Types and Programming Languages by Benjamin Pierce
Chapter 5

5.2.2: Find another way to define the successor function on Church numerals.

در تعریف اصلی، n s بار روی z اعمال می‌شود تا c_n و یک بار دیگر s روی c_n اعمال شده تا c_{n+1} به‌دست بیاید.
می‌توان به n+1 بار s را روی z اعمال کنیم:

$scc_2 = \lambda n. \lambda s. \lambda z. n\ s\ (s\ z)$

5.2.3: Is it possible to define multiplication on Church numerals without using *plus*?

mxn بار s را به n اعمال کنیم:

$times_2 = \lambda m. \lambda n. \lambda s. m\ (n\ s)$

5.2.4: Define a term for raising one number to the power of another.

با توجه به تعریف times داریم:

$power = \lambda m. \lambda n. m\ (times\ n) c_1 = n^m$

5.2.6: Approximately how many steps of evaluation (as a function of *n*) are required to calculate *prd* c_n?

prd:

zz = pair c_0 c_0 --> 1
ss = λp. pair (snd p) (plus c_1 (snd p)) --> 1
prd = λn. fst (n ss zz) --> 1 + n

Number of steps: (1) + (1) + (1 + n) = (n + 3) steps = O(n)

5.2.7: Write a function *equal* that tests two numbers for equality and returns a Church Boolean.

اگر تفریق یک عدد از دیگری صفر شود، این دو عدد برابرند؛ با استفاده از تابع subtract داریم:

iseql = $\lambda m.\lambda n.\, and\ \left(iszro\ \left(subtract\ (m\ n)\right)\right)\left(iszro\ \left(subtract\ (n\ m)\right)\right)$

5.2.8: A list can be represented in the lambda calculus by its fold function. (OCaml's name for this function is *fold_left*; it is also sometimes called reduce.) For example, the list [x,y,z] becomes a function that takes two arguments c and n and returns c x (c y (c z n))).What would the representation of nil be? Write a function cons that takes an element h and a list (that is, a fold function) t and returns a similar representation of the list formed by prepending h to t. Write *isnil* and head functions, each taking a list parameter. Finally, write a tail function for this representation of lists (this is quite a bit harder and requires a trick analogous to the one used to define *prd* for numbers).

nil = $pair\ tru\ tru$
cons = $\lambda h.\lambda t.\, pair\ fls\ (pair\ h\ t)$
isnil = fst
head = $\lambda z.\, fst\ (snd\ z)$
tail = $\lambda z.\, snd\ (snd\ z)$

5.2.11: Use *fix* and the encoding of lists from Exercise 5.2.8 to write a function that sums lists of Church numerals.
fix(Z-combinator) = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y))

f = $\lambda f.\lambda t.\, text(isnil\ t)(\lambda x.\, c_0)(\lambda x.\, (plus(head\ t)(f(tail\ t))))c_0$

sum_list = $fix\ ff$

Concepts in Programming Languages by John C. Mitchell
Chapter 4

4.3: Lambda Calculus Reduction

$(\lambda x.\lambda y.xy)(\lambda x.xy) = (\lambda x.\lambda y_1.xy_1)(\lambda x.xy) = \lambda x.\lambda y_1.xyy_1 = \lambda y_1.xyy_1$

اگر اندیس‌گذاری نکنیم، Binderها و متغیرهای متفاوت مشخص نمی‌شود(name collision اتفاق می‌افتد).

4.4: Symbolic Evaluation

$\big((\lambda f.\lambda g.f(g\ 1))(\lambda x.x+4)\big)(\lambda y.3-y)$

a)

$\big((\lambda f.\lambda g.f(g\ 1))(\lambda x.x+4)\big)(\lambda y.3-y)$
$\rightarrow\big(\lambda g.(\lambda x.x+4)(g\ 1)\big)(\lambda y.3-y)$
$\rightarrow(\lambda x.x+4)((\lambda y.3-y)1)$
$\rightarrow\Big(\big((\lambda y.3-y)1\big)+4\Big) = (3-1)+4 = 6$

b)

$\big((\lambda f.\lambda g.f(g\ 1))(\lambda x.x+4)\big)(\lambda y.3-y)$
$\rightarrow\big(\lambda g.(\lambda x.x+4)(g\ 1)\big)(\lambda y.3-y)$
$\rightarrow(\lambda x.x+4)((\lambda y.3-y)1)$
$\rightarrow\Big(\big((\lambda x.x+4)2\big)\Big) = 2+4 = 6$

4.5: Lambda Reduction with Sugar

$(\lambda compose.(\lambda h.compose\ h\ h\ 3)\ \lambda x.x+x)\ \lambda f.\lambda g.\lambda x.f(g\ x)$
$\rightarrow\big(\lambda h.\big(\lambda f.\lambda g.\lambda x.f(g\ x)\big)h\ h\ 3\big)\lambda x.x+x$
$\rightarrow\Big(\big(\lambda f.\lambda g.\lambda x.f(g\ x)\big)(\lambda x.x+x)(\lambda x.x+x)3\Big)$
$\rightarrow(\ \lambda g.\lambda x.(\lambda x.x+x)(g\ x))(\lambda x.x+x)3$
$\rightarrow(\ \lambda x.(\lambda x.x+x)((\lambda x.x+x)\ x))3$
$\rightarrow(\ \lambda x.(\lambda x.x+x)((\lambda x.x+x)))3$
$\rightarrow\lambda x.((x+x)+(x+x))3$
$\rightarrow\big((3+3)+(3+3)\big) = 12$

4.6: Translation into Lambda Calculus

$$f = \lambda g.\, g\ g \rightarrow f\ f = (\lambda g.\, g\ g)(\lambda g.\, g\ g) = (\lambda g.\, g\ g)\big((\lambda g.\, g\ g)(\lambda g.\, g\ g)\big)$$
$$= (\lambda g.\, g\ g)\Big((\lambda g.\, g\ g)\big((\lambda g.\, g\ g)(\lambda g.\, g\ g)\big)\Big) = \cdots$$

این برنامه در main (f(f را فراخوانی می‌کند و این باعث می‌شود که f مرتباً خود را فراخوانی کند:

$$f(f) = f\big(f(f)\big) = f\Big(f\big(f(f)\big)\Big) = f\Big(f\big(f(f(f))\big)\Big) = \cdots$$

Exercise. Give a recursive definition of the function “plus” that takes two natural numbers and returns their sum in a call-by-value setting. Apply your function to some example arguments and check your answer.

Z = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y))
$Sum_{recursive}$ = Z (λf. λa. λb if b = 0 then a else (2 + f (a b-1))
$Sum_{recursive}$ = Z (λf. λa. λb. test (iszero b) c0 (plus c2 (f (subtract b c1 a))))