5.7 The Primitive Recursive Functions

The class of primitive recursive functions is defined in terms of base functions and closure operations.

Definition 5.12. Let $\Sigma = \{a_1, \ldots, a_N\}$. The *base functions* over Σ are the following functions:

(1) The *erase function* E, defined such that $E(w) = \epsilon$, for all $w \in \Sigma^*$;

(2) For every j, $1 \leq j \leq N$, the *j-successor function* S_j, defined such that $S_j(w) = wa_j$, for all $w \in \Sigma^*$;

(3) The *projection functions* P_i^n, defined such that

$$P_i^n(w_1, \ldots, w_n) = w_i,$$

for every $n \geq 1$, every i, $1 \leq i \leq n$, and for all $w_1, \ldots, w_n \in \Sigma^*$.

Note that P_1^1 is the identity function on Σ^*. Projection functions can be used to permute the arguments of another function.

A crucial closure operation is (extended) composition.

Definition 5.13. Let $\Sigma = \{a_1, \ldots, a_N\}$. For any function

$$g\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m} \to \Sigma^*,$$

and any m functions

$$h_i\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{n} \to \Sigma^*,$$

the *composition of g and the h_i* is the function

$$f\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{n} \to \Sigma^*,$$

denoted as $g \circ (h_1, \ldots, h_m)$, such that

$$f(w_1, \ldots, w_n) = g(h_1(w_1, \ldots, w_n), \ldots, h_m(w_1, \ldots, w_n)),$$

for all $w_1, \ldots, w_n \in \Sigma^*$.

As an example, $f = g \circ (P_2^2, P_1^2)$ is such that

$$f(w_1, w_2) = g(w_2, w_1).$$

Another crucial closure operation is *primitive recursion*.

Definition 5.14. Let $\Sigma = \{a_1, \ldots, a_N\}$. For any function

$$g\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m-1} \to \Sigma^*,$$

where $m \geq 2$, and any N functions

$$h_i\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m+1} \to \Sigma^*,$$

the function

$$f\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m} \to \Sigma^*,$$

is defined by *primitive recursion from g and $h_1, \ldots, h_N$*, if

$$\begin{aligned}
f(\epsilon, w_2, \ldots, w_m) &= g(w_2, \ldots, w_m),\\
f(ua_1, w_2, \ldots, w_m) &= h_1(u, f(u, w_2, \ldots, w_m), w_2, \ldots, w_m),\\
&\cdots = \cdots\\
f(ua_N, w_2, \ldots, w_m) &= h_N(u, f(u, w_2, \ldots, w_m), w_2, \ldots, w_m),
\end{aligned}$$

for all $u, w_2, \ldots, w_m \in \Sigma^*$.

When $m = 1$, for some fixed $w \in \Sigma^*$, we have

$$\begin{aligned} f(\epsilon) &= w, \\ f(ua_1) &= h_1(u, f(u)), \\ \cdots &= \cdots \\ f(ua_N) &= h_N(u, f(u)), \end{aligned}$$

for all $u \in \Sigma^*$.

For numerical functions (i.e., when $\Sigma = \{a_1\}$), the scheme of primitive recursion is simpler:

$$\begin{aligned} f(0, x_2, \ldots, x_m) &= g(x_2, \ldots, x_m), \\ f(x+1, x_2, \ldots, x_m) &= h_1(x, f(x, x_2, \ldots, x_m), x_2, \ldots, x_m), \end{aligned}$$

for all $x, x_2, \ldots, x_m \in \mathbb{N}$.

The *successor function* S is the function

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

Addition, multiplication, exponentiation, and super-exponentiation can, be defined by primitive recursion as follows (being a bit loose, we should use some projections ...):

$$\begin{aligned}
add(0, n) &= n,\\
add(m+1, n) &= S(add(m, n)),\\
mult(0, n) &= 0,\\
mult(m+1, n) &= add(mult(m, n), n),\\
rexp(0, m) &= 1,\\
rexp(m+1, n) &= mult(rexp(m, n), n),\\
exp(m, n) &= rexp \circ (P_2^2, P_1^2),\\
supexp(0, n) &= 1,\\
supexp(m+1, n) &= exp(n, supexp(m, n)).
\end{aligned}$$

As an example over $\{a, b\}^*$, the following function $g\colon \Sigma^* \times \Sigma^* \to \Sigma^*$, is defined by primitive recursion:

$$\begin{aligned} g(\epsilon, v) &= P_1^1(v), \\ g(ua_i, v) &= S_i \circ P_2^3(u, g(u, v), v), \end{aligned}$$

where $1 \leq i \leq N$. It is easily verified that $g(u, v) = vu$. Then,

$$f = g \circ (P_2^2, P_1^2)$$

computes the concatenation function, i.e. $f(u, v) = uv$.

Definition 5.15. Let $\Sigma = \{a_1, \ldots, a_N\}$. The class of *primitive recursive functions* is the smallest class of functions (over Σ^*) which contains the base functions and is closed under composition and primitive recursion.

We leave as an exercise to show that every primitive recursive function is a total function. The class of primitive recursive functions may not seem very big, but it contains all the total functions that we would ever want to compute.

Although it is rather tedious to prove, the following theorem can be shown.

Theorem 5.4. *For an alphabet $\Sigma = \{a_1, \ldots, a_N\}$, every primitive recursive function is Turing computable.*

The best way to prove the above theorem is to use the computation model of RAM programs. Indeed, it was shown in Theorem 5.2 that every RAM program can be converted to a Turing machine.

It is also rather easy to show that the primitive recursive functions are RAM-computable.

In order to define new functions it is also useful to use predicates.

Definition 5.16. An *n-ary predicate* P (over Σ^*) is any subset of $(\Sigma^*)^n$. We write that a tuple $(x_1, \ldots, x_n)$ *satisfies* P as $(x_1, \ldots, x_n) \in P$ or as $P(x_1, \ldots, x_n)$. The *characteristic function* of a predicate P is the function $C_P \colon (\Sigma^*)^n \to \{a_1\}^*$ defined by

$$C_p(x_1, \ldots, x_n) = \begin{cases} a_1 & \text{iff } P(x_1, \ldots, x_n) \\ \epsilon & \text{iff } \textbf{not } P(x_1, \ldots, x_n). \end{cases}$$

A predicate P is *primitive recursive* iff its characteristic function C_P is primitive recursive.

We leave to the reader the obvious adaptation of the the notion of primitive recursive predicate to functions defined over $\mathbb{N}$. In this case, 0 plays the role of ϵ and 1 plays the role of a_1.

It is easily shown that if P and Q are primitive recursive predicates (over $(\Sigma^*)^n$), then $P \vee Q$, $P \wedge Q$ and $\neg P$ are also primitive recursive.

As an exercise, the reader may want to prove that the predicate (defined over $\mathbb{N}$):
prime(n) iff n is a prime number, is a primitive recursive predicate.

For any fixed $k \geq 1$, the function:
$\mathrm{ord}(k, n)$ = exponent of the kth prime in the prime factorization of n, is a primitive recursive function.

We can also define functions by cases.

Lemma 5.5. *If $P_1, \ldots, P_n$ are pairwise disjoint primitive recursive predicates (which means that $P_i \cap P_j = \emptyset$ for all $i \neq j$) and $f_1, \ldots, f_{n+1}$ are primitive recursive functions, the function g defined below is also primitive recursive:*

$$g(\overline{x}) = \begin{cases} f_1(\overline{x}) & \text{iff } P_1(\overline{x}) \\ \vdots & \\ f_n(\overline{x}) & \text{iff } P_n(\overline{x}) \\ f_{n+1}(\overline{x}) & \text{otherwise.} \end{cases}$$

(writing $\overline{x}$ for $(x_1, \ldots, x_n)$.)

It is also useful to have bounded quantification and bounded minimization.

Definition 5.17. If P is an $(n+1)$-ary predicate, then the *bounded existential predicate* $\exists y/x P(y, \overline{z})$ holds iff some prefix y of x makes $P(y, \overline{z})$ true.

The *bounded universal predicate* $\forall y/x P(y, \overline{z})$ holds iff every prefix y of x makes $P(y, \overline{z})$ true.

Lemma 5.6. *If P is an $(n+1)$-ary primitive recursive predicate, then $\exists y/x P(y, \overline{z})$ and $\forall y/x P(y, \overline{z})$ are also primitive recursive predicates.*

As an application, we can show that the equality predicate, $u = v?$, is primitive recursive.

Definition 5.18. If P is an $(n+1)$-ary predicate, then the *bounded minimization of P*, $\min y/x\, P(y, \overline{z})$, is the function defined such that $\min y/x\, P(y, \overline{z})$ is the shortest prefix of x such that $P(y, \overline{z})$ if such a y exists, xa_1 otherwise.

The *bounded maximization of P*, $\max y/x\, P(y, \overline{z})$, is the function defined such that $\max y/x\, P(y, \overline{z})$ is the longest prefix of x such that $P(y, \overline{z})$ if such a y exists, xa_1 otherwise.

Lemma 5.7. *If P is an $(n+1)$-ary primitive recursive predicate, then $\min y/x\, P(y, \overline{z})$ and $\max y/x\, P(y, \overline{z})$ are primitive recursive functions.*

So far, the primitive recursive functions do not yield all the Turing-computable functions. In order to get a larger class of functions, we need the closure operation known as minimization.

5.8 The Partial Recursive Functions

Minimization can be viewed as an abstract version of a while loop.

Let $\Sigma = \{a_1, \ldots, a_N\}$. For any function

$$g\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m+1} \to \Sigma^*,$$

where $m \geq 0$, for every j, $1 \leq j \leq N$, the function

$$f\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m} \to \Sigma^*$$

looks for the shortest string u over a_j^* (for a given j) such that

$$g(u, w_1, \ldots, w_m) = \epsilon :$$

$u := \epsilon$;
while $g(u, w_1, \ldots, w_m) \neq \epsilon$ **do**
$u := ua_j$;
endwhile
let $f(w_1, \ldots, w_m) = u$

The operation of minimization (sometimes called minimalization) is defined as follows.

Definition 5.19. Let $\Sigma = \{a_1, \ldots, a_N\}$. For any function

$$g\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m+1} \to \Sigma^*,$$

where $m \geq 0$, for every j, $1 \leq j \leq N$, the function

$$f\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{m} \to \Sigma^*,$$

is defined by *minimization over $\{a_j\}^*$ from g*, if the following conditions hold for all $w_1, \ldots, w_m \in \Sigma^*$:

(1) $f(w_1, \ldots, w_m)$ is defined iff there is some $n \geq 0$ such that
$g(a_j^p, w_1, \ldots, w_m)$ is defined for all p, $0 \leq p \leq n$, and

$$g(a_j^n, w_1, \ldots, w_m) = \epsilon.$$

(2) When $f(w_1, \ldots, w_m)$ is defined,

$$f(w_1, \ldots, w_m) = a_j^n,$$

where n is such that

$$g(a_j^n, w_1, \ldots, w_m) = \epsilon$$

and

$$g(a_j^p, w_1, \ldots, w_m) \neq \epsilon$$

for every p, $0 \leq p \leq n - 1$.

We also write

$$f(w_1, \ldots, w_m) = min_j u[g(u, w_1, \ldots, w_m) = \epsilon].$$

Note: When $f(w_1, \ldots, w_m)$ is defined,

$$f(w_1, \ldots, w_m) = a_j^n,$$

where n is the smallest integer such that condition (1) holds. It is very important to require that all the values $g(a_j^p, w_1, \ldots, w_m)$ be defined for all p, $0 \leq p \leq n$, when defining $f(w_1, \ldots, w_m)$. Failure to do so allows non-computable functions.

Remark: Kleene used the *μ-notation*:

$$f(w_1, \ldots, w_m) = \mu_j u[g(u, w_1, \ldots, w_m) = \epsilon],$$

actually, its numerical form:

$$f(x_1, \ldots, x_m) = \mu x[g(x, x_1, \ldots, x_m) = 0],$$

The class of partial computable functions is defined as follows.

Definition 5.20. Let $\Sigma = \{a_1, \ldots, a_N\}$. The class of *partial recursive functions* is the smallest class of partial functions (over Σ^*) which contains the base functions and is closed under composition, primitive recursion, and minimization. The class of *recursive functions* is the subset of the class of partial recursive functions consisting of functions defined for every input (i.e., total functions).

One of the major results of computability theory is the following theorem.

Theorem 5.8. *For an alphabet $\Sigma = \{a_1, \ldots, a_N\}$, every partial recursive function is Turing-computable. Conversely, every Turing-computable function is a partial recursive function. Similarly, the class of recursive functions is equal to the class of Turing-computable functions that halt in a proper ID for every input.*

To prove that every partial recursive function is indeed Turing-computable, since by Theorem 5.2, every RAM program can be converted to a Turing machine, the simplest thing to do is to show that every partial recursive function is RAM-computable.

For the converse, one can show that given a Turing machine, there is a primitive recursive function describing how to go from one ID to the next. Then, minimization is used to guess whether a computation halts. The proof shows that every partial recursive function needs minimization at most once. The characterization of the recursive functions in terms of TM's follows easily.

There are recursive functions that are not primitive recursive. Such an example is given by Ackermann's function.

Ackermann's function: A recursive function which is **not** primitive recursive:

$$\begin{aligned} A(0,\, y) &= y + 1, \\ A(x+1,\, 0) &= A(x,\, 1), \\ A(x+1,\, y+1) &= A(x,\, A(x+1, y)). \end{aligned}$$

It can be shown that:

$$\begin{aligned} A(0,\, x) &= x + 1, \\ A(1,\, x) &= x + 2, \\ A(2,\, x) &= 2x + 3, \\ A(3,\, x) &= 2^{x+3} - 3, \end{aligned}$$

and

$$A(4, x) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{16}}}}}\Big\}x} - 3,$$

with $A(4, 0) = 16 - 3 = 13$.

For example

$$A(4,1) = 2^{16} - 3, \quad A(4,2) = 2^{2^{16}} - 3.$$

Actually, it is not so obvious that A is a total function. This can be shown by induction, using the lexicographic ordering $\preceq$ on $\mathbf{N} \times \mathbf{N}$, which is defined as follows:

$$\begin{aligned}
&(m,n) \preceq (m',n') \quad \text{iff either} \\
&m = m' \text{ and } n = n', \text{ or} \\
&m < m', \text{ or} \\
&m = m' \text{ and } n < n'.
\end{aligned}$$

We write $(m,n) \prec (m',n')$ when $(m,n) \preceq (m',n')$ and $(m,n) \neq (m',n')$.

We prove that $A(m,n)$ is defined for all $(m,n) \in \mathbf{N} \times \mathbf{N}$ by complete induction over the lexicographic ordering on $\mathbf{N} \times \mathbf{N}$.

In the base case, $(m, n) = (0, 0)$, and since $A(0, n) = n + 1$, we have $A(0, 0) = 1$, and $A(0, 0)$ is defined.

For $(m, n) \neq (0, 0)$, the induction hypothesis is that $A(m', n')$ is defined for all $(m', n') \prec (m, n)$. We need to conclude that $A(m, n)$ is defined.

If $m = 0$, since $A(0, n) = n + 1$, $A(0, n)$ is defined.

If $m \neq 0$ and $n = 0$, since

$$(m - 1, 1) \prec (m, 0),$$

by the induction hypothesis, $A(m - 1, 1)$ is defined, but $A(m, 0) = A(m - 1, 1)$, and thus $A(m, 0)$ is defined.

If $m \neq 0$ and $n \neq 0$, since

$$(m, n-1) \prec (m, n),$$

by the induction hypothesis, $A(m, n-1)$ is defined. Since

$$(m-1, A(m, n-1)) \prec (m, n),$$

by the induction hypothesis, $A(m-1, A(m, n-1))$ is defined. But $A(m, n) = A(m-1, A(m, n-1))$, and thus $A(m, n)$ is defined.

Thus, $A(m, n)$ is defined for all $(m, n) \in \mathbf{N} \times \mathbf{N}$. It is possible to show that A is a recursive function, although the quickest way to prove it requires some fancy machinery (the recursion theorem).

Proving that A is not primitive recursive is harder.

The following lemma shows that restricting ourselves to total functions is too limiting.

Let $\mathcal{F}$ be any set of total functions that contains the base functions and is closed under composition and primitive recursion (and thus, $\mathcal{F}$ contains all the primitive recursive functions).

We say that a function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is *universal* for the one-argument functions in $\mathcal{F}$ iff for every function $g : \Sigma^* \to \Sigma^*$ in $\mathcal{F}$, there is some $n \in \mathbb{N}$ such that

$$f(a_1^n, u) = g(u)$$

for all $u \in \Sigma^*$.

Lemma 5.9. *For any countable set $\mathcal{F}$ of total functions containing the base functions and closed under composition and primitive recursion, if f is a universal function for the functions $g : \Sigma^* \to \Sigma^*$ in $\mathcal{F}$, then $f \notin \mathcal{F}$.*

Proof. Assume that the universal function f is in $\mathcal{F}$. Let g be the function such that

$$g(u) = f(a_1^{|u|}, u)a_1$$

for all $u \in \Sigma^*$. We claim that $g \in \mathcal{F}$. It it enough to prove that the function h such that

$$h(u) = a_1^{|u|}$$

is primitive recursive, which is easily shown.

Then, because f is universal, there is some m such that

$$g(u) = f(a_1^m, u)$$

for all $u \in \Sigma^*$. Letting $u = a_1^m$, we get

$$g(a_1^m) = f(a_1^m, a_1^m) = f(a_1^m, a_1^m)a_1,$$

a contradiction. □

Thus, either a universal function for $\mathcal{F}$ is partial, or it is not in $\mathcal{F}$.