RECURSIVE AND COMPUTABLE FUNCTIONS

0. Preliminary set-theoretic notions

Before we begin, let us introduce the following convenient definitions from set theory[1]: We let $0 = \varnothing$, and in general the natural number n is defined to be

$$n = \{0, \dots, n-1\}.$$

So in particular, $1 = \{0\} = \{\varnothing\}$, $2 = \{0,1\} = \{\varnothing, \{\varnothing\}\}$, $3 = \{0,1,2\}$, etc. We let

$$\omega = \{0, 1, 2, \dots, n, \dots\},$$

that is, ω is the set of non-negative integers. The set ω is often denoted by $\mathbb{N}$ or $\mathbb{N}_0$ in other math classes.

Ordered tuples of elements are denotes $(a_1, \dots, a_n)$ or $\langle a_1, \dots, a_n \rangle$. The unusual-looking angled parenthesis are used as an alternative simply to avoid a possible future confusion with the (rounded) parenthesis that are part of our formal language.

If $A_1, \dots, A_n$ are sets (n of them), then recall that the *Cartesian product* is defined as

$$A_1 \times \cdots \times A_n = \{\langle a_1, \dots, a_n \rangle : a_1 \in A_1, \dots, a_n \in A_n\},$$

that is, the Cartesian product $A_1 \times \cdots \times A_n$ consists of *all* n-tuples $\langle a_1, \dots, a_n \rangle$ where $a_i \in A_i$ for all $1 \leqslant i \leqslant n$. For a set A we let A^n denote the n-fold Cartesian product of A with itself. A^1 is of course just A, and so we naturally identify $a \in A$ with $\langle a \rangle$.

In this course, a *relation* is a subset R of some (finite) Cartesian product of sets. To be precise, an *n-ary* relation is a subset $R \subseteq A_1 \times \cdots \times A_n$ of some n-fold cartesian product. One often writes $R(x_1, \dots, x_n)$ rather than $\langle x_1, \dots, x_n \rangle \in R$; if this is the case then we say that $x_1, \dots, x_n$ are *related by R* or that R holds of $\langle x_1, \dots x_n \rangle$. If $R \subseteq A^n$ for some fixed set A, then we call R an *n-ary relation on A*.

Let A be a set. A *sequence of length $n \in \omega$* in A is a function $s : n \to A$. It is natural to identify A^n with the set $\{s : n \to A : s \text{ is a function}\}$. We let

$$A^{<\omega} = \bigcup_{n \in \omega} A^n;$$

that is, $A^{<\omega}$ is the set the set of all finite sequences in A. For $s \in A^{<\omega}$, we let $\ell\mathrm{h}(s) = \mathrm{dom}(s)$, the *length* of s.

For any function $f : B \to A$, where B and A are some sets, and a subset $B_0 \subseteq B$, we let $f \restriction B_0$ denote the *restriction* of f to B_0, that is, $f \restriction B_0$ is the function with domain B_0 defined by $(f \restriction B_0)(x) = f(x)$ for all $x \in B_0$.

[1]Until further notice, we will use sets naïvely in this course, that is, sets are treated just like they are in other math course you've taken.

1. Recursion theory

1.1. **Background.** What does it mean for a function $f : \omega \to \omega$ to be computable? This is a hard question to answer. Intuitively, it should mean something like this: There is a fixed procedure (given by a finite description of some sort), such that given a natural number n, one can carry out this procedure in a finite amount of time with input n, and determine $f(n)$.

For instance, most people feel that the function $f(x) = x^2 + x + 1$ is computable, because given n, we first compute n^2 by adding n to itself n times (which requires further subtasks, perhaps), and then we add n to it, and then we add 1, and this is the answer, $f(n)$. Of course, one must investigate why it is we believe that adding n to itself n times is a computable task, etc., which leads one to go back to how we were taught to add numbers in elementary school. At any rate, everyone believes this function f is computable.

Several attempts have been made to formalize the notion of a computable function. Two such are Church's *Lambda calculus*, and Turing's *Turing machines*. A third possibility is the notion of a recursive function, given below. It turns out that all these approaches define the same concept (though they look rather different), which is seen as evidence that they all correctly formalize the (same) intuitive concept of a computable function.

1.2. **Primitive recursive functions.** We now describe a class of certain functions of the form $f : \omega^n \to \omega$, where n is allowed to range over ω. These functions are supposed to be intuitively computable. The following gets us started.

Definition 1.1. The class of *elementary* functions consists of:

1. The successor function $S : \omega \to \omega$, defined by $S(x) = x + 1$.
2. The projection functions, defined for each $n \geqslant 1$ and $1 \leqslant i \leqslant n$ by $I_i^n(x_1, \dots, x_n) = x_i$.
3. All constant functions, i.e. $c : \omega^n \to \omega$ where $c(x_1, \dots, x_n) = k$ for some $k \in \omega$, independently of $x_1, \dots, x_n$.

We make the convention that any function $f : \omega^0 \to \omega$ is to be thought of as a constant function (taking some fixed value in ω). Alternatively, a function $f : \omega^0 \to \omega$ may be thought of as an element of ω.

The class of primitive recursive functions will be built from the elementary functions using two function construction "schemes".

Composition scheme. Given $h : \omega^m \to \omega$ and $g_1, \dots, g_m : \omega^k \to \omega$, we may define a new function $f : \omega^k \to \omega$ by

$$f(x_1, \dots, x_k) = h(g_1(x_1, \dots, x_k), g_2(x_1, \dots, x_k), \dots, g_m(x_1, \dots, x_k)).$$

Primitive recursion scheme. Given $k \geqslant 1$, and functions $h : \omega^{k-1} \to \omega$, $g : \omega^{k+1} \to \omega$, we can define a new function $f : \omega^k \to \omega$ by letting

$$f(0, x_2, \dots, x_k) = h(x_2, \dots, x_k)$$

and

$$f(x_1 + 1, x_2, \dots, x_k) = g(x_1, f(x_1, \dots, x_k), x_2, \dots, x_k)$$

Note that here the convention that 0-ary functions are constant functions is handy, since it allows us to treat the case $k = 1$ elegantly and uniformly with the case $k > 1$.

Definition 1.2. The class of *primitive recursive functions* is the smallest class of functions which contains the elementary functions and which is closed under the composition scheme and the primitive recursion scheme.

A more descriptive way of defining the class of primitive recursive functions is the following: A function f is primitive recursive if only if there is a finite sequence $f_1, \dots, f_n$ of functions with $f_n = f$, and where each f_i is either elementary, or obtained by applying one of the schemes to functions that comes *before* f_i on the list.

Exercise 1. Check these two formulations of the definition are in fact equivalent!

Example 1.3. Addition is primitive recursive. More precisely, what this means is that the function $f : \omega \times \omega \to \omega$ defined by $f(m, n) = m+n$ is recursive. To see this, first use the composition scheme on I^3_2 and S to see that the function $g(m, u, n) = u + 1$ is primitive recursive. This follows because $g = S(I^3_2(m, u, n))$. Next use the recursion scheme on the function $h = I^1_1$ and g to obtain that the function defined by $f(0, n) = h(n)$ and $f(m + 1, n) = g(m, f(m, n), n)$ is primitive recursive. Now an easy induction on m, using that $f(m + 1, n) = f(m, n) + 1$, shows that for each fixed n we have $f(m, n) = m + n$. This proves that addition is primitive recursive.

Exercise 2. Prove that multiplication is primitive recursive, i.e., that the function $f(m, n) = m \cdot n$ is primitive recursive.

Example 1.4. The factorial function $f(n) = n!$ is primitive recursive. Informally, this can be seen from the recursive definition of the factorial, which is how it is usually introduced: $f(0) = 1$, and $f(n + 1) = (n + 1) \cdot f(n)$. Formally, we must "build" this function from elementary functions (or other functions already known to be primitive recursive) using only the two previous schemes. Use the primitive recursion scheme as follows: Let $h : \omega^0 \to \omega$ be the function which is constant 1, and let $g : \omega^2 \to \omega$ be the function $g(m, n) = S(m) \cdot n$. By the previous exercise (and the composition scheme), g is primitive recursive. By the primitive recursion scheme the function $f : \omega \to \omega$ defined by $f(0) = h = 1$ and $f(n + 1) = g(n, f(n)) = S(n) \cdot f(n)$ is primitive recursive. It is clear that $f(n) = n!$. (Formally, you could prove that this is so by induction.)

Example 1.5. There is no primitive recursive function $f : \omega^2 \to \omega$ which is universal in the sense that for every primitive recursive $g : \omega \to \omega$, there is $m \in \omega$ such that $g(n) = f(m, n)$ for all n. Indeed, if f was such a function then define

$$g_\Delta(n) = S(f(n, n)).$$

Then the composition scheme would ensure that g_Δ would be primitive recursive, and so there would be some $m \in \omega$ such that $f(m, n) = g_\Delta(n)$ for all n. But this means that

$$f(m, m) = g_\Delta(m) = S(f(m, m)) = f(m, m) + 1,$$

a contradiction.

Example 1.6. The function $\mathrm{sg} : \omega \to \omega$ defined by

$$\mathrm{sg}(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ otherwise} \end{cases}$$

is primitive recursive. Indeed, let $\mathrm{sg}(0) = 0$, and in general $\mathrm{sg}(n+1) = 1$. More explicitly, we let $h\colon \omega^0 \to \omega$ take the value 0 and we let $g\colon \omega^2 \to \omega$ be the constant function with value 1. Both are elementary as they are constant functions, so we may apply the recursion scheme to them. The equations of the recursion scheme now become $f(0) = 0$ and $f(n+1) = 1$ for all $n \in \omega$, so $f = \mathrm{sg}$ is a recursive function.

Exercise 3. Show that the function

$$\mathrm{pred}(n) = \begin{cases} n-1 \text{ if } n > 0 \\ 0 \text{ otherwise} \end{cases}$$

is primitive recursive.

Exercise 4. Show that every eventually constant function is primitive recursive (a function $f\colon \omega \to \omega$ is eventually constant if there are n and k such that for all $\bar{n} \geqslant n$, we have $f(\bar{n}) = k$).

Exercise 5. Show that the function “monus”, defined by

$$n \dot{-} m = \begin{cases} n-m \text{ if } n > m \\ 0 \text{ otherwise,} \end{cases}$$

is primitive recursive.

Proposition 1.7. *(1) If $f : \omega^n \to \omega$ is primitive recursive and $m \geqslant n$, then the function*

$$\tilde{f}(x_1, \dots, x_m) = f(x_1, \dots, x_n)$$

is primitive recursive.

(2) If $f : \omega^n \to \omega$ is primitive recursive and $\sigma : n \to n$ is a permutation (i.e., bijection of n onto itself), then

$$f_\sigma(x_1, \dots, x_n) = (x_{\sigma(0)}, \dots, x_{\sigma(n-1)})$$

is primitive recursive.

(3) If $f : \omega^{n+1} \to \omega$ is primitive recursive, then so are the functions

$$f_\Sigma(x_1, \dots, x_n, z) = \sum_{y<z} f(x_1, \dots, x_n, y)$$

(where we make $f_\Sigma(x_1, \dots, x_n, 0) = 0$ by convention) and

$$f_\Pi(x_1, \dots, x_n, z) = \prod_{y<z} f(x_1, \dots, x_n, y)$$

(where $f_\Pi(x_1, \dots, x_n, 0) = 1$ by convention).

Proof. (1) By the composition scheme,

$$\tilde{f}(x_1, \dots, x_m) = f(I^m_1(x_1, \dots, x_m), \dots, I^m_n(x_1, \dots, x_m))$$

is primitive recursive when f is.

(2) Exercise.

(3) For simplicity, let's do it for $n = 1$. Let $a(x, z) = x + z$, which we know is primitive recursive. Let $g(z, u, x) = a(f(x, z), u)$, which is primitive recursive (why?). Then let $f_\Sigma(x, 0) = 1$, and

$$f_\Sigma(x, z+1) = g(z, f_\Sigma(x, z), x),$$

so that f_Σ is recursive by the primitive recursion scheme.

Showing that f_Π is primitive recursive is left as an exercise. ☐

Definition 1.8. A set (relation) $R \subseteq \omega^n$ is primitive recursive just in case the characteristic function

$$\mathbf{1}_R(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } \langle x_1, \dots, x_n \rangle \in R \\ 0 & \text{otherwise.} \end{cases}$$

is primitive recursive.

Example 1.9. The relation

$$<= \{\langle m, n \rangle \in \omega^2 : m < n\}$$

is primitive recursive. Namely, $\mathbf{1}_<(m, n) = \operatorname{sg}(n \dot{-} m)$.

Recall that the notation $R(x_1, \dots, x_n)$ means exactly the same as $\langle x_1, \dots, x_n \rangle \in R$.

Proposition 1.10. *Suppose $A, B, R \subseteq \omega^n$, $n > 0$, are primitive recursive sets (relations). Then:*

(1) $A \cup B$ and $A \cap B$ are primitive recursive. So is R^c, where $R^c = \omega^n \backslash R$, i.e. the relation defined by

$$\langle x_1, \dots x_n \rangle \in R^c \iff \langle x_1, \dots x_n \rangle \notin R.$$

(2) For any $m \geqslant n$, the relation $\tilde{R} \in \omega^m$ defined by

$$\{\langle x_1, \dots, x_m \rangle \in \tilde{R} : (x_1, \dots, x_n) \in R\}$$

is primitive recursive.

(3) The sets

$$R^{\exists <} = \{\langle x_1, \dots, x_n \rangle \in \omega^n : (\exists z < x_n) R(x_1, \dots, z)\}$$

$$R^{\forall <} = \{\langle x_1, \dots, x_n \rangle \in \omega^n : (\forall z < x_n) R(x_1, \dots, z)\}$$

are primitive recursive.

Note: (1) says that the primitive recursive relations are closed under basic set theoretic operations; (3) says that the primitive recursive relations are closed under "bounded quantification".

Proof. (1) $\mathbf{1}_{A \cap B}(x_1, \dots, x_n) = \mathbf{1}_A(x_1, \dots, x_n) \cdot \mathbf{1}_B(x_1, \dots, x_n)$, and

$$\mathbf{1}_{A \cup B} = \operatorname{sg}(\mathbf{1}_A(x_1, \dots, x_n) + \mathbf{1}_B(x_1, \dots, x_n)),$$

$$\mathbf{1}_{R^c}(x_1, \dots, x_n) = 1 \dot{-} \mathbf{1}_R(x_1, \dots, x_n).$$

(2) Exercise.

(3) We have

$$\mathbf{1}_{R^{\exists <}}(x_1, \dots, x_n) = \operatorname{sg}(\sum_{z < x_n} \mathbf{1}_R(x_1, \dots, x_{n-1}, z)),$$

and checking that $R^{\forall <}$ is primitive recursive is left as an exercise. ☐

1.3. **Recursive functions.** While the class of primitive recursive functions is indeed very large, and, as it will turn out, essentially sufficient for our purposes, it isn't quite big enough to capture all reasonably computable functions. We obtain a larger class of functions, called the *recursive* functions, by adding one more scheme to the mix.

μ-OPERATOR SCHEME. If $g : \omega^{k+1} \to \omega$ is a function for which it holds that

$$(\forall x_1, \dots, x_n \in \omega)(\exists y \in \omega) g(y, x_1, \dots, x_n) = 0$$

then we may form a new function $f : \omega^n \to \omega$ by

$$f(x_1 \dots, x_n) = \mu y[g(y, x_1, \dots, x_n) = 0]$$

where $\mu y[...]$ means "the least y such that [...]".

Definition 1.11. The class of recursive functions is the smallest class of functions which contains the elementary functions, and which is closed under the composition scheme, the primitive recursion scheme, and the μ-operator scheme.

Some people would refer to this class as the *total recursive functions* (to distinguish them from yet another class, the so-called *partial recursive functions*, which we will not define). In this course, we shall just call them the recursive functions.

Note that all primitive recursive functions are recursive. An alternative description of the class of recursive functions is: A function f is recursive iff there is a finite list $f_1, \dots, f_n$, where $f = f_n$, and where each element on the list is either an elementary function, or obtained by applying one of the three schemes to functions appearing earlier on the list.

There are recursive functions that are not primitive recursive, but that is a story for another day.

Definition 1.12. (1) A relation $R \subseteq \omega^n$ is recursive iff its characteristic function

$$\mathbf{1}_R(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } \langle x_1, \dots, x_n \rangle \in R \\ 0 & \text{otherwise} \end{cases}$$

is recursive.

(2) A set $A \subseteq \omega$ is *recursively enumerable* if there is a recursive function $f : \omega \to \omega$ with $\operatorname{ran}(f) = A$.

Computable vs. recursive functions. All recursive functions are computable (in the intuitive sense). This can be seen by induction: All elementary functions are clearly computable; if a function is formed by one of the schemes from computable functions, then it is also computable (if you don't see it, go back and re-read the schemes and convince yourself.)

In 1936, American logician Alonzo Church (1903–1995) introduced a class of computable functions in a different way, using what he called λ-calculus. It turns out that the class of recursive functions and the class of functions defined using λ-calculus are the same. They are also the same as the class of Turing computable functions, introduced by Alan Turing (1912–1954) in 1936 using an idealized notion of a computer. Church hypothesized the following:

Church's thesis. *The class of intuitively computable function on the natural numbers corresponds exactly to the class of recursive functions.*

This claim cannot be proved, since the notion of intuitively computable function is not a mathematical notion, but rather an intuitive idea. Church's thesis could potentially be *disproved* by giving and example of an intuitively computable function which is not recursive, but no-one has ever succeeded in doing so. Rather, the vast majority of mathematicians familiar with mathematical logic believe Church's thesis to be true.

2. Additional exercises

Exercise 6. Show that the function $E(k, n) = k^n$ is primitive recursive. (Here $k^0 = 1$ always.)

Exercise 7. Let $k \in \omega$. Prove that the functions $\max_k : \omega^k \to \omega$ and $\min_k : \omega^k \to \omega$ defined by

$$\max_k(x_1, \dots, x_k) = \text{ the largest of } x_1, \dots, x_k$$

and

$$\min_k(x_1, \dots, x_k) = \text{ the smallest of } x_1, \dots, x_k$$

are primitive recursive.

Hint: Use induction on k. The case $k = 1$ is trivial (why?) and irrelevant, so concentrate on the case $k = 2$.

Exercise 8. Show that if $P \subseteq \omega^k$ is a primitive recursive set[2] then so is the complement, $\omega^k \backslash P$.

Exercise 9. Prove the "definition by cases" theorem:

Theorem. *Let $P_1, \dots, P_m \subseteq \omega^k$ be primitive recursive subsets of ω^k for some (fixed) k, and let $f_1, \dots, f_{m+1} : \omega^k \to \omega$ be primitive recursive. Suppose $P_1, \dots, P_k$ are disjoint. Then*

$$f(x) = \begin{cases} f_1(x) & \textit{if } x \in P_1 \\ \vdots & \vdots \\ f_m(x) & \textit{if } x \in P_m \\ f_{m+1}(x) & \textit{otherwise.} \end{cases}$$

is primitive recursive.

Hint: Multiplying f_i with $\mathbf{1}_{P_i}$ looks promising.

Exercise 10. Prove that if $f : \omega^k \to \omega$ is primitive recursive, then the *graph* of f, defined by

$$\text{graph}(f) = \{(x_1, \dots, x_k, y) \in \omega^{k+1} : f(x_1, \dots, x_k) = y\},$$

is primitive recursive as a subset of ω^{k+1}.

Exercise 11. Show that the relation $\text{Div} \subseteq \omega^2$ defined by

$$\text{Div}(m, n) \iff m \text{ divides } n$$

is primitive recursive.

[2]Recall that this just means that the characteristic functions $\mathbf{1}_P$ is primitive recursive.

Hint: Consider the relation

$$R = \{(m, n, k) \in \omega^3 : m \cdot k = n\}.$$

Argue that this relation is recursive (see exercise 5), and use bounded existential quantification on the last coordinate. From there you can easily express $\mathbf{1}_{\mathrm{Div}}$ a composition of functions.

Exercise 12. Show that the set

$$\text{Prime} = \{n \in \omega : n \text{ is a prime number}\}$$

is primitive recursive.

The following problem is a bit harder, and should be viewed as a challenge.

Bonus problem. Let p_i be the $(i + 1)$'th prime, i.e., $p_0 = 2$, $p_1 = 3$, $p_2 = 5$, etc. Show that the function $f(i) = p_i$ is primitive recursive.