Typing Benefits

Debugging

Fast execution

Autodocumentation

Basic Types

int, real, string, bool

2 2.0 "abc" true

Complex Types

Int list, real list, nested list

hd [1, 2, 3] ⇒ 1

[1, 2, "ab", "cd"] ⇒ type error

[1, 2, [3, 4]] ⇒ type error

Tuple Types

(2, "abc") int · string

(2, 3.0, "abc") int · real · string

General Syntax

$T_1 * T_2 * \ldots * T_n$

represents a tuple with n field, where field i has type T_i.

((1, 2), "abc") (int · int) · string

#i t ⟹ field i of tuple t

#2 (6, 7, "abc") ⟹ 7

#3 (6, 7, "abc") ⟹ "abc"

(1, ("ab", 2), "cd") : int · (string · int) · string

#1 (#2 t) ⟹ "ab"

[[1, 2], [3, 4]] : (int list) list

(2, 3, [4, 5]) : int · int · (int list)

(2, (3.0, "ab"), "cd") : int · (real · str) · str

[(1, "a"), (3, "bc"), (7, "fg")] : (int · str) list

Functions

func : input type → output type

square 3.0 ⟹ 9.0

square : real ⟶ real

list sum [1, 2, 3] ⟹ 1 + 2 + 3 = 6

list sum (int list) ⟶ int

≡ int list ⟶ int

add1 3 ⟹ 4

add 1 : int → int

fun add1 x = x + 1;

ML infers that add1 int → int

add1hd [1, 2, 3] ⇒ [2, 2, 3]

fun add1hd L = (1 + hd L) :: (tl L)

ML infers add1hd: int list → int list

All ML functions are Unary, have 1 argument

fun hypot (x, y) = sqrt (x · x + y · y);

Recursion

```
fun fact(n)= if n = 0
                then 1
                else n · fact(n - 1);
```

fact int → int

```
fun fib n
    = if n =0
        then 0
        else if n=1
            then 1
            else fib(n - 1)+fib (n - 2);
```

```
fun list Sum L
    = if (null L)
```

```
                  then 0
                  else (hd L)+ listSum (tl L);

fun   listSum []=0
 |    list sum L= (hd L) + listSum (tl L);

fun   listSum  [ ]=0
   |    listSum  (h:t) = h + listSum(t);
```