Mutual Recursion

```
fun even (N) = if N = 0
                 then true
                 else odd(N - 1)
and odd(N) = if N=0
               then false
               else even (N - 1) ;
```

even: int ⟶ bool

odd: int → bool

```
fun even(0) = true
  | even(N) = odd(N - 1)
and odd (0) = false
  | odd (N) = even (N - 1) ;
```

Overloading

+ : int · int → int ✓

+ : real · real ⟶ real ✓

+ : real · int → real ✗

```
fun square(x : real) = x * x ;
fun square (x) = x * (x : real) ;
fun square (x) = (x * x): real ;
```

All cases infer square : real ⟶ real

[1, 2, 3] : int list list

(1, "ab", 3.0) : int · string · real tupple record {name ="tony", age = 35, salary = 200} ; {name: string, age : int, salary: int}

$\{ f_1 : T_1 , f_2 : T_2 , \ldots , f_n : T_n \}$

is a record type with infields, with field names $f_1 , \ldots , f_n$ where field f_i has type T_i

An instance of this record type is $\{F_1 : V_1 , \ldots , F_n : V_n\}$ where value v_i has type T_i

f r => field f of record r

eg # salary {name= " tom ", age = 35, salary = 200}

=> 200

tupler

```
# 2 (7,"ab", "cd") => "ab"
# 2 {1 =7, 2 = "ab", 3= "cd"} => "ab"
```

Type declerations (named types)

```
type waitress = {name : str
                 wages : int
                 tips : int }

fun income (W:waidresses) = # wages (w)
                          + # tips (w);
```

```
fun income(w: waitress) = # wages(w)
                        + # tips(w);

    income: waitress → int
```

```
fun income(w) = # wages (w) + # tips (w);
```

type error

```
fun income {name = N, wages = W, tips T}
           = W + T;
```

```
income: {name: N, wages = W, tips = T} → int
```

```
type waitresse = waitress list
    [{name= "sally", wages = 20, tips = 10}
     {names = "alice", wages = 15, tips = 15}]: waitresses
```

```
fun total WL·waitresses =
        if WL = []
        then 0
        else income (hd WL)
             + total (tl WL);
  total·waitress → int
```

```
fun total ([]: waitresses) = 0
```

```
    | total (w:: WL) = income (w)
                        + total (WL);
```

Polymorphism

```
fun length L
     = if (null L) then 0
                 else 1+ length (tl L);
```

length [1, 2, 3, 4] ⟹ 4

length ["a", "b", "cd"] ⟹ 3

length [[1, 2] [3] [5, 6]] ⟹ 3

length 'b list → int   polymorphic

type variable

id (1) = 1   fun id x = x

id (ab) = "ab"

id ([2.0, 3.0]) = [2.0, 3.0]

id (1, "ab") = (1, "ab")

```
fun listify x = [x];
listify 3.0 ⟹ [3.0]
listify: 'a → 'a list
```

```
fun double X = (x, x)

double 7.3 => (7.3, 7.3)

double "a"=> ("a","a")

double true=> (true, true)

double [1, 3] =>([1,3], [1, 3])

        listify: 'a → 'a · 'a


fun inc(N, x) = (N+1), x);

inc (3 ,7.3)=> (4, 7.3)

inc(5, [1,7])=>(6, [1, 7])

            inc: int * 'a → int*'a


fun swap (X, Y) = (Y = X)

          swap: 'a * 'b → 'b * 'a


fun apply (F, X) = F(X);

apply (square, 3)=> square (3) = 9

apply(bd , [4,5,6])=>  bd [4,5,6] = 4

apply: ('a → 'b) * 'a → 'b


fun applyTwice(F,x)= f(F(x));

applyTwice(square, 2.0) = square (square 2.0)= 16.0

applyTwice(+1, [2,3,4,5]) = +1 (+1 [2,3,4,5])) =[4, 5]

            hd: 'a list → 'a

hd [1, 2, 3] => 1, hd[ "ab", "c "] => "ab"
```

applyTwice : (a → `a) * 'a ⟶ 'a

applyTwice (hd, [1, 2, 3, 4]) ⟹ type error

= hd(hd [1 2 3 4])

= hd(1)

= error

fun F(x, y, z) = (x + y, z)

f : int · int · 'a → int · 'a

: real · real · 'a → real · 'a

Named Polymorphic Types

'a · 'a

type 'a pair = 'a · 'a

(1, 2) : int pair

("bc", "cd") : str pair

fun ~~kg_to_lbs N = N * 2.2~~

~~kg-to-lbs : real → real~~

~~→ kg-to-lbs (kg-to-lbs !0)~~

type kg = real

type lb = real

User - Defined types

```
datatype pound = lbs of real                (lbs 3.0): pound

datatype kilogram = kg of real              (kg 3.0): kilogram

                              type constructors


fun kg_to_lbs(kg N) = (lbs 2.2 * N);

    kg_to_lbs: kilogram → pound


kg_to_lbs (kg 2.0) ⇒ (lbs 4.4)

kg_to_lbs(lbs 2.0) ⟹ type error

kg_to_lbs (2.0) ⟹ type error

kg_to_lbs (kg_to_lbs (kg 2.0))

          ⟹ type error
```