CHAPTER 5

# What's a computer, anyway? (Cont'd)

## 2. Recursive functions are defined one step at a time

**2.1. The Basics.** Now that we've talked about models of computation, we'd like to figure out what kinds of functions they can compute. Functions will be at the heart of our discussion, so I'd like to have some handy notation for them. Let's fix some.

For $n \in \mathbb{N}$ we will write $\mathcal{F}_n$ for the set $\{f : \mathbb{N}^n \to \mathbb{N} : f \text{ is a function}\}$, and we will write $\mathcal{F}$ for $\bigcup_{n \in \mathbb{N}} \mathcal{F}_n$. I'll be using a little bit of **$\lambda$-calculus** notation in this chapter. Namely, if $f \in \mathcal{F}_n$, then I'll write:

$$f = \lambda x_1 \cdots x_n . f(x_1, \ldots, x_n).$$

If this notation is confusing to you, don't worry too much about it. In any case, the main advantage it has is that it easily allows us to see functions of multiple arguments as sequences of functions of one argument. For example:

$$\lambda x_1 x_2 . f(x_1, x_2) = \lambda x_1 . (\lambda x_2 f(x_1, x_2))$$

and that it allows us to pass functions to other functions as variables (i.e. it makes writing composition of functions easy). For example:

$$(\lambda x . f(x))(\lambda x . g(x)) = \lambda x . f(g(x)).$$

Anyway, whatever.

The most basic functions in $\mathcal{F}$ are the following:

- The **successor function** $S = \lambda x . x + 1$.
- The **nullary constant function** $C_0^0$ which is always equal to 0, for notation's sake $C_0^0 = \lambda x . 0$.
- For every $n \in \mathbb{N}$ and every $i \leq n$ the function $P_i^n$, that given an $n$-tuple returns its $i$-th coordinate, we call these the **projection functions**. In our $\lambda$ notation:
$$P_i^n = \lambda x_1 \ldots x_n . x_i.$$

The set of **basic functions** $\mathcal{B} \subseteq \mathcal{F}$ is the following set:

$$\mathcal{B} := \{S\} \cup \{C_0^0\} \cup \left( \bigcup_{n \in \mathbb{N}} \{P_i^n : i \leq n\} \right).$$

**Exercise 2.1.1.** Show that if $f \in \mathcal{B}$, then $f$ is register-machine computable.

### 2.2. Primitive Recursive Functions.

**Definition 2.2.1.** The set of *primitive recursive functions*, $\mathcal{E} \subseteq \mathcal{F}$, is defined by induction, as follows:

(1) $\mathcal{B} \subseteq \mathcal{E}$.

(2) If $f_1, \ldots, f_m \in \mathcal{E} \cap \mathcal{F}_n$ and $g \in \mathcal{F}_m \cap \mathcal{E}$, then the function $h$ defined by:
$$h : \mathbb{N}^n \to \mathbb{N}$$
$$(x_1, \ldots, x_n) \mapsto g(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)).$$
is also in $\mathcal{E}$.

(3) If $f \in \mathcal{F}_n \cap \mathcal{E}$, $g \in \mathcal{F}_{n+2} \cap \mathcal{E}$, then the function $h$ defined by:
$$h : \mathbb{N}^{k+1} \to \mathbb{N}$$
$$(x_1, \ldots, x_{k+1}) \mapsto \begin{cases} f(x_1, \ldots, x_k) & \text{if } x_{k+1} = 0, \\ g(x_1, \ldots, x_k, x_{k+1} - 1, h(x_1, \ldots, x_k, x_{k+1} - 1)) & \text{otherwise} \end{cases}$$
is also in $\mathcal{E}$.

(4) That's it.

Let's fix some terminology/notation:

- If $\mathcal{S}$ is a set of functions satisfying (2) then we say that $\mathcal{S}$ is **closed under composition**. We write $g(f_1, \ldots, f_n)$ for the function $h$ defined in (2).[1]
- If $\mathcal{S}$ is a set of functions satisfying (3) then we say that $\mathcal{S}$ is **closed under primitive recursion**.
- For each $n, k \in \mathbb{N}$ we let $C_k^n$ denote the constant function $\lambda x_1 \cdots x_n.k$.

Some first steps:

[1] I think I've said this before.

**Lemma 2.2.2.** *The following functions:*

*(1)* $C^n_k$ *for all* $k, n \in \mathbb{N}$*.*

*(2)* $\lambda xy.x + y$

*(3)* $\lambda xy.x \cdot y$*.*

*(4)* $\lambda xy.x^y$*.*

*(5)* $\lambda xy.x \dot{-} y$ *(Recall:* $\dot{-}$ *is the bounded subtraction function from a while back).*

*are primitive recursive.*

PROOF. Let's do some of these for practice:

(1) Trivial.

(2) Let $h$ be the function defined by recursion as follows:

$$\begin{aligned} &h : \mathbb{N}^2 \to \mathbb{N} \\ &(x, y) \mapsto \begin{cases} P^2_1(x, y) & \text{if } y = 0 \\ P^3_3(x, y, S(h(x, y - 1))) & \text{if y>0.} \end{cases} \end{aligned}$$

We can prove by induction that this $h(x, y) = x + y$, and we will do so, but first, observe that the way I've written this function is very convoluted, just to fit it exactly with the definition. An easier way of writing this would be:

$$\begin{aligned} &h : \mathbb{N}^2 \to \mathbb{N} \\ &(x, y) \mapsto \begin{cases} x & \text{if } y = 0 \\ S(h(x, y - 1)) & \text{if y>0,} \end{cases} \end{aligned}$$

observing that since primitive recursive functions are closed under composition and contain projections, we can view any $f \in \mathcal{E} \cap \mathcal{F}_p$ as a function $f \in \mathcal{E} \cap \mathcal{F}_q$, for any $q \geq p$.

Now to prove that $h(x, y) = x + y$. We do so by induction on $y$. If $y = 0$ then $h(x, y) = h(x, 0) = x = x + 0$. For the inductive step, suppose that $y > 0$ and $h(x, y - 1) = x + (y - 1)$. Then:

$$\begin{aligned} h(x, y) &= S(h(x, y - 1)) = S(x + (y - 1)) = (x + (y - 1)) + 1 \\ &= x + y. \end{aligned}$$

(3) Now that we have shown that addition is primitive recursive, we can just use it. Let $h$ be the function defined by:

$$h : \mathbb{N}^2 \to \mathbb{N}$$
$$(x, y) \mapsto \begin{cases} 0 & \text{if } y = 0 \\ x + h(x, y-1) & \text{if y>0,} \end{cases}$$

The proof is the same inductive argument. The base case is trivial. For the inductive step, think about the following for a few seconds:

$$\begin{aligned} h(x, y) &= x + h(x, y-1) = x + x \times (y-1) = x \times ((y-1)) + 1) \\ &= x \times y. \end{aligned}$$

(4) Exercise.

(5) First, let $h_0$ be the function:

$$h_0 : \mathbb{N} \to \mathbb{N}$$
$$x \mapsto \begin{cases} x & \text{if } x = 0 \\ x - 1 & \text{if x>0.} \end{cases}$$

Of course, this is just $\lambda x.x \dot{-} 1$. Now, we can define our function proper:

$$h : \mathbb{N}^2 \to \mathbb{N}$$
$$(x, y) \mapsto \begin{cases} x & \text{if } y = 0 \\ h(x, y-1) \dot{-} 1 & \text{if y>0.} \end{cases}$$

□

At this point, you really should be thinking back to register machines and the "closure" properties we proved.

THEOREM 2.2.3. *Every primitive recursive function is register machine computable*

PROOF. We've actually already shown this in Proposition 1.1.4, Proposition 1.1.6, and Proposition 1.1.7. □

We naturally extend our definition to subsets of $\mathbb{N}^n$. Recall that we can associate every subset $Y$ of a set $X$ with its characteristic function $\mathbb{1}_Y$, which is the function:

$$\begin{aligned}\mathbb{1}_Y : X &\to \{0,1\}\\ x &\mapsto \begin{cases}1 & \text{if } x \in Y\\ 0 & \text{otherwise.}\end{cases}\end{aligned}$$

Some people call this the indicator function of $Y$.

**Definition 2.2.4.** Let $X \subseteq \mathbb{N}^n$. We say that $X$ is *primitive recursive* if its characteristic function $\mathbb{1}_X$ is primitive recursive.

**Example 2.2.5.** The set $\mathbb{N}_{>0}$ is primitive recursive. Indeed:

$$\mathbb{1}_{\mathbb{N}_{>0}} = 1 \dot{-} (1 \dot{-} x).[2]$$

The set $X = \{(x,y) \in \mathbb{N}^2 : x < y\}$ is primitive recursive. Indeed:

$$\mathbb{1}_X = \mathbb{1}_{\mathbb{N}_{>0}}(y \dot{-} x).$$

Let's prove some properties of primitive recursive functions and sets. The next lemma will be extremely useful later on both in that it will allow us to build tones of primitive recursive functions but also in that (if we read between the lines) it is starting to show us where the limitations of primitive recursion lie.[3]

**Lemma 2.2.6.**

*(1) The set of primitive recursive functions is closed under permutations of variables.*

*(2) If $X \subseteq \mathbb{N}^n$ is primitive recursive and $f_1, \ldots, f_n \in \mathcal{F}^p$ are primitive recursive, then so is the set $\{(x_1, \ldots, x_p) : (f_1(\bar{x}), \ldots, f_n(\bar{x})) \in X\}$.*

*(3) The set of primitive recursive subsets of $\mathbb{N}^n$ contains $\emptyset$, $\mathbb{N}^n$, and is closed under $\cup, \cap$ and relative complements.*

*(4) **Definition by cases**: Let $A_1, \ldots, A_k$ be a partition of $\mathbb{N}^p$ into primitive recursive sets and let $f_1, \ldots, f_k : \mathbb{N}^p \to \mathbb{N}$ be primitive recursive. Then, the*

[2]This is either clever or stupid, but I'll let you decide.
[3]The word bounded will appear multiple times.

*function:*

$$f : \mathbb{N}^p \to \mathbb{N}$$
$$(x_1, \dots x_p) \mapsto \begin{cases} f_1(x_1, \dots, x_p) & \text{if } (x_1, \dots x_p) \in A_1 \\ f_2(x_1, \dots, x_p) & \text{if } (x_1, \dots x_p) \in A_2 \\ & \vdots \\ f_k(x_1, \dots, x_p) & \text{if } (x_1, \dots x_p) \in A_k \end{cases}$$

*is primitive recursive.*

(5) ***Bounded sums and products**: If $f \in \mathcal{F}_{p+1}$ is primitive recursive, then so are the functions:*

$$\lambda x_1, \dots, x_n, y. \sum_{i=0}^{y} f(x_1, \dots, x_n, y) \text{ and } \lambda x_1, \dots, x_n, y. \prod_{i=0}^{y} f(x_1, \dots, x_n, y)$$

(6) ***Bounded $\mu$-operation**: Let $X \subseteq \mathbb{N}^{p+1}$ be primitive recursive. Then the function:*

$$f : \mathbb{N}^{p+1} \to \mathbb{N}$$
$$(\bar{x}, z) \mapsto \begin{cases} 0 & \text{if there is no } t \leq z \text{ with } (\bar{x}, t) \in X \\ t_0 & \text{if } t_0 \text{ is minimal in } \mathbb{N} \text{ s.t. } t_0 \leq z \text{ and } (\bar{x}, t_0) \in X. \end{cases}$$

*We write $f(\bar{x}, z) = \mu(t \leq z). ((\bar{x}, t) \in X)$.*[4]

(7) ***Bounded quantification**: If $X \subseteq \mathbb{N}^{p+1}$ is primitive recursive, then so are:*

$$X_e := \{(x_1, \dots, x_p, z) : \text{ if there is some } t \leq z \text{ s.t. } (\bar{x}, t) \in X\}$$

*and*

$$X_a := \{(x_1, \dots, x_p, z) : \text{ if for all } t \leq z \text{ we have } (\bar{x}, t) \in X\}.$$

PROOF.

(1) This is trivial, since we can compose in funky ways with projection functions.

(2) The indicator function of the set in question is but the function $\mathbb{1}_X(f_1, \dots, f_n)$

(3) It suffices to show complements and intersections $\mathbb{1}_{\mathbb{N}^n \setminus X} = 1 \dot{-} \mathbb{1}_X$ and $\mathbb{1}_{X \cap Y} = \mathbb{1}_X \times \mathbb{1}_X$.

[4] We write $\mu$ for "minimal". The expression here means that the function returns the smallest $t$ below $z$ which satisfies a primitive recursive condition. We are talking about a bounded operation, since we have an upper bound – we'll only try things up to $z$.

(4) We simply observe that:

$$f = \sum_{i=1}^{k} \mathbb{1}_{A_i} \times f_i$$

(5) For example:

$h : \mathbb{N}^p \to \mathbb{N}$

$$(x_1, \dots, x_p, y) \mapsto \begin{cases} f(x_1, \dots, x_p, 0) & \text{if } y = 0 \\ f(x_1, \dots, x_p, y) + h(x_1, \dots, x_p, y-1) & \text{otherwise.} \end{cases}$$

(6) This one is certainly clever: We set $f(\bar{x}, 0) = 0$, of course. Then for the recursive step:

$$f(\bar{x}, z+1) = \begin{cases} f(\bar{x}, z) & \text{if } \sum_{t=0}^{z} \mathbb{1}_X(\bar{x}, t) \geq 1 \\ z+1 & \text{if } \sum_{t=0}^{z} \mathbb{1}_X(\bar{x}, t) = 0 \text{ and } (\bar{x}, z+1) \in X \\ 0 & \text{otherwise.} \end{cases}$$

We have here used both bounded sums and definitions by cases.

(7) It is enough to show $X_e$ is primitive recursive (we can then take complements). To see this:

$$\mathbb{1}_{X_e}(\bar{x}, z) = \begin{cases} 1 & \text{if } \sum_{t=1}^{z} \mathbb{1}_X(\bar{x}, t) \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

□

This lemma lets us see that many many sets we know and love are primitive recursive. Some of the most lovable ones will be collected in the next corollary.

**Corollary 2.2.7.**

*(1) The set $\{(x, y) \in \mathbb{N}^2 : y|x\}$ is primitive recursive.*[5]

*(2) The set of $P \subseteq \mathbb{N}$ of prime numbers is primitive recursive.*

*(3) The function $\mathsf{pr} : \mathbb{N} \to \mathbb{N}$ which on input $n$ returns the $(n+1)$-st prime number is primitive recursive.*

*(4) There is a primitive recursive bijection $\mathsf{pair} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.*

[5]Here $y|x$ denotes the assertion "$x$ is divisible by $y$"

(5) *There are primitive recursive functions* $\mathsf{unpair}_1 : \mathbb{N} \to \mathbb{N}$ *and* $\mathsf{unpair}_2 : \mathbb{N} \to \mathbb{N}$ *such that:*

$$\mathsf{unpair}_i(\mathsf{pair}(x_1, x_2)) = x_i,$$

*for* $i \leq 2$.

(6) *More generally, for all* $n \in \mathbb{N}$ *there are primitive recursive bijections*

$$\mathsf{tuple}^n : \mathbb{N}^n \to \mathbb{N}$$

*and primitive recursive functions*

$$\mathsf{untuple}^n_i : \mathbb{N}^n \to \mathbb{N}$$

*for each* $i \leq n$ *such that:*

$$\mathsf{untuple}^n_i(\mathsf{tuple}^n(x_1, \ldots, x_n)) = x_i.$$

Proof.

(1) First, we see that the function $q(x, y)$ which given $(x, y)$ returns the floor of $\frac{x}{y}$ if $y > 0$ and 0 otherwise is primitive recursive. Indeed:

$$q(x, y) = (\mu t \leq x)((t + 1) \times y > x).$$

Given this, we have that the characteristic function of the set $\{(x, y) \in \mathbb{N}^2 : y|x\}$ is just:

$$1 \dot{-} (x \dot{-} q(x, y) \times y).$$

(2) We know that primes are the numbers greater than 1 that are only divisible by themselves and 1. Consider the following three primitive recursive sets:

$$X_1 = \{x \in \mathbb{N} : x > 1\}$$

$$X_2 = \{(x, y) \in \mathbb{N}^2 : y \leq 1\} \cup \{(x, y) \in \mathbb{N}^2 : x = y\} \cup \{(x, y) \in \mathbb{N}^2 : y \not| x\}$$

Then, the following set is also primitive recursive:

$$X_3 = \{(x, z) \in \mathbb{N}^2 : \text{ for all } y \leq z \text{ we have } z \in X_2\}$$

And thus, the set

$$X_4 = X_3 \cap \{(x, y) \in \mathbb{N}^2 : x = y\}$$

is primitive recursive. Finally, the set:

$$X_1 \cap P^2_1(X_4)$$

is primitive recursive, and is, indeed the set of all primes.

(3) After a little bit of thought, we see that:

$$\mathsf{pr}(n) = \begin{cases} 2 & \text{if } n = 0 \\ \mu(z \leq \mathsf{pr}(n-1)! + 1).(z > \mathsf{pr}(n-1) \text{ and } z \in P) & \text{otherwise,} \end{cases}$$

since there is always a prime strictly between $n$ and $n! + 2$ [WHY?]. Now to elaborate a bit about the shorthand used above, in case you're very pedantic like I proudly used to be when I was younger. For every $y \in \mathbb{N}$ the set:

$$X_{>y} := \{x \in \mathbb{N} : x > y\}$$

is primitive recursive. Let $g(x, y)$ be the following function:

$$g(x, y) = \mu(z \leq y)(z \in X_{>y} \cap P)$$

This is primitive recursive. What we took before was $\mathsf{pr}(n+1) = g(n, \mathsf{pr}(n))$.

(4) The map in question is:

$$\mathsf{pair}(x, y) = \frac{1}{2}(x + y)(x + y + 1) + y.$$

(5) The map $\mathsf{unpair}_1$ is given by

$$\mu z \leq x.(\text{ there is } t \leq x \text{ s.t. } \mathsf{pair}(z, t) = x)$$

and the map $\mathsf{unpair}_2$ is defined analogously. I have again used here a similar shorthand as the one I used in (3).

(6) Immediate from (4) and (5) by induction.

□

**Remark 2.2.8.** In our proof that the set of all primes $P \subseteq \mathbb{N}$ is primitive recursive I tried to be as formal as possible. For instance, arguing as in that proof, we can see that for instance the following set is always primitive recursive:

$$X = \{x \in \mathbb{N} : \text{ for all } z \leq x \text{ we have } z \in Y\}$$

for any primitive recursive set $Y$. More generally, since primitive recursive sets are closed under Boolean combinations, we can shortcut things a lot.

If you're worried that the details of parts (4)-(5) went a bit fast, fret not:

**Exercise 2.2.9.**

(1) Prove that $\mathsf{pair}$ is a primitive recursive bijection.

(2) Prove that $\mathsf{unpair}_i$ have the required properties (from the statement of the corollary).

(3) Construct for all $n \in \mathbb{N}$ the map $\mathsf{tuple}^n$ and the maps $\mathsf{untuple}^n_i$.

We now have all the tools to define something very crucial for our later exploration of incompleteness, our primitive recursive way of coding sequences of numbers into numbers. We'll do this now that all the ideas are fresh in our heads, and return to it when we need it:

**2.3. Let's put this here for later.** If $(x_0, \ldots, x_{n-1}) \in \mathbb{N}^n$, then we define the **Gödel number** of $(x_0, \ldots, x_{n-1})$, denoted $\langle x_0, \ldots, x_{n-1} \rangle$, as follows:

$$\langle x_0, \ldots, x_{n-1} \rangle := \mathsf{pr}(0)^{x_0} \times \cdots \times \mathsf{pr}(n-2)^{x_{n-2}} \times \mathsf{pr}(n-1)^{x_{n-1}}.$$

and, for the sake of completeness:

$$\langle\rangle = 1.$$

Let's summarise the main properties of this new beast:

**Lemma 2.3.1.** *Gödel numbering lets us define a map from the set of all finite sequences of natural numbers to $\mathbb{N} \setminus \{0\}$ It satisfies the following properties:*

*(1) The* ***binary component*** *function:*

$$\mathbb{N} \times \mathbb{N} \to \mathbb{N}$$
$$(x, i) \mapsto \begin{cases} x_i & \text{if } x = \langle x_0, \ldots, x_{n-1} \rangle \text{ and } i < n \\ 0 & \text{otherwise} \end{cases}$$

*is primitive recursive. We write $(x)_i$ for the binary component of $x$.*

*(2) The* ***length function*** *given by $\mathsf{lg}(\langle x_0, \ldots, x_{n-1} \rangle) = n$ is primitive recursive.*

*(3) For all $n \in \mathbb{N}$ the map $\langle\rangle \restriction_{\mathbb{N}^n} \to \mathbb{N}$ is primitive recursive.*

*(4) For all $x \in \mathbb{N}$, $\mathsf{lg}(x) \leq x$.*

*(5) For all $x \in \mathbb{N}_{>0}$, $(x)_i < x$, for all $i \in \mathbb{N}$.*

Proof. The bullets here are not that hard once we uncover what they mean. I'll let everyone think about them for a bit, before I spoil the fun. □

Anyway, enough Gödel stuff for now. Let's get back to register machines. It really should be starting to feel like these primitive recursive fellas are good at capturing what we can compute using a "computer program". Unfortunately, it turns out that there are functions which we can intuitively compute, but are not primitive recursive.

We'll take a glimpse at one now, to justify the "correct" notion of a computable function.

**2.4. The Ackerman function.** We'll here build a classical example of a function that is intuitively computable (i.e. we can sit down with pen and paper and compute it) but is not primitive recursive. It will turn out that this function is register machine computable, and once we expand our notion of primitive recursive just a bit, we'll get the "right" notion of computation.

We define a map $A : \mathbb{N}^2 \to \mathbb{N}$ as follows:

- $A(0, x) = 2^x$, for all $x \in \mathbb{N}$.
- $A(y, 0) = 1$, for all $y \in \mathbb{N}$.
- For all $x, y$ we set $A(y + 1, x + 1) = A(y, A(y_1, x))$.

For each $n \in \mathbb{N}$ set:

$$A_n = \lambda x.A(n, x).$$

Then, $A_0 = 2^x$ and, clearly for all $n \in \mathbb{N}_{>0}$ we have that

$$A_n(0) = 0 \text{ and } A_n(x + 1) = A_{n-1}(A_n(x)).$$

Two things:

- This shows that the function $A : \mathbb{N}^2 \to \mathbb{N}$ exists.
- Each $A_n$ is primitive recursive.

At this point we'd love to be able to say that $A$ is also primitive recursive, but of course, we all see the writing on the wall at this point.

**Lemma 2.4.1.** *For all $n, x \in \mathbb{N}$ we have $A_n(x) > x$.*

Proof. Easy exercise on induction. □

**Corollary 2.4.2.** *For all $n \in \mathbb{N}$, the function $A_n$ is strictly increasing.*

Proof. Well. This is obvious for $n = 0$. For $n > 0$ this is by the previous lemma and the fact that $A_n(x + 1) = A_{n-1}(A_n(x))$. □

Similarly, we can also deduce the following:

**Lemma 2.4.3.** *For all $n \geq 1$ and all $x \in \mathbb{N}$ we have that $A_n(x) \geq A_{n-1}(x)$.*

A little bit more notation. For $k \in \mathbb{N}$ set $A_n^k$ to be the function $A_n$ iterated $k$ times (i.e. composed with itself $k$ times). Then:

**Lemma 2.4.4.** *The functions $A_n^k$ are all strictly increasing. Moreover:*

$$A_n^k(x) < A_n^{k+1}(x), A_n^k(x) \geq x, A_n^k \circ A_n^h = A_n^{k+h},$$

*and if $m \leq n$ then $A_m^k \leq A_n^k$, pointwise.*

Why all of this you may ask... The functions $A_n^k$ provide a pretty fine way of cutting up the primitive recursive functions in terms of how fast they grow. We say that a function $f \in \mathcal{F}_1$ **dominates** a function $g \in \mathcal{F}_p$ if there is some $N \in \mathbb{N}$ such that for all $\bar{x} \in \mathbb{N}^p$ we have that:

$$g(\bar{x}) \leq f(\max\{x_1, \ldots, x_p, A\}).$$

Let us define

$$C_n = \{g \in \mathcal{F} : \text{ for some } k \in \mathbb{N} \text{ we have that } A_n^k \text{ dominates } g\}$$

You can take the following on faith:

$$\bigcup_{n \in \mathbb{N}} C_n = \mathcal{E},$$

where recall $\mathcal{E}$ is the set of all primitive recursive functions. It's actually not so hard to show:

- $\bigcup_{n \in \mathbb{N}} C_n$ clearly contains all basic functions.
- By one of the lemmas on $A_n^k$, it's easy to deduce that $\bigcup_{n \in \mathbb{N}} C_n$ is closed under composition.
- The crux is showing that $\bigcup_{n \in \mathbb{N}} C_n$ is closed under primitive recursion (and okay fine, that's actually pretty hard).

Suppose that $A \in \mathcal{E}$. Then $\lambda x.A(x, 2x) \in \mathcal{E} = \bigcup_{n \in \mathbb{N}} C_n$. Then there exist integers $n, k$ and $N$ such that for all $x > N$ we have that

$$A(x, 2x) \leq A_n^k(x)$$

Thus, for all $x > N$ we have:

$$A(x, 2x) \leq A_n^k(x) \leq A_{n+1}(x + k).$$

Moreover, if $x > \max\{N, k, n_1\}$ we have:

$$A_{n+1}(x + k) < A_{n+1}(2x) < A_x(2x) < A(x, 2x).$$

We have thus concluded that $A(x, 2x) < A(x, 2x)$ which is stupid. Thus, $A \notin \mathcal{E}$ and indeed, the function $\lambda x.A(x, x)$ dominates ALL primitive recursive functions.

End of digression

So (if you were brave enough to read through the previous section) now you know of a function that we can see how to compute (and given enough time we could really write a register machine program for) which is not primitive recursive. This justifies the word *primitive* in the name! Now we'll define the actual recursive functions. The definition will start of rather similar, but there is a small caveat: