ICOM 4075:
Foundations of Computing

Lecture 6
Functions (2)

Department of Electrical and Computer Engineering
University of Puerto Rico at Mayagüez

Lecture Notes Originally Written By Prof. Yi Qian

Homework 4 – Due Tuesday March 9, 2010

- Section 2.1: (pp.88-90)

1. b. d.
2. b. d. f.
3. b. d.
4. b.
6. b. d.
7. b. d. f.
8. b.
9.
10.
11. b.
13. b. d.
14. b. d.
15. b. d. f.
17.

Reading

- Textbook: James L. Hein, *Discrete Structures, Logic, and Computability*, 2nd edition, Chapter 2. Section 2.2

Constructing Functions

- Composition of Functions
  - *Composition* of functions is a natural process that we often use without even thinking.
    - E.g., floor($\log_2(6)$) involves the composition of the two functions floor and $\log_2$. To evaluate the expression, we first evaluate $\log_2(6)$, which is a number between 2 and 3. Then we apply the floor function to this number, obtaining the value 2.
- Definition of Composition
  - The *composition* of two functions f and g is the function denoted by $f \circ g$ and defined by $(f \circ g)(x) = f(g(x))$.
- Notice that composition makes sense only for values of x in the domain of g such that g(x) is in the domain of f.
  - So if $g: A \to B$ and $f: C \to D$ and $B \subset C$, then the composition $f \circ g$ makes sense. In other words, for every $x \in A$ it follows that $g(x) \in B$, and since $B \subseteq C$ it follows that $f(g(x)) \in D$. It also follows that $f \circ g: A \to D$.
  - E.g., $\log_2: R^+ \to R$ and floor: $R \to Z$, where R^+ denotes the set of positive real numbers. So for any positive real number x, the expression $\log_2(x)$ is a real number and thus floor($\log_2(x)$) is an integer. So the composition $\text{floor} \circ \log_2: R^+ \to Z$.

Composition of Functions

- Composition of functions is *associative*:
  - If f, g, and h are functions of the right type such that $(f\circ g)\circ h$ and $f\circ(g\circ h)$ make sense, then $(f\circ g)\circ h = f\circ(g\circ h)$.
- Composition of functions is *not commutative*:
  - E.g., suppose that f and g are defined by $f(x) = x + 1$ and $g(x) = x^2$. To show that $f\circ g \neq g\circ f$, we only need to find one number x such that $(f\circ g)(x) \neq (g\circ f)(x)$. We'll try $x = 3$ and observe that

    $(f\circ g)(3) = f(g(3)) = f(3^2) = 3^2 + 1 = 10.$

    $(g\circ f)(3) = g(f(3)) = g(3 + 1) = (3 + 1)^2 = 16.$

    Therefore, $(f\circ g)(3) \neq (g\circ f)(3)$
- A function that always returns its argument is called an *identity* function. For a set A we sometimes write "id_A" to denote the identity function defined by $id_A(a) = a$ for all $a \in A$. If $f: A \rightarrow B$, then we always have the following equation: $f\circ id_A = f = id_B \circ f$

The Sequence, Distribute, and Pairs Functions

- The *sequence* function “seq” has type N→lists(N) and is defined as follows for any natural number n: seq(n) = <0, 1, …, n>.
  - E.g., seq(0) = <0>, seq(2) = <0, 1, 2>, seq(5) = <0, 1, 2, 3, 4, 5>.

- The *distribute* function “dist” has type A x lists(B) → lists(A x B). It takes an element x from A and a list y from lists(B) and returns the list of pairs made up by pairing x with each element of y.
  - E.g., dist(x, <r, s, t>) = <(x, r), (x, s), (x, t)>.

- The *pairs* function takes two lists of equal length and returns the list of pairs of corresponding elements.
  - E.g., pairs(<a, b, c>, <d, e, f>) = <(a, d), (b, e), (c, f)>.
  - Since the domain of pairs is a proper subset of lists(A) x lists(B), it is a partial function of type lists(A) x lists(B) → lists(A x B).

Composing Functions with Different Arities

- Composition can also occur between functions with different arities.
  - E.g., suppose we define the following function f(x, y) = dist(x, seq(y)). In this case dist has two arguments and seq has one argument. For example, we'll evaluate the expression f(5, 3).

f(5, 3) = dist(5, seq(3))
= dist(5, <0, 1, 2, 3>)
= <(5, 0), (5, 1), (5, 2), (5, 3)>.

Distribute a Sequence

- We'll show that the definition f(x, y) = dist(x, seq(y)) is a special case of the following more general form of *composition*, where X can be replaced by any number of arguments. $f(X) = h(g_1(X),\ldots,g_n(X))$.
- Distribute a Sequence
  - We'll show that the definition f(x, y) = dist(x, seq(y)) fits the general form of composition. To make it fit the form, we'll define the functions one(x, y) = x and two(x, y) = y. Then we have the following representation of f.

    f(x, y) = dist(x, seq(y))

    = dist(one(x, y), seq(two(x, y)))

    = dist(one(x, y), (seq○two(x, y))).

    The last expression has the general form of composition

    $f(X) = h(g_1(X), g_2(X))$,

    where X = (x, y), h = dist, g_1 = one, and g_2 = seq○two

The Max Function

- The Max Function
  - Suppose we define the function “max”, to return the maximum of two numbers as follows:

    max(x, y) = if x < y then y else x.

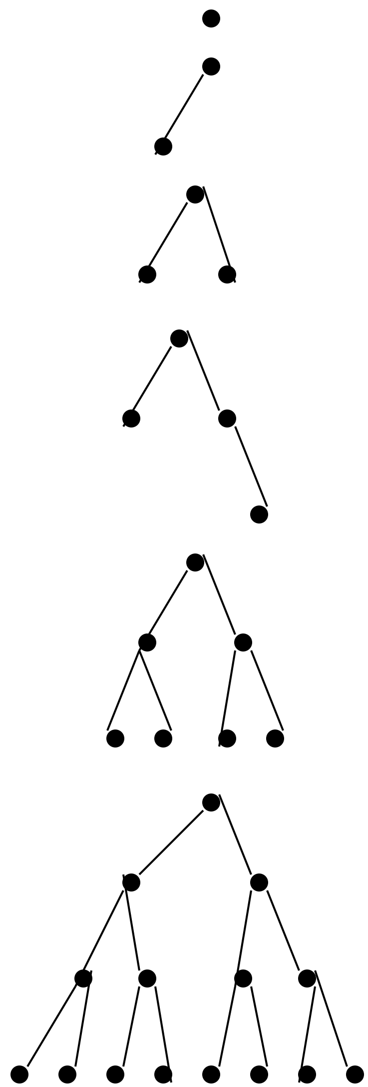
    Then we can use max to define the function “max3”, which returns the maximum of three numbers, by the following composition:

    max3(x, y, z) = max(max(x, y), z).

Minimum Depth of a Binary Tree

- To find the minimum depth of a binary tree in terms of the numbers of nodes:
  - The following figure lists a few sample cases in which the trees are as compact as possible, which means that they have the least depth for the number of nodes. Let n denote the number of nodes. Notice that when $4 \le n < 8$, the depth is 2. Similarly, the depth is 3 whenever $8 \le n < 16$.
  - At the same time we know that $\log_2(4) = 2$, $\log_2(8) = 3$, and for $4 \le n < 8$ we have $2 \le \log_2(n) < 3$. So $\log_2(n)$ almost works as the depth function.
  - In general, we have the minimum depth function as the composition of the floor function and the $\log_2$ function:

    $\text{minDepth}(n) = \text{floor}(\log_2(n))$.



Binary tree	Nodes	Depth
	1	0
	2	1
	3	1
	4	2
	7	2
	15	3

List of Pairs

- Suppose we want to construct a definition for the following function in terms of known functions

  f(n) = <(0, 0), (1, 1), …, (n, n)> for any $n \in N$.

  Starting with this informal definition, we'll transform it into a composition of known functions.

  f(n) = <(0, 0), (1, 1), …, (n, n)>
  = pairs(<0, 1, …, n>, <0, 1, …, n>)
  = pairs(seq(n), seq(n)).

- Suppose we want to construct a definition for the following function in terms of known functions

  g(k) = <(k, 0), (k, 1), …, (k, k)> for any $k \in N$.

  Starting with this informal definition, we'll transform it into a composition of known functions.

  g(k) = <(k, 0), (k, 1), …, (k, k)>
  = dist(k, <0, 1, …, k>)
  = dist(k, seq(k)).

The Map Function

- Definition of the Map Function:
  - Let f be a function with domain A and let $<x_1, \ldots, x_n>$ be a list of elements from A. Then

    $map(f, <x_1, \ldots, x_n>) = <f(x_1), \ldots, f(x_n)>$.

    So the type of the map function can be written as

    map: $(A \rightarrow B) \times list(A) \rightarrow lists(B)$.
  - E.g.,

    map(floor, <-1.5, -0.5, 0.5, 1.5, 2.5>)

    = <floor(-1.5), floor(-0.5), floor(0.5), floor(1.5), floor(2.5)>

    = <-2, -1, 0, 1, 2>.

    $map(floor \circ log_2, <2, 3, 4, 5>)$

    $= <floor(log_2(2)), floor(log_2(3)), floor(log_2(4)), floor(log_2(5))>$

    = <1, 1, 2, 2>.

    map(+, <(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)>)

    = < +(1, 2), +(3, 4), +(5, 6), +(7, 8), +(9, 10)>

    = <3, 7, 11, 15, 19>
  - The map function is an example of a *higher-order* function, which is any function that either has a function as an argument or has a function as a value. This is an important property that most good programming languages possess.

A List of Squares

- Suppose we want to compute sequences of squares of natural numbers, such as 0, 1, 4, 9, 16. In other words, we want to compute f: N → lists(N) defined by f(n) = <0, 1, 4, …, n^2>. We have two different ways:
  - First way: define s(x) = x*x and then construct a definition for f in terms of map, s, and seq as follows.

    f(n) = <0, 1, 4, …, n^2>

    = <s(0), s(1), s(2), …, s(n)>

    = map(s, <0, 1, 2, …, n>)

    = map(s, seq(n)).
  - Second way: construct a definition for f without using the function s that we defined for the first way.

    f(n) = <0, 1, 4, …, n^2>

    = <0*0, 1*1, 2*2, …, n*n>

    = map(*, <(0, 0), (1, 1), (2, 2), …, (n, n)>)

    = map(*, pairs(<0, 1, 2, …, n>, <0, 1, 2, …, n>))

    = map(*, pairs(seq(n), seq(n))).