Combinatory Logic

FP-Syd April 19, 2012

Fundamental Combinators   S K I

(I f)	===>	f
((K a) x)	===>	a
(((S f) g) x)	===>	((f x) (g x))

Identity,   Konstant,   Spreading

Fundamental Combinators S K I

I f	===>	f
K a x	===>	a
S f g x	===>	f x (g x)

Derived Combinators

B f g x ===> f (g x)

B := S (K S) K

C f g x ===> (f x) g

C := S (B B S) (K K)

I' x ===> x

I' := S K K

Substitution

[R / x] x ===> R

[R / x] M ===> M

[R / x] (Mx Nx) ===>
([R / x] Mx) ([R / x] Nx)

Warning - non-standard notation
M does not contain x, Mx and Nx might contain x.

Bracket Abstraction

[x].x	===>	I
[x].M	===>	K M
[x].(Mx Nx)	===>	S ([x].Mx) ([x].Nx)

$$([x].Mx)\ x \quad ===> \quad Mx$$

[x].x	===>	I
I x	===>	x
[x].M	===>	K M
K M x	===>	M
[x].(Mx Nx)	===>	S ([x].Mx) ([x].Nx)

S ([x].Mx) ([x].Nx) x

===> (([x].Mx) x) (([x].Nx) x)

===> Mx Nx

One argument is sufficient

$[x, y] . Mxy \quad := \quad ([x] . ([y] . Mxy))$

$[x, y, z] . Mxyz \quad := \quad ([x] . ([y] . ([z] . Mxyz)))$

...

Bracket Abstraction

([x].Mx) x ===> Mx

([x].Mx) N ===> [N / x] Mx

Bracket abstraction gives us an implementation of lambda calculus in S K I combinators.

Boolean Logic

Treat 'if', 'then' and 'else' as whitespace

if c then x else y ===> c x y

true x y ===> x

false x y ===> y

Boolean Logic

true := [x, y] . x   i.e.   K

K x y   ===>   x

false := [x, y] . y   i.e.   K I

(K I x) y   ===>   I y

===>   y

Boolean Logic

and := [c, d, x, y] . c (d x y) y

or := [c, d, x, y] . c x (d x y)

not := [c, x, y] . c y x

Arithmetic

For simplicity, we will use Church numerals, a unary representation of numbers as functions.

Once we have defined 'zero' and 'succ', we will have all natural numbers.

7 ===>

succ (succ (succ (succ (succ (succ (succ zero))))))

Arithmetic

zero := K I

zero x y	===>	(K I x) y
	===>	I y
	===>	y

succ := S B

(succ n) x y	===>	S B n x y
	===>	B x (n x) y
	===>	x (n x y)

7 x y	===>	(x (x (x (x (x (x (x y)))))))

Predecessor

pred := [x].(x ([u,v].v (u succ)) (K zero) I)

pred 0 ===> 0 ([u,v].v (u succ)) (K zero) I
===> K zero I
===> zero

pred (succ n) x y ===> (x (x … (x y))

Arithmetic

plus := [m, n].m succ n

minus := [m, n].n pred m

times := B
times m n x y ===> m (n x) y

exp := [m, n].n (times m) one

Arithmetic

zero := K I

zero x y ===> (K I x) y

===> I y

===> y

succ := S B

(succ n) x y ===> S B n x y

===> B x (n x) y

===> x (n x y)

7 x y ===> (x (x (x (x (x (x (x y)))))))

Data Structures

Let D x y represent a pair with elements x, y.

first (D x y) ===> x

second (D x y) ===> y

Data Structures

D := [x, y, f].f (K y) x

first := [d].d zero

first (D x y)	===>	D x y zero
	===>	zero (K y) x
	===>	x

second := [d].d one

second (D x y)	===>	D x y one
	===>	one (K y) x
	===>	K y x
	===>	y

Sample Application

```
factorial := [n].second
    (n
        ([p]. D
                (pred (first p))
                (times (first p) (second p)) )
        (D n one))
```