Leftrightarrow \hat{f}(n,m) = 1
\]

and similarly \( \text{NAF} \equiv \forall x < t(n,m) \varphi(x,m) \Leftrightarrow \hat{f}(n,m) = 0. \)
Exercise: Use bounded sums to prove that

If $\exists (n,m) \mid \mathcal{N} \models \varphi(n,m)$ is computable

then $\exists (n,m) \mid \mathcal{N} \models \exists x \leq t(n,m) \varphi(x,m)$ is computable.

Definition: The class of bounded quantifier $\mathcal{L}$-formulas is defined by induction:

1. All atomic formulas are bounded quantifier formulas.
2. If $\varphi$ and $\psi$ are bounded quantifier formulas, then so are $\top \varphi$, $\varphi \land \psi$, $\varphi \lor \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$.
3. If $\varphi$ is a bounded quantifier formula and $t$ is any term, then $\exists x < \!t \varphi$ and $\forall x < \!t \varphi$ are bounded quantifier formulas.

So as usual, the class of bounded quantifier formulas is built by starting with (1) and closing under (2) and (3).
Notation: We say "\( \varphi \) is a \( \Sigma_0 \) formula" to mean "\( \varphi \) is a bounded quantifier formula". Later this notation will make more sense. We will be interested in the sets which can be defined by \( \Sigma_0 \) formulas.

Example: The set \( X \subseteq \mathbb{N} \) consisting of even numbers is defined by a \( \Sigma_0 \) formula:

\[
X = \{ n \in \mathbb{N} \mid n \text{ is even}\} \\
= \{ n \in \mathbb{N} \mid \exists x \leq n (2x=n)\}
\]

So if \( \varphi(y) \) is \( \exists x \leq y (2x=y) \)
we have \( n \in X \Leftrightarrow \mathbb{N} \models \varphi(n) \)

Example: The formula

\[
\text{Div}(x,y) \sim \exists z \leq y (x \cdot z = y)
\]

is a \( \Sigma_0 \) definition for the set

\[
\{ (n,m) \mid n \text{ divides } m \}
\]
Exercise: Write a $\Sigma^0_3$ formula $\text{Prime}(y)$ such that

$$\{ p \in \mathbb{N} \mid p \text{ is prime} \} = \{ p \in \mathbb{N} \mid \mathbb{N} \models \text{Prime}(p) \}$$

Thm: If $\varphi(x)$ is a $\Sigma^0_3$ formula, then the set $\{ n \mid \mathbb{N} \models \varphi(n) \}$ is computable.

Pf: We proceed by induction on the definition of $\Sigma^0_3$ formulas.

Base Case: If $\varphi(x)$ is atomic then show $\{ n \mid \mathbb{N} \models \varphi(n) \}$ is computable. This is exactly what we did in Step 2!

Inductive Cases: Assume $\varphi(x), \psi(x)$ are $\Sigma^0_3$ and we know $\{ n \mid \mathbb{N} \models \varphi(n) \}$ and $\{ n \mid \mathbb{N} \models \psi(n) \}$ are computable.
We need to show that sets defined by 
\( \varphi(x) \vee \psi(x) \), \( \varphi(x) \vee \psi(x) \), \( \neg \varphi(x) \), \( \cdots \) are computable. This is what we did in Step 3!

Also, if \( t(x) \) is a term, we need to show the sets defined by \( \exists x_0 < t(x) \varphi(x_0, x_1, \ldots, x_{k-1}) \) and
\( \forall x_0 < t(x) \varphi(x_0, x_1, \ldots, x_{k-1}) \) are computable. This is what we did in Step 4.

This theorem is quite powerful — it allows us to say right away that things like the set of prime numbers is computable. In general, we will move towards "intuitive" explanations of why certain sets are computable, but frequently, one can typically formalize these "intuitive" explanations using this theorem. (Note: The converse of this theorem is not true.)
Let me illustrate these ideas about computable functions and relations with one more application that is very useful for coding: we will do later.

**Lemma:** Let $q(x,y)$ be a formula such that $X = \{ (n,m) \mid \text{NT} \models q(n,m) \}$ is computable and such that

$\text{NT} \models \forall x \exists! y \; q(x,y)$ where $\exists!$ = "there exists a unique".

Then the function $f(n) = \text{unique } m \text{ s.t. NT} \models q(n,m)$ is computable.

**Pf:** Let $f(n,m)$ be computable function such that

$(n,m) \in X \iff f(n,m) = 1$ and

$(n,m) \notin X \iff f(n,m) = 0$.

Then since computable functions are closed under the $\mu$-operator, the function

$$g(x) = \mu y \left( f(x,y) = 1 \right)$$

is computable and

$$g(n) = m \iff f(n,m) = 1 \iff \text{NT} \models q(n,m).$$
Def: A partial function \( f: \mathbb{N}^k \to \mathbb{N} \) is \textit{partial computable} \( \iff \) there is a register machine \( M \) such that \( f^k_M = f \). If \( f: \mathbb{N}^k \to \mathbb{N} \) is partial computable and total (i.e., \( \text{domain}(f) = \mathbb{N}^k \)) then \( f \) is called \textit{computable}. We will focus on case when \( k = 1 \).

\[ \text{Question} \quad \text{How many partial computable functions are there? Since each register machine is given by a finite number of registers and a finite program, there are only countably many register machines. Therefore, there are only countably many partial computable functions.} \]

\[ \text{Notation: We list the partial computable functions of one variable as} \]

\[ \varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots \]

One of the key facts is that we can make this list effectively in the following sense.