LAMBDA CALCULUS: AN INTRODUCTION

Chandreyee Chowdhury

LAMBDA CALCULUS TO PROGRAMMING

Data Types

- Booleans, numbers
- Collections

Conditional expressions

Arithmetic expressions

Recursions

a ***combinator*** is a λ-term with no free variables

PAIR

λleft. λright. (left)(right)

get_left = λleft. λright. (true)(left)(right)

get_right = λleft. λright. (false)(left)(right)

make_pair = λleft. λright. λf. f(left)(right)

make_pair(left_sock)(right_sock)

get_right(make_pair(left_sock)(right_sock))

get_right = λpair. pair(false)

GET_LEFT = ΛPAIR. PAIR(TRUE)
GET_RIGHT = ΛPAIR. PAIR(FALSE)
MAKE_PAIR = ΛLEFT. ΛRIGHT. ΛF. F(LEFT)(RIGHT)

get_right(make_pair(three)(two))

= get_right(λf. f(three)(two))

=(λpair. pair(false))(λf. f(three)(two))

=(λpair. pair(false))(λf. f(three)(two))

=(λf. f(three)(two)) (false)

=(false) (three)(two)

=two

LISTS

A list may contain

- Nothing (empty)
- One thing
- Multiple things

List contains 2 values

- make_pair = λleft. λright. λf. f(left)(right)
- get_left = λpair. pair(true)
- get_right = λpair. pair(false)

A list will have the form # (empty, (head, tail))

null=make_pair(true)(true)

is_empty = get_left

is_empty(null) would return true

LISTS

prepend = λitem. λl. make_pair(false)(make_pair(item)(l))

non-empty lists

- *[2, 1] ==> (empty=false, (2,(1,null)))*
- *# (false, (1,null))*
  - single_item_list = prepend(one)(null)
- *# (false, (3,(2,(1,null))))*
  - multi_item_list = prepend(three)(prepend(two)(single_item_list))

head = λl. get_left(get_right(l))

tail = λl.get_right(get_right(l))

PREDECESSOR

the general strategy will be to create a pair (n,n-1) and then pick the second element of the pair as the result

- make_pair = λleft. λright. λf. f(left)(right)

Left=λpair. pair(true)

Right= λpair. pair(false)

Φ combinator generates from the pair (n, n-1) (which is the argument p in the function) the pair (n + 1, n)

Φ =λp. λf. f (succ(p true))(p true)

(succ(p true))(p true)

(zero,zero)→(one,zero)→(two,one)→(three,two)→…

The predecessor of a number n is obtained by applying n times the function to the pair (f zero zero) and then selecting the second member of the new pair

Pred=λn. n Φ