D

Number Register Machines

A <u>number register machine</u> 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.)

<u>Convention</u>: IF there is no ELSE clause, then if $R_K = 0$ just go to the next instruction.

2)

If a register machine M has registers $R_0, \dots, R_n$ and instructions $L_0, \dots, L_m$ we can start it with whatever initial values in $R_0, \dots, R_n$ (from $\mathbb{N}$) we want. We call these initial values the <u>input</u>. If M eventually reaches a HALT instruction, then we view the contents of register R_0 as the <u>output</u> of M on the given input.

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

$$f_M^k : \mathbb{N}^k \to \mathbb{N}$$

by $f_M^k(n_0, n_1, \dots, n_{k-1})$ is calculated by starting M with $R_0 = n_0$, $R_1 = n_1, \dots, R_{k-1} = n_{k-1}$, $R_k = 0, \dots, R_n = 0$ and setting $f_M^k(n_0, n_1, \dots, n_{k-1}) =$ output of M when it reaches HALT instruction with these inputs.

WARNING: M might not reach HALT instruction!

Example 1. Let M have one register M_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 $f_M^k : \mathbb{N}^k \to \mathbb{N}$ defined by register machines are partial functions. This means their domain is

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

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 $L0$.

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

R_0	n
R_1	m

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

Therefore, $f_M^2 : \mathbb{N}^2 \to \mathbb{N}$ is $f_M^2(n,m) = n+m$.

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

So, addition is register machine computable.

Exercise 1 Prove that

$$f(n,m) = \begin{cases} n-m & \text{if } m \le n \\ 0 & \text{otherwise} \end{cases}$$

is register machine computable. We denote $f(n,m)$ by $n \dot{-} m$. (So $n \dot{-} m = n - m$ if $m \le n$ and $n \dot{-} m = 0$ 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_i^*.

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 $f_M^2 : \mathbb{N}^2 \to \mathbb{N}$.)

R_0	n
R_1	m
R_2	0
R_3	0

L_0^*: Add R_0 to R_2 [really this means to 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_1.

L_6: HALT.

You should convince yourself that using M we get

$$f_M^2(n,m) = n \cdot m$$

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 $R_1 := R_1 + R_0$!

$\boxed{\text{Exercise 2}}$ Show that $g(n,m) = n^m$ is register machine computable.

$\boxed{\text{Example 4}}$ The composition of partial 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.

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

We want machine to compute $g(f(n))$

We build M^* with $i+k+1$ registers and the following program:

R_0	n	We will use for "f" part
$\vdots$	0 $\vdots$	
R_{i-1}	0	
R_i	0	We will use for "g" part
R_{i+1}	0	
$\vdots$	$\vdots$	
R_{i+k-1}	0	
R_{i+k}	0	← We will use for "Go to" statement.

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

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

L_{j+1}^*: Add R_0 to R_{i+1}

L_{j+2}: ... L_{j+l+2}: } Put in instructions from $\hat{M}$ for g (but relabelled as $L_{j+2}, \dots, L_{j+l+2}$ instead of $L_0, \dots, L_{l-1}$ and as using registers $R_i, \dots, R_{i+k-1}$ instead of $R_0, \dots, R_{k-1}$). Replace HALT by "IF $R_{i+k} \neq 0$ then $R_{i+k} = R_{i+k} - 1$ and L^*_{j+l+3}".

L^{*}_{j+l+3} : Add R_i to R_0.

L_{j+l+4} : HALT.

Convince yourself that $\hat{M}$ computes $g(f(n))$.

$\boxed{\text{Exercise 3}}$ 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).

$\boxed{\text{Exercise 4}}$ Show that for each $K \geq 1$ and $1 \leq i \leq K$ the function $\pi_i^K : \mathbb{N}^K \to \mathbb{N}$ given by

$$\pi_i^K(n_1, n_2, \ldots, n_K) = n_i$$ is register machine

computable.

Primitive Recursion

<u>Def:</u> 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 <u>primitive recursion</u> from g and $h \iff$

$$f(\bar{x}, 0) = g(\bar{x}) \quad \text{for } \bar{x} \in \mathbb{N}^k$$

and $$f(\bar{x}, y+1) = h(\bar{x}, y, f(\bar{x}, y))$$

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 $\mathbb{N}$).

$\boxed{\text{Prim. Rec. Ex. 1}}$ We can define addition by primitive recursion from $g(x) = x$ and $h(x,y,z) = z+1$

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

$$f(x,y+1) = h(x,y,f(x,y)) = h(x,y,x+y)$$ by induction hypothesis

$$= (x+y)+1 = x+(y+1).$$

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

$\boxed{\text{Exercise 5}}$ Show multiplication is defined by primitive recursion using $g(x) = 0$ and $h(x,y,z) = x + z$.

Show the exponentiation is defined by prim. rec. using $g(x) = 1$ and $h(x,y,z) = x \cdot z$.

1

Our next goal is to show that if

$g: \mathbb{N}^k \to \mathbb{N}$ and $h: \mathbb{N}^{k+2} \to \mathbb{N}$ are register machine computable, then so is $f: \mathbb{N}^{k+1} \to \mathbb{N}$ defined by primitive recursion from g, h. To do this, we introduce some new "psuedo-instructions":

$\boxed{\text{Erase } R_i}$ This means set $R_i = 0$ as in previous exercise. Notice we can do this with instruction of form:

$$L_j: \text{ If } R_i \neq 0 \text{ then } R_i = R_{i-1} \text{ and } L_j \text{ else } L_{j+1}.$$

$\boxed{\text{Move } R_i \text{ to } R_j}$ This means Erase R_j and use our old "add" program to add R_j to R_i. In the end we have

New R_j = Old R_i

New R_i = 0.

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

L^*_ℓ. Erase R_j and R_*.

$L^*_{\ell+1}$. Move R_i to both R_j and R_*.

$L^*_{\ell+2}$: Move R_* to R_i.

So we get New R_j = Old R_i

New R_i = Old R_i

New R_* = 0

$\boxed{\text{Compare } R_i, R_j; \text{ If } R_i \neq R_j \text{ then } \underline{L_m} \text{ else } \underline{L_n}}$

For this psuedo-instruction we will use 4 auxiliary registers $R_*, R_{*+1}, R_{*+2}, R_{*+3}$.

(13) Our instruction to do compare operation look like:

L^*_l: Copy R_i to R_* and R_{*+1}

L^*_{l+1}: Copy R_j to R_{*+2} and R_{*+3}

At this point we have
$R_* = R_i$
$R_{*+1} = R_i$
$R_{*+2} = R_j$
$R_{*+3} = R_j$

L^*_{l+2}: Use our subtraction program to get $R_* := R_* \dot{-} R_{*+2}$

L^*_{l+3}: Use our subtraction program to get $R_{*+3} = R_{*+3} \dot{-} R_{*+2}$

At this point we have
$R_* = R_i \dot{-} R_j$
$R_{*+3} = R_j \dot{-} R_i$

L^*_{l+4}: If $R_* \neq 0$ then $R_* = R_* - 1$ and L_m else L^*_{l+5}

L^*_{l+5}: If $R_{*+3} \neq 0$ then $R_{*+3} = R_{*+3} - 1$ and L_m else L_n.

Notice: If either $R_i \dot{-} R_j$ or $R_j \dot{-} R_i$ is NOT 0, then we go to L_m.

But if $R_i \dot{-} R_j = 0$ (so $j \geq i$) and $R_j \dot{-} R_i = 0$ (so $i \geq j$) then $R_i = R_j$ and we go to L_n.

$\boxed{M}$

$\boxed{\text{Goto } L_j}$ We use one auxiliary node R_*

$L^*_\ell : R_* = R_* + 1$

$L^*_{\ell+1} :$ If $R_* \neq 0$ then $R_* = R_* - 1$ and L_j.

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 $\quad \ell = \max\{i, j\} + 2 + ?$

where $? = \#$ 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 $f(n,2) = h(n,1, f(n,1)), \ldots$ until we reach $f(n,m)$.

15 To give you an intuitive idea of the use of our registers:

- We use $R_0, \ldots, R_{\ell-1}$ to do our "g" and "h" calculation. (OK since $\ell \geq \max\{i,j\}$)

- We use R_ℓ to hold n

  $R_{\ell+1}$ to hold m

and $R_{\ell+2}$ to be a counter so that when $R_{\ell+2} = r$ it means we are currently calculating $f(n,r)$.

We increment $R_{\ell+1}$ beginning with 0 and when we get to having $R_{\ell+1} = R_\ell$ we know we are finally calculating $f(n,m)$ since $R_\ell = m$. Throughout our calculation, $R_{\ell+2} \leq R_{\ell+1}$ (i.e. $R_{\ell+2}$ will count up from 0 to $R_{\ell+1}$.)

So... our original set up is

$R_0 = n$

$R_1 = m$

$R_q = 0$ for all $q > 1$.

L_0^*. Copy R_0 to R_ℓ

L_1^*. Move R_1 to $R_{\ell+1}$

$\left\{\begin{array}{l} \text{At this point } R_0 = R_\ell = n, \\ R_1 = 0, R_2 = 0, \ldots, R_{\ell-1} = 0, \\ R_{\ell+1} = m \text{ and } R_{\ell+2} = 0. \end{array}\right.$

L_2^*: Use M to calculate $g(n)$ using $R_0, \ldots, R_{\ell-1}$. Replace HALTS in M by Goto L_3^*

$\left\{\begin{array}{l} \text{When this ends we have} \\ R_0 = g(n) = f(n,0) \\ R_\ell = n \\ R_{\ell+1} = m \text{ and } R_{\ell+2} = 0 \end{array}\right.$

L_3^*: Compare $R_{\ell+1}$ and $R_{\ell+2}$.
If $R_{\ell+1} = R_{\ell+2}$ then L_4^*, else L_5^* (and remember this means $R_{\ell+2} < R_{\ell+1} = m$).

L_4^*: HALT.

L_5^*: Erase $R_1, \ldots, R_{\ell-1}$.

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

L_7^*: Copy R_ℓ to R_0 [So $R_0 = R_\ell = n$]

L_8^*: Copy $R_{\ell+2}$ to R_1

L_9^*: $R_{\ell+2} = R_{\ell+2} + 1$

L_{10}^*: Use $\hat{M}$ to calculate $h(n, \overset{\text{old}}{R_{\ell+2}}, f(n, \overset{\text{old}}{R_{\ell+2}}))$

Which we are set up to do using $R_0, \ldots, R_{\ell-1}$

Since $R_0 = n$ and $R_1 = \overset{old}{R_{\ell+2}}$, $R_2 = f(n, \overset{old}{R_{\ell+2}})$

Replace HALTs in $\hat{M}$ 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:

$\boxed{\text{Example}}$ Bounded Sums: If $f(x)$ is register machine computable, then so is $\sum\limits_{y \le x} f(y)$.

Why? Define $\sum\limits_{y \le x} f(y)$ from f by primitive recursion!

$$\sum_{y \le 0} f(y) = f(0)$$

$$\sum_{y \le x+1} f(y) = \sum_{y \le x} f(x) + f(x+1)$$

<u>Think</u>: What are g and h here?

$\boxed{\text{Exercise}}$ Show register machine computable functions are closed under bounded products. That is, if $f(x)$ is register machine computable, so is $\prod_{y \le x} f(y)$.

$\boxed{\text{The } \mu \text{ operation}}$

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

$\mu y\, f(\bar{x}, y)$ = the least $y \in \mathbb{N}$ such that $f(\bar{x}, y) = 0$ and $f(\bar{x}, z)$ is <u>defined</u> but not $\neq 0$ for all $z < y$.

Exercise Prove that if $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is partial register machine computable then $\mu y\, f(\bar{x})$ is partial register machine computable.
(Just worry about $k=1$ case.)

---

2 Models of Computation

Register Machines: A partial function $f : \mathbb{N}^k \to \mathbb{N}$ is <u>partial register machine computable</u> $\iff$ there is a register machine M 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 $f : \mathbb{N}^k \to \mathbb{N}$ (for $k \geq 1$) such that

It contains

1. $I(x) = x$
2. $S(x) = x+1$
3. $\pi_i^k(\bar{x}) = x_i$ for all $i \le k$

and is closed under the operations of

4. composition
5. primitive recursion

and 6. μ-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 $f : \mathbb{N}^k \to \mathbb{N}$ 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".

Computable Relations

Def: A relation $X \subseteq \mathbb{N}^k$ is called Computable (or register machine computable) $\Longleftrightarrow$ there is a computable function $f: \mathbb{N}^k \to \mathbb{N}$ such that for all $\bar{x} \in \mathbb{N}^k$

$$f(\bar{x}) = 1 \Longleftrightarrow \bar{x} \in X$$

$$f(\bar{x}) = 0 \Longleftrightarrow \bar{x} \notin X.$$

Example $X = \{(n,m) \mid n = m\} \subseteq \mathbb{N}^2$ is computable.

Use following program to calculate f:

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

L_0^*. Compare R_0 and R_1; if $R_0 \neq R_1$ 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.

$\boxed{\text{Exercise}}$ 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

$$\mathcal{L} = \{+, \cdot, E, 0, 1, <\}$$

binary function symbol

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

$+^{\mathbb{N}}$ = plus, $\cdot^{\mathbb{N}}$ = times, $E^{\mathbb{N}}$ = exponentiation

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

<u>Recall:</u> $+, \cdot, E$ are all computable.

Our next goal is to show that all sets defined by special types of $\mathcal{L}$-formulas (called bounded quantifier formulas) in $\mathbb{N}$ are computable.

$\boxed{\text{Step 1}}$

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 occuring in t.

If $\bar{n} \in \mathbb{N}^k$, then $t(\bar{n})$ is the element of $\mathbb{N}$ 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 $+, \cdot, E$ and since these functions are all computable <u>and</u> the computable functions are closed under composition we have that the function

$$\bar{n} \longmapsto t(\bar{n}) \quad \text{is computable.}$$

$\boxed{\text{Step 2}}$

Consider an atomic $\mathcal{L}$-formula $\varphi(\bar{x})$. It has the form $t_0(\bar{x}) = t_1(\bar{x})$ or $t_0(\bar{x}) < t_1(\bar{x})$.

<u>Claim</u>: $X = \{ \bar{n} \mid \mathbb{N} \models \varphi(\bar{n}) \}$ is computable.

<u>Why?</u> Suppose $\varphi(\bar{x})$ is $t_0(\bar{x}) = t_1(\bar{x})$.

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 $t_0(\bar{x}) < t_1(\bar{x})$.

$\boxed{\text{Step 3}}$

Fix $\mathcal{L}$-formulas $\varphi(\bar{x})$ and $\psi(\bar{x})$ such that

$$X = \{\bar{n} \mid \mathbb{N} \models \varphi(\bar{n})\}$$
$$Y = \{\bar{n} \mid \mathbb{N} \models \psi(\bar{n})\}$$

are computable.

Then $X \cap Y = \{\bar{n} \mid \mathbb{N} \models (\varphi \wedge \psi)(\bar{n})\}$ is computable.

<u>Why?</u> Suppose $f(\bar{x}), g(\bar{x})$ are computable so that

$$\bar{n} \in X \Leftrightarrow f(\bar{n}) = 1 \ ; \quad \bar{n} \notin X \Leftrightarrow f(\bar{n}) = 0$$
$$\bar{n} \in Y \Leftrightarrow g(\bar{n}) = 1 \ ; \quad \bar{n} \notin Y \Leftrightarrow g(\bar{n}) = 0$$

Let $h(\bar{x}) = f(\bar{x}) \cdot g(\bar{x})$. Then

$$\bar{n} \in X \cap Y \Leftrightarrow f(\bar{n}) = g(\bar{n}) = 1 \Leftrightarrow h(\bar{n}) = 1$$
$$\bar{n} \notin X \cap Y \Leftrightarrow f(\bar{n}) = 0 \text{ or } g(\bar{n}) = 0 \Leftrightarrow h(\bar{n}) = 0$$

The same idea works for

$$\left.\begin{array}{l} X \cup Y = \{\bar{n} \mid \mathbb{N} \models (\varphi \vee \psi)(\bar{n})\} \\ \overline{X} = \{\bar{n} \mid \mathbb{N} \models \neg\varphi(\bar{n})\} \end{array}\right\} \text{CHECK THESE}$$

<u>EXERCISE</u>

Step 4

If t is a term and $\varphi(x)$ is a formula, we define the bounded quantifiers formulas as:

$$\exists x < t\ \varphi(x) \sim \exists x (x < t \wedge \varphi(x))$$

$$\forall x < t\ \varphi(x) \sim \forall x (x < t \rightarrow \varphi(x)).$$

Notice: $\mathbb{N} \models \exists x < t\ \varphi(x) \iff \mathbb{N} \models \varphi(n)$ for some $n < t^{\mathbb{N}}$

$\mathbb{N} \models \forall x < t\ \varphi(x) \iff \mathbb{N} \models \varphi(n)$ for all $n < t^{\mathbb{N}}$.

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

$\{ (n,m) \mid \mathbb{N} \models \varphi(n,m) \}$ is computable.

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

$\{ (n,m) \mid \mathbb{N} \models \forall x < t(n,m)\ \varphi(x,m) \}$ is computable.

<u>Why?</u> Let $f(x,y)$ be computable function such that

$\mathbb{N} \models \varphi(n,m) \iff f(n,m) = 1$ and $\mathbb{N} \not\models \varphi(n,m) \iff f(n,m) = 0$.

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

Let $h(z,y) = \prod_{x<z} f(x,y)$ be bounded product 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:

$\mathbb{N} \models \forall x < t(n,m)\, \varphi(x,m) \iff$ for all $x < g(n,m)$ $f(x,m) = 1$

$\iff \prod_{x<g(n,m)} f(x,m) = 1 \iff \hat{f}(n,m) = 1$

and similarly $\mathbb{N} \not\models \forall x < t(n,m)\, \varphi(x,m) \iff \hat{f}(n,m) = 0$.

Exercise Use bounded sums to prove that

if $\{(n,m) \mid \mathbb{N} \models \varphi(n,m)\}$ is computable

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

Def: The class of <u>bounded</u> <u>quantifier</u> <u>$\mathcal{L}$-formulas</u> is defined by induction:

1. All atomic formulas are bounded quantifier formulas
2. If φ and ψ are bounded quantifier formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$, $\varphi \leftrightarrow \psi$.
3. If φ is 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 "φ is a Σ_0^0 formula" to mean "φ is a bounded quantifier formula". Later this notation will make more sense. We will be interested in the sets which can be defined by Σ_0^0 formulas.

$\boxed{\text{Example}}$ The set $X \subseteq \mathbb{N}$ consisting of even numbers is defined by a Σ_0^0 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 $n \in X \Leftrightarrow \mathbb{N} \models \varphi(n)$

$\boxed{\text{Example}}$ The formula

$$Div(x,y) \sim \exists z \leq y \, (x \cdot z = y)$$

is a Σ_0^0 definition for the set

$$\{ \langle n,m \rangle \mid n \text{ divides } m \}$$

$\boxed{\text{Exercise}}$ Write a Σ^0_0 formula 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 $\varphi(\bar{x})$ is a Σ^0_0 formula, then the set

$$\{\bar{n} \mid \mathbb{N} \models \varphi(\bar{n})\}$$ is computable.

Pf: We proceed by induction on the definition of Σ^0_0 formulas.

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

Inductive Cases: Assume $\varphi(\bar{x})$, $\psi(\bar{x})$ are Σ^0_0 and we know $\{\bar{n} \mid \mathbb{N} \models \varphi(\bar{n})\}$ and $\{\bar{n} \mid \mathbb{N} \models \psi(\bar{n})\}$ are computable.

We need to show that sets defined by

$\varphi(\bar{x}) \wedge \psi(\bar{x})$, $\varphi(\bar{x}) \vee \psi(\bar{x})$, $\neg\varphi(\bar{x})$, ... are

computable. This is what we did in Step 3!

Also, if $t(\bar{x})$ is a term, we need to show the

sets defined by $\exists x_0 < t(\bar{x})\ \varphi(x_0, x_1, \ldots, x_{k-1})$ and

$\forall x_0 < t(\bar{x})\ \varphi(x_0, x_1, \ldots, x_{k-1})$ 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 those "intuitive" explanation using this theorem. (<u>Note</u>: The converse of this theorem is <u>not</u> 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 $\varphi(x,y)$ be a formula such that $X = \{(n,m) \mid \mathbb{N} \models \varphi(n,m)\}$ is computable and such that $\mathbb{N} \models \forall x\, \exists!\, y\, \varphi(x,y)$ where $\exists!$ = "there exists a unique".
Then the function $f(n)$ = unique m s.t. $\mathbb{N} \models \varphi(n,m)$ 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 μ-operator, the function

$$g(x) = \mu y\,(f(x,y) = 1)$$ is computable and

$$g(n) = m \iff f(n,m) = 1 \iff \mathbb{N} \models \varphi(n,m). \quad ※$$

Computability Theory

<u>Def</u>: A partial function $f: \mathbb{N}^k \to \mathbb{N}$ is <u>partial computable</u> $\Leftrightarrow$ there is a register machine M such that $f_M^k = f$. If $f: \mathbb{N}^k \to \mathbb{N}$ is partial computable and <u>total</u> (i.e. $\text{domain}(f) = \mathbb{N}^k$) then f is called <u>computable</u>. We will focus on case when $k = 1$.

$\boxed{\text{Question}}$ 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.

<u>Notation</u>: We list the partial computable functions of one variable as

$\varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots$ ~~We write $\varphi_e(x)\downarrow$ and say $\varphi_e(x)$ halts, if x is in the domain of φ_e~~

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