

# CS5000: Lecture 20

Vladimir Kulyukin

Department of Computer Science

Utah State University

# Outline

- Composition and Recursion
- Primitive Recursive Closure

# Review: Composition

Let  $f$  be a function of  $k$  variables.

Let  $g_1, \dots, g_k$  be functions of  $n$  variables.

Let

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

Then  $h$  is obtained from  $f$  and  $g_1, \dots, g_k$  by composition.

# Review: Theorem 1.1 (Ch. 3)

If  $h$  is obtained from the (partially) computable functions  $f, g_1, \dots, g_k$  by composition, then  $h$  is (partially) computable.

# Example 1

Show that  $h(x) = 4x^2 + 2x$  is computable.

# Example 1

Proof :

$f(x, y) = x + y$  is computable.

$g_1(x) = 4x^2$  is computable.

$g_2(x) = 2x$  is computable.

$h(x) = f(g_1(x), g_2(x)) = 4x^2 + 2x$   
is computable (by Theorem 1.1).

# Example 2

Show that  $h(x) = 4x^2 - 2x$  is computable.

# Example 2

Proof :

$f(x, y) = x - y$  is partially computable.

$g_1(x) = 4x^2$  is computable.

$g_2(x) = 2x$  is computable.

Then

$h(x) = f(g_1(x), g_2(x))$  is computable

(by Theorem 1.1).

# Recursion

Suppose  $k$  is some fixed number.

Let  $g(x, y)$  be a total function.

$$h(0) = k$$

$$h(t + 1) = g(t, h(t))$$

Then  $h$  is obtained from  $g$  by  
primitive recursion (recursion).

# Example

$$h(0) = k$$

$$h(t + 1) = g(t, h(t))$$

What is  $h(2)$ ?

$$h(2) = h(1 + 1)$$

$$= g(1, h(1))$$

$$= g(1, h(0 + 1))$$

$$= g(1, g(0, h(0)))$$

$$= g(1, g(0, k))$$

# Theorem 2.1 (Ch. 3)

Let  $h$  be obtained from  $g$  by primitive recursion. Let  $g$  be computable. Then  $h$  is computable.

# Proof 2.1

$f(x) = k$  is computable.

Why?

$$\left. \begin{array}{l} Y \leftarrow Y + 1 \\ Y \leftarrow Y + 1 \\ \dots \\ Y \leftarrow Y + 1 \end{array} \right\} k \text{ lines}$$

# Proof 2.1

$h(x)$  is computed by the following  
program :

$Y \leftarrow k$

[A] IF  $X = 0$  GOTO  $E$

$Y \leftarrow g(Z, Y)$

$Z \leftarrow Z + 1$

$X \leftarrow X - 1$

GOTO A

# Proof 2.1

- Why does this program work? Suppose  $X = j$
- $Y$  takes the values  $h(0), h(1), \dots, h(j)$
- $Y = k = h(0)$
- $Y = g(0, h(0)) = g(0, k) = h(1)$
- $Y = g(1, h(1)) = g(1, g(0, k)) = h(2)$
- $Y = g(2, h(2)) = g(2, g(1, g(0, k))) = h(3)$
- ...
- $Y = g(j-1, h(j-1)) = h(j)$

# Recursion: Definition 2

$$h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n)$$

$$h(x_1, \dots, x_n, t + 1) = g(t, h(x_1, \dots, x_n, t), x_1, \dots, x_n)$$

# Theorem 2.2 (Ch. 3)

Let  $h$  be obtained from  $f$  and  $g$  by primitive recursion and let  $f$  and  $g$  be computable. Then  $h$  is computable.

# Proof 2.2

$$Y \leftarrow f(X_1, \dots, X_n)$$

[A] IF  $X_{n+1} = 0$  GOTO  $E$

$$Y \leftarrow g(Z, Y, X_1, \dots, X_n)$$

$$Z \leftarrow Z + 1$$

$$X_{n+1} \leftarrow X_{n+1} - 1$$

GOTO A

# Proof 2.2

- Why does this program work? Suppose  $X_{n+1} = j$
- $Y$  takes the values  $h(0), h(1), \dots, h(j)$
- $Y = f(x_1, x_2, \dots, x_n)$
- $Y = g(0, h(x_1, x_2, \dots, x_n, 0), x_1, x_2, \dots, x_n)$
- $Y = g(1, h(x_1, x_2, \dots, x_n, 1), x_1, x_2, \dots, x_n)$
- $Y = g(2, h(x_1, x_2, \dots, x_n, 2), x_1, x_2, \dots, x_n)$
- ...
- $Y = g(j-1, h(x_1, x_2, \dots, x_n, j-1), x_1, x_2, \dots, x_n)$

# A Perspective on Primitive Computation

Primitive Computation = Composition +  
Stack (Memory)

# Primitives in Formal Systems

- Any formal system must define its primitives, i.e. elements that cannot be decomposed any further.
- Examples:
  - Points in Euclidean geometry
  - Elementary Particles in physics
  - Numbers in mathematics
  - Assembly operations in compilers

# Primitives of Computability

- Theory of primitive recursive functions also has its primitives: they are called ***initial functions***.
- Why are they called primitives? Because we can effectively construct more complicated functions out of them through composition and primitive recursion.

# Initial Functions

1. The successor function :  $s(x) = x + 1$
2. The null function :  $n(x) = 0$
3. The projection function :  $u_i^n(x_1, \dots, x_n) = x_i, 1 \leq i \leq n$

# PRC: Primitive Recursively Closed

- A class of *total* functions  $\mathbf{C}$  is a PRC class if
  - The initial functions are in  $\mathbf{C}$ ;
  - Any function obtained from the functions already in  $\mathbf{C}$  by *composition* or *recursion* is also in  $\mathbf{C}$ .

# Recommended Reading

- Sections 2.1, 2.2, 2.3