THEORY OF COMPUTATION
Primitive Recursive Functions - 4

Prof. Dan A. Simovici

UMB

1 Function Composition

2 Recursion

3 Primitive Recursively Closed Classes

4 Building the Class of Primitive Recursive Functions

Definition

Let f be a function of k variables and let $g_1, \ldots, g_k$ be functions of n variables. The function h defined as

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

is said to be obtained from f and $g_1, \ldots, g_k$ by *composition*.

The functions $f, g_1, \ldots, g_k$ need not be total. $h(x_1, \ldots, x_n)$ is defined when all of $z_1 = g_1(x_1, \ldots, x_n), \ldots, z_k = g(x_1, \ldots, x_n)$ are defined and $f(z_1, \ldots, z_n)$ is defined.

Theorem

If h is obtained from the computable functions $f, g_1, \ldots, g_k$ by composition, then h is computable.

Proof.

The following programm computes h:

$$\begin{aligned}
&Z_1 \leftarrow g_1(X_1, \ldots, X_n)\\
&\vdots\\
&Z_k \leftarrow g_k(X_1, \ldots, X_n)\\
&Y \leftarrow f(Z_1, \ldots, Z_k)
\end{aligned}$$

$\square$

Example

We saw that the functions

$$x, x+y, x \cdot y, x-y$$

are partially computable. Therefore, $2x = x + x$ and $4x^2 = (2x) \cdot (2x)$ are partially computable. So are $4x^2 + 2x$ and $4x^2 - 2x$. Note that $4x^2 - 2x$ is total although is obtained from a non-total function $x - y$ by composition with $4x^2$ and $2x$.

Recursion is a modality of constructing a new function from a given one.

Definition

Suppose that g is a total function of two variables and k is a fixed number, $k \in \mathbb{N}$.
The function $h : \mathbb{N} \longrightarrow \mathbb{N}$ is obtained from g by primitive recursion if

$$
\begin{aligned}
h(0) &= k, \\
h(t+1) &= g(t, h(t)).
\end{aligned}
$$

Theorem

If h is obtained from the computable function g by primitive recursion, then h is also computable.

Proof.

Note that the constant function $f(x) = k$ is computed by the program

$$
\begin{aligned}
& Y \leftarrow Y + 1 \\
& Y \leftarrow Y + 1 \\
& \vdots \\
& Y \leftarrow Y + 1
\end{aligned}
$$

that contains k lines. $\square$

Proof cont'd

Proof.

This shows that we can use the macro $Y \leftarrow k$. The following is a program that computed $h(x)$:

$$\begin{array}{ll} & Y \leftarrow k \\ [A] & \text{IF } X = 0 \text{ GOTO } E \\ & Y \leftarrow g(Z, Y) \\ & Z \leftarrow Z + 1 \\ & X \leftarrow X - 1 \\ & \text{GOTO } A \end{array}$$

Note that if Y has the value $h(z)$ before executing instruction labeled A, then it has the value $g(z, h(z)) = h(z+1)$ after executing $Y \leftarrow g(Z, Y)$. □

A slightly more complicated kind of recursion

The function $h : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ is defined starting from the functions $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ and $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ as

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n), \\ h(x_1, \ldots, x_n, t+1) &= g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n). \end{aligned}$$

This modality of constructing h is known as *primitive recursion*.
The functions f and g are total.

Theorem

Let $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ *and* $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ *be two computable functions. The function* h *defined from* f *and* g *by primitive recursion is computable.*

Proof.

The following program computes $h(x_1, \ldots, x_n, x_{n+1})$:

$$
\begin{array}{ll}
 & Y \leftarrow f(X_1, \ldots, X_n) \\
[A] & \text{IF } X_{n+1} = 0 \text{ GOTO } E \\
 & Y \leftarrow g(Z, Y, X_1, \ldots, X_n) \\
 & Z \leftarrow Z + 1 \\
 & X_{n+1} \leftarrow X_{n+1} - 1 \\
 & \text{GOTO } A
\end{array}
$$

$\square$

Definition

The set of *initial functions* consists of the following:

- the *successor function* $s : \mathbb{N} \longrightarrow \mathbb{N}$ defined by $s(x) = x + 1$ for $x \in \mathbb{N}$;
- the *null function* $n : \mathbb{N} \longrightarrow \mathbb{N}$ defined by $n(x) = 0$ for $x \in \mathbb{N}$;
- the *projection functions* $u_i^n : \mathbb{N}^n \longrightarrow \mathbb{N}$ given by $u_i^n(x_1, \ldots, x_n) = x_i$ for $1 \leqslant i \leqslant n$.

Note that because the initial functions contain the projection functions, the class of initial functions contains an infinite number of functions.

Example

The *projection function* $u_2^5 : \mathbb{N}^5 \longrightarrow \mathbb{N}$ is given by

$$u_2^5(x_1, x_2, x_3, x_4, x_5) = x_2$$

for $x_1, x_2, x_3, x_4, x_5 \in \mathbb{N}$.

Definition

A *primitive recursively closed class* (a PRC class) is a set of total functions $\mathcal{C}$ that satisfies the following conditions:

1. the initial functions belong to $\mathcal{C}$, and
2. a function obtained from functions belonging to $\mathcal{C}$ by either composition or recursion belongs to $\mathcal{C}$.

Theorem

The class of computable functions is a PRC class.

Proof.

It suffices to show that the initial functions are computable.

- The function $s(x) = x + 1$ is computable by $Y \leftarrow X + 1$.
- $n(x)$ is computed by the empty program, and
- $u_i^n(x_1, \ldots, x_n)$ is computed by the program

$$Y \leftarrow X_i$$

□

Definition

A function is *primitive recursive* if it can be obtained from the initial functions by a finite number of applications of composition and recursion.

It is clear that the class of primitive recursive functions is a PRC class.

Theorem

A function is primitive recursive if and only if it belongs to every PRC class.

Proof.

If a function belongs to every PRC class then, in particular, it belongs to the class of primitive recursive functions.
Conversely, let f be a primitive recursive function and let $\mathcal{C}$ be some PRC class.
Since f is primitive recursive, there is a list $f_1, \ldots, f_n$ of functions such that $f_n = f$ and each f_i is either an initial function or it can be obtained from preceeding functions by composition or recursion.
The initial functions belong to $\mathcal{C}$ and we saw that the application of composition or recursion to functions in $\mathcal{C}$ results in a function in $\mathcal{C}$. Hence any function in $f_1, \ldots, f_n$ belongs to $\mathcal{C}$. In particular, $f_n = f \in \mathcal{C}$. □

Corollary

Every primitive recursive function is computable.

Proof.

Every primitive recursive function belongs to the PRC class of computable functions. □

Example

Let $f(x, y) = x + y$. We have

$$\begin{aligned} f(x,0) &= x = u_1^1(x), \\ f(x,y+1) &= f(x,y)+1. \end{aligned}$$

The second equality can be written as

$$f(x, y+1) = g(y, f(x,y), x),$$

where

$$g(x_1, x_2, x_3) = 1 + x_2 = s(u_2^3(x_1, x_2, x_3)).$$

Thus, g is primitive recursive and f is primitive recursive because is obtained by primitive recursion from primitive recursive functions.

Example

Let $h(x, y) = x \cdot y$. We have:

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

or

$$\begin{aligned} h(x, 0) &= n(x), \\ h(x, y+1) &= g(y, h(x, y), x), \end{aligned}$$

where

$$g(x_1, x_2, x_3) = f(u_2^3(x_1, x_2, x_3), u_3^3(x_1, x_2, x_3)) = f(x_2, x_3),$$

where $f(x, y) = x + y$ was shown to be primitive recursive on Slide 19.

Example

Let $\ell : \mathbb{N} \longrightarrow \mathbb{N}$ be defined as $\ell(x) = x!$. The recursion equations are $\ell(0) = 1$ and $\ell(x+1) = \ell(x) \cdot s(x)$, which represent the equalities:

$$0! = 1 \text{ and } (x+1)! = x!(x+1).$$

Formally, we have:

$$\begin{aligned} \ell(0) &= 1, \\ \ell(t+1) &= g(t, \ell(t)), \end{aligned}$$

where $g(x_1, x_2) = s(x_1) \cdot x_2$. The function g is primitive recursive because $g(x_1, x_2) = s(u_1^2(x_1, x_2)) \cdot u_2^2(x_1, x_2)$ and multiplication is already known to be primitive recursive.

Example

The exponentiation function x^y:
The recursion equations are

$$\begin{aligned} x^0 &= 1, \\ x^{y+1} &= x^y \cdot x \end{aligned}$$

Note that the for the “special case” 0^0 we have $0^0 = 1$.

Example

The *predecessor function* defined as

$$p(x) = \begin{cases} x - 1 & \text{if } x \neq 0, \\ 0 & \text{if } x = 0, \end{cases}$$

is primitive recursive because we have

$$\begin{aligned} p(0) &= 0, \\ p(t+1) &= t. \end{aligned}$$

The function $x \dot{-} y$ defined as

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geqslant y \\ 0 & \text{if } x < y \end{cases}$$

should not be confused with the partial function $x - y$ which is undefined if $x < y$. The function $x \dot{-} y$ is a total function and is defined by

$$\begin{aligned} x \dot{-} 0 &= x, \\ x \dot{-} (t+1) &= p(x \dot{-} t). \end{aligned}$$

Note: the symbol $\dot{-}$ is read “monus”.

Example

The function $|x - y|$ is primitive recursive because

$$|x - y| = (x \dot{-} y) + (y \dot{-} x).$$

Example

The function $\alpha(x)$, where

$$\alpha(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{if } x \neq 0, \end{cases}$$

is primitive recursive because $\alpha(x) = 1 \dot{-} x$.
Alternatively, we can write the recursion equations:

$$\begin{aligned} \alpha(0) &= 1, \\ \alpha(t+1) &= 0. \end{aligned}$$