1.7 The Primitive Recursive Functions

Historically the primitive recursive functions were defined for numerical functions (computing on the natural numbers).

Since one of our goals is to show that the RAM-computable functions are partial recursive, we define the primitive recursive functions as functions $f : (\Sigma^*)^m \to \Sigma^*$, where $\Sigma = \{a_1, \ldots, a_k\}$ is a finite alphabet.

As usual, by assuming that $\Sigma = \{a_1\}$, we can deal with numerical functions $f : \mathbb{N}^m \to \mathbb{N}$.

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

Definition 1.13. Let $\Sigma = \{a_1, \ldots, a_k\}$. 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, $1 \leq j \leq k$, 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, duplicate, or drop the arguments of another function.

In the special case where we are only considering numerical functions ($\Sigma = \{a_1\}$), the function $E\colon \mathbb{N} \to \mathbb{N}$ is the *zero function* given by $E(n) = 0$ for all $n \in \mathbb{Z}$, and it is often denoted by Z.

There is a single *successor function* $S_{a_1}\colon \mathbb{N} \to \mathbb{N}$ usually denoted S (or **Succ**) given by $S(n) = n+1$ for all $n \in \mathbb{N}$.

Even though in this section we are primarily interested in total functions, later on, the same closure operations will be applied to partial functions so we state the definition of the closure operations in the more general case of partial functions.

The first closure operation is (extended) composition.

Definition 1.14. Let $\Sigma = \{a_1, \ldots, a_k\}$. For any partial or total function

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

and any $m \geq 1$ partial or total functions

$$h_i\colon \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{n} \to \Sigma^*, \quad n \geq 1,$$

the *composition of g and the h_i* is the partial 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 $w_1, \ldots, w_n \in \Sigma^*$. If g and all the h_i are total functions, then $g \circ (h_1, \ldots, h_m)$ is obviously a total function.

But if g or any of the h_i is a partial function, then the value $(g \circ (h_1, \ldots, h_m))(x_1, \ldots, x_n)$ *is defined if and only if all the values* $h_i(x_1, \ldots, x_n)$ *are defined for* $i = 1, \ldots, m$, *and* $g(h_1(w_1, \ldots, w_n), \ldots, h_m(w_1, \ldots, w_n))$ *is defined.*

Thus even if g "ignores" some of its inputs, in computing $g(h_1(w_1, \ldots, w_n), \ldots, h_m(w_1, \ldots, w_n))$, all arguments $h_i(w_1, \ldots, w_n)$ *must* be evaluated.

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

$$f(w_1, w_2) = g(P_2^2(w_1, w_2), P_1^2(w_1, w_2)) = g(w_2, w_1).$$

The second closure operation is *primitive recursion*. First we define primitive recursion for numerical functions because it is simpler.

Definition 1.15. Given any two partial or total functions $g\colon \mathbb{N}^{m-1} \to \mathbb{N}$ and $h\colon \mathbb{N}^{m+1} \to \mathbb{N}$ $(m \geq 2)$, the partial or total function $f\colon \mathbb{N}^m \to \mathbb{N}$ is defined by *primitive recursion from g and h* if f is given by

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

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

When $m = 1$, we have

$$\begin{aligned} f(0) &= b, \\ f(n+1) &= h(n, f(n)), \quad \text{for all } n \in \mathbb{N}, \end{aligned}$$

for some fixed natural number $b \in \mathbb{N}$.

If g and h are total functions, it is easy to show that f is also a total function.

If g or h is partial, obviously $f(0, x_2, \ldots, x_m)$ is defined iff $g(x_2, \ldots, x_m)$ is defined, and $f(n+1, x_2, \ldots, x_m)$ is defined iff $f(n, x_2, \ldots, x_m)$ is defined and
$h(n, f(n, x_2, \ldots, x_m), x_2, \ldots, x_m)$ is defined.

Definition 1.15 is quite a straightjacket in the sense that $n+1$ must be the first argument of f, and the definition only applies if h has $m+1$ arguments, but in practice a "natural" definition often ignores the argument n and some of the arguments $x_2, \ldots, x_m$.

This is where the projection functions come into play to drop, duplicate, or permute arguments.

For example, a "natural" definition of the *predecessor function* $pred$ is

$$\begin{aligned} pred(0) &= 0 \\ pred(m+1) &= m, \end{aligned}$$

but this is not a legal primitive recursive definition.

To make it a legal primitive recursive definition we need the function $h = P_1^2$, and a legal primitive recursive definition for $pred$ is

$$\begin{aligned} pred(0) &= 0 \\ pred(m+1) &= P_1^2(m, pred(m)). \end{aligned}$$

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

$$
\begin{aligned}
add(0,n) &= P_1^1(n) = n,\\
add(m+1,n) &= S \circ P_2^3(m, add(m,n), n)\\
&= S(add(m,n))\\
mult(0,n) &= E(n) = 0,\\
mult(m+1,n) &= add \circ (P_2^3, P_3^3)(m, mult(m,n), n)\\
&= add(mult(m,n), n),\\
rexp(0,n) &= S \circ E(n) = 1,\\
rexp(m+1,n) &= mult(rexp(m,n), n),\\
exp(m,n) &= rexp \circ (P_2^2, P_1^2)(m,n),\\
supexp(0,n) &= 1,\\
supexp(m+1,n) &= exp(n, supexp(m,n)).
\end{aligned}
$$

We usually write $m+n$ for $add(m,n)$, $m*n$ or even mn for $mult(m,n)$, and m^n for $exp(m,n)$.

There is a minus operation on $\mathbb{N}$ named *monus*. This operation denoted by $\dot{-}$ is defined by

$$m \dot{-} n = \begin{cases} m - n & \text{if } m \geq n \\ 0 & \text{if } m < n. \end{cases}$$

Then *monus* is defined by

$$\begin{aligned} m \dot{-} 0 &= m \\ m \dot{-} (n+1) &= pred(m \dot{-} n), \end{aligned}$$

except that the above is not a legal primitive recursion. For one thing, recursion should be performed on m, not n.

We can define $rmonus$ as

$$rmonus(n, m) = m \dot{-} n,$$

and then $m \dot{-} n = (rmonus \circ (P_2^2, P_1^2))(m, n)$, and

$$\begin{aligned} rmonus(0 \dot{-} m) &= P_1^1(m) \\ rmonus(n+1, m) &= pred \circ P_2^2(n, rmonus(n, m)). \end{aligned}$$

The following functions are also primitive recursive:

$$sg(n) = \begin{cases} 1 & \text{if } n > 0 \\ 0 & \text{if } n = 0, \end{cases}$$

$$\overline{sg}(n) = \begin{cases} 0 & \text{if } n > 0 \\ 1 & \text{if } n = 0, \end{cases}$$

as well as

$$abs(m, n) = |m - m| = m \dot{-} n + n \dot{-} m,$$

and

$$eq(m, n) = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n. \end{cases}$$

Finally, the function

$$cond(m, n, p, q) = \begin{cases} p & \text{if } m = n \\ q & \text{if } m \neq n, \end{cases}$$

is primitive recursive since

$$cond(m, n, p, q) = eq(m, n) * p + \overline{sg}(eq(m, n)) * q.$$

We can also design more general version of *cond*. For example, define $compare_{\leq}$ as

$$compare_{\leq}(m, n) = \begin{cases} 1 & \text{if } m \leq n \\ 0 & \text{if } m > n, \end{cases}$$

which is given by

$$compare_{\leq}(m, n) = 1 \dot{-} sg(m \dot{-} n).$$

Then we can define

$$cond_{\leq}(m, n, p, q) = \begin{cases} p & \text{if } m \leq n \\ q & \text{if } m > n, \end{cases}$$

with

$$cond_{\leq}(m, n, n, p) = compare_{\leq}(m, n) * p \\ + \overline{sg}(compare_{\leq}(m, n)) * q.$$

The above allows to define functions by cases.

We now generalize primitive recursion to functions defined on strings (in Σ^*).

The new twist is that instead of the argument $n+1$ of f, we need to consider the k arguments ua_i of f for $i = 1, \ldots, k$ (with $u \in \Sigma^*$), so instead of a single function h, we need k functions h_i to define primitive recursively what $f(ua_i, w_2, \ldots, w_m)$ is.

Definition 1.16. Let $\Sigma = \{a_1, \ldots, a_k\}$. For any partial or total function

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

where $m \geq 2$, and any k partial or total functions

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

the partial function

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

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

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

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

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

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

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

Again, if g and the h_i are total, it is easy to see that f is total.

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 $1 \leq i \leq k$.

It is easily verified that $g(u, v) = vu$.

Then,

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

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

Here are some primitive recursive functions that often appear as building blocks for other primitive recursive functions.

The *delete last* function $dell$ given by

$$\begin{aligned} dell(\epsilon) &= \epsilon \\ del(ua_i) &= u, \quad 1 \leq i \leq k,\ u \in \Sigma^* \end{aligned}$$

is defined primitive recursively by

$$\begin{aligned} dell(\epsilon) &= \epsilon \\ dell(ua_i) &= P_1^2(u, dell(u)), \quad 1 \leq i \leq k,\ u \in \Sigma^*. \end{aligned}$$

For every string $w \in \Sigma^*$, the *constant function* c_w given by

$$c_w(u) = w \quad \text{for all } u \in \Sigma^*$$

is defined primitive recursively by induction on the length of w by

$$\begin{aligned} c_\epsilon &= E \\ c_{va_i} &= S_i \circ c_v, \quad 1 \leq i \leq k. \end{aligned}$$

The *sign function* sg given by

$$sg(x) = \begin{cases} \epsilon & \text{if } x = \epsilon \\ a_1 & \text{if } x \neq \epsilon \end{cases}$$

is defined primitive recursively by

$$\begin{aligned} sg(\epsilon) &= \epsilon \\ sg(ua_i) &= (c_{a_1} \circ P_1^2)(u, sg(u)). \end{aligned}$$

The *anti-sign function* $\overline{sg}$ given by

$$\overline{sg}(x) = \begin{cases} a_1 & \text{if } x = \epsilon \\ \epsilon & \text{if } x \neq \epsilon \end{cases}$$

is primitive recursive. The proof is left an an exercise.

The function end_j $(1 \leq j \leq k)$ given by

$$end_j(x) = \begin{cases} a_1 & \text{if } x \text{ ends with } a_j \\ \epsilon & \text{otherwise} \end{cases}$$

is primitive recursive. The proof is left an an exercise.

The *reverse function* $rev\colon \Sigma^* \to \Sigma^*$ given by $rev(u) = u^R$ is primitive recursive, because

$$\begin{aligned} rev(\epsilon) &= \epsilon \\ rev(ua_i) &= (con \circ (c_{a_i} \circ P_1^2, P_2^2))(u, rev(u)), \quad 1 \leq i \leq k. \end{aligned}$$

The *tail function* $tail$ given by

$$\begin{aligned} tail\epsilon) &= \epsilon \\ tail(a_i u) &= u \end{aligned}$$

is primitive recursive, because

$$tail = rev \circ dell \circ rev.$$

The *last function* $last$ given by

$$\begin{aligned} last(\epsilon) &= \epsilon \\ last(ua_i) &= a_i \end{aligned}$$

is primitive recursive, because

$$\begin{aligned} last(\epsilon) &= \epsilon \\ last(ua_i) &= c_{a_i} \circ P_1^2(u, last(u)). \end{aligned}$$

The *head function* $head$ given by

$$\begin{aligned} head(\epsilon) &= \epsilon \\ head(a_i u) &= a_i \end{aligned}$$

is primitive recursive, because

$$head = last \circ rev.$$

We are now ready to define the class of primitive recursive functions.

Definition 1.17. Let $\Sigma = \{a_1, \ldots, a_k\}$. The class of *primitive recursive functions* is the smallest class of (total) functions (over Σ^*) which contains the base functions and is closed under composition and primitive recursion.

In the special where $k = 1$, we obtain the class of *numerical primitive recursive functions*.

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 1.4. *For any alphabet* $\Sigma = \{a_1, \ldots, a_k\}$*, every primitive recursive function is RAM computable, and thus Turing computable.*

The proof is given in an appendix.

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

1.8 Primitive Recursive Predicates

Primitive recursive predicates will be used in Section 2.2.

Definition 1.18. An *n-ary predicate* P over $\mathbb{N}$ is any subset of $\mathbb{N}^n$. 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 $C_P \colon \mathbb{N}^n \to \{0, 1\}$ defined by

$$C_p(x_1, \ldots, x_n) = \begin{cases} 1 & \text{iff } P(x_1, \ldots, x_n) \text{ holds} \\ 0 & \text{iff } \textbf{not } P(x_1, \ldots, x_n). \end{cases}$$

A predicate P (over $\mathbb{N}$) is *primitive recursive* iff its characteristic function C_P is primitive recursive.

More generally, 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

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

A predicate P (over Σ^*) is *primitive recursive* iff its characteristic function C_P is primitive recursive.

Since we will only need to use primitive recursive predicates over $\mathbb{N}$ in the following chapters, for simplicity of exposition we will restrict ourselves to such predicates.

It is easily shown that if P and Q are primitive recursive predicates (over $(\mathbb{N}^n)$, 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,

prime(n) iff n is a prime number, is a primitive recursive predicate.

For any fixed $k \geq 1$, the function

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.

Proposition 1.5. *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:*

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

Here we write $\overline{x}$ for $(x_1, \ldots, x_n)$.

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

Recall that we are restricting our attention to numerical predicates and functions, so all variables range over $\mathbb{N}$.

Definition 1.19. If P is an $(n+1)$-ary predicate, then the *bounded existential predicate* $(\exists y \leq x)P(y, \overline{z})$ holds iff some $y \leq x$ makes $P(y, \overline{z})$ true.

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

Proposition 1.6. *If P is an $(n+1)$-ary primitive recursive predicate, then $(\exists y \leq x)P(y, \overline{z})$ and $(\forall y \leq x)P(y, \overline{z})$ are also primitive recursive predicates.*

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

The following slight generalization of Proposition 1.6 will be needed in Section 2.2.

Proposition 1.7. *If P is an $(n+1)$-ary primitive recursive predicate and $f : \mathbb{N}^n \to \mathbb{N}$ is a primitive recursive function, then $(\exists y \leq f(\overline{z}))P(y, \overline{z})$ is also a primitive recursive predicate.*

Definition 1.20. If P is an $(n+1)$-ary predicate, then the *bounded minimization of P*, $\min(y \leq x)\, P(y, \overline{z})$, is the function defined such that $\min(y \leq x)\, P(y, \overline{z})$ is the least natural number $y \leq x$ such that $P(y, \overline{z})$ if such a y exists, $x + 1$ otherwise.

The *bounded maximization of P*, $\max(y \leq x)\, P(y, \overline{z})$, is the function defined such that $\max(y \leq x)\, P(y, \overline{z})$ is the largest natural number $y \leq x$ such that $P(y, \overline{z})$ if such a y exists, $x + 1$ otherwise.

Proposition 1.8. *If P is an $(n+1)$-ary primitive recursive predicate, then $\min(y \leq x)\, P(y, \overline{z})$ and $\max(y \leq x)\, P(y, \overline{z})$ are primitive recursive functions.*

So far the primitive recursive functions do not yield all the Turing-computable functions.

The following proposition also 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).

Definition 1.21. 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 $u \in \Sigma^*$.

Proposition 1.9. *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 $\mathcal{F}$.

In order to get a larger class of functions, we need the closure operation known as minimization.

1.9 The Partial Computable Functions

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

First let us consider the simpler case of numerical functions.

Consider a function $g\colon \mathbb{N}^{m+1} \to \mathbb{N}$, with $m \geq 0$. We would like to know if for any fixed $n_1, \ldots, n_m \in \mathbb{N}$, the equation

$$g(n, n_1, \ldots, n_m) = 0 \quad \text{with respect to } n \in \mathbb{N}$$

has a solution $n \in \mathbb{N}$, and if so, we return the smallest such solution.

Thus we are defining a (partial) function $f\colon \mathbb{N}^m \to \mathbb{N}$ such that

$$f(n_1, \ldots, n_m) = \min\{n \in \mathbb{N} \mid g(n, n_1, \ldots, n_m) = 0\},$$

with the understanding that $f(n_1, \ldots, n_m)$ is undefined otherwise. If g is computed by a RAM program, computing $f(n_1, \ldots, n_m)$ corresponds to the while loop

$n := 0$;
while $g(n, n_1, \ldots, n_m) \neq 0$ **do**
$n := n + 1$;
endwhile
let $f(n_1, \ldots, n_m) = n$.

Definition 1.22. For any function $g\colon \mathbb{N}^{m+1} \to \mathbb{N}$, where $m \geq 0$, the function $f\colon \mathbb{N}^m \to \mathbb{N}$ is defined by *minimization from* g, if the following conditions hold for all $n_1, \ldots, n_m \in \mathbb{N}$:

(1) $f(n_1, \ldots, n_m)$ is defined iff there is some $n \in \mathbb{N}$ such that $g(p, n_1, \ldots, n_m)$ is defined for all p, $0 \leq p \leq n$, and

$$g(n, n_1, \ldots, n_m) = 0;$$

(2) When $f(n_1, \ldots, n_m)$ is defined,

$$f(n_1, \ldots, n_m) = n,$$

where n is such that $g(n, n_1, \ldots, n_m) = 0$ and $g(p, n_1, \ldots, n_m) \neq 0$ for every p, $0 \leq p \leq n - 1$. In other words, n is the smallest natural number such that $g(n, n_1, \ldots, n_m) = 0$.

Following Kleene, we write

$$f(n_1, \ldots, n_m) = \mu n[g(n, n_1, \ldots, n_m) = 0].$$

Remark: When $f(n_1, \ldots, n_m)$ is defined, $f(n_1, \ldots, n_m) = n$, where n is the smallest natural number such that condition (1) holds.

It is very important to require that all the values $g(p, n_1, \ldots, n_m)$ be defined for all p, $0 \leq p \leq n$, when defining $f(n_1, \ldots, n_m)$. Failure to do so allows non -computable functions.

Minimization can be generalized to functions defined on strings as follows.

Given a function $g\colon (\Sigma^*)^{m+1} \to \Sigma^*$, for any fixed $w_1, \ldots, w_m \in \Sigma^*$, we wish to solve the equation

$$g(u, w_1, \ldots, w_m) = \epsilon \quad \text{with respect to } u \in \Sigma^*,$$

and return the "smallest" solution u, if any.

The only issue is, what does smallest solution mean.

We resolve this issue by restricting u to be a string of a_j's, for some fixed letter $a_j \in \Sigma$.

Thus there are k variants of minimization corresponding to searching for a shortest string in $\{a_j\}^*$, for a fixed j, $1 \leq j \leq k$.

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

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

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

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

looks for the shortest string u over $\{a_j\}^*$ (for a fixed j) such that

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

This corresponds to the following while loop:

$u := \epsilon$;
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 1.23. Let $\Sigma = \{a_1, \ldots, a_k\}$. For any function

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

where $m \geq 0$, for every j, $1 \leq j \leq k$, 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 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 natural number 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: Inspired by Kleene's notation in the case of numerical functions, we may use the *μ-notation*:

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

The class of partial computable functions is defined as follows.

Definition 1.24. Let $\Sigma = \{a_1, \ldots, a_k\}$. The class of *partial computable functions* (in the sense of Herbrand–Gödel–Kleene), also called *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 *computable functions* also called *recursive functions* is the subset of the class of partial computable 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 1.10. *For an alphabet $\Sigma = \{a_1, \ldots, a_k\}$, every partial computable function (partial recursive function) is RAM-computable, and thus Turing -computable. Conversely, every Turing-computable function is a partial computable function (partial recursive function). Similarly, the class of computable functions (recursive functions) is equal to the class of Turing-computable functions that halt in a proper ID for every input.*

First we prove that every partial computable function is RAM-computable, which is not that difficult because composition, primitive recursion, and minimization are easily implemented using RAM programs.

By Theorem 1.2, every RAM program can be converted to a Turing machine, so every partial computable function is Turing-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 computable function needs minimization *at most once*. The characterization of the computable functions in terms of TM's follows easily.

We will prove in Section 2.2 that every RAM-computable function (over $\mathbb{N}$) is partial computable. This will be done by encoding RAM programs as natural numbers.

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

Ackermann's function:

This is a function $A\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N}$ which is defined by the following recursive clauses:

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

It turns out that A is a computable function which is **not** primitive recursive.

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

It is possible to show that A is a computable (recursive) function, although the quickest way to prove it requires some fancy machinery (the recursion theorem). Proving that A is *not* primitive recursive is even harder.