

Lect 3: [Recursion]

+

[Induction]

A natural number is

either

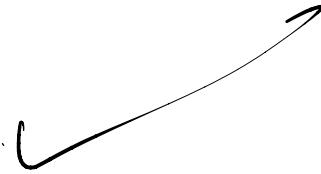
- 0, or

-  $1 + n$ , where  $n$  is  
a nat

→ and that's it!

0

is a nat



1

is a nat  $1 = 1 + \underline{0}$

2  
=

is a nat  $2 = 1 + 1$

c  
l  
c  
r  
r

(\* Purpose: double the input \*)  
n  
natural number n  
comment start

(\* Examples: double(2) = 4  
double(3) = 6 \*)  
comment end

fun double(n : int) : int =

~~2 \* n~~

fun double (n: int): int =

case n of

$$0 \Rightarrow \boxed{0}$$

$$1 - \Rightarrow \boxed{2 + \text{double}(n-1)}$$

recursive  
call

double(2)

$\mapsto$  case 2 of  $0 \Rightarrow 0 \} - \Rightarrow 2 + \text{double}(2-1)$

$\mapsto 2 + \text{double}(2-1)$

$\mapsto 2 + \text{double}(1)$

$\mapsto 2 + \text{case } 1 \text{ of } 0 \Rightarrow 0 \} - \Rightarrow 2 + \text{double}(1-1)$

$\mapsto 2 + (2 + \text{double}(1-1))$

2

$\mapsto 2 + (2 + \text{double } 0)$

$\mapsto 2 + (2 + \text{case } 0 \text{ of } 0 \Rightarrow 0 \} - \Rightarrow 2 + \text{double}(1-0))$

$\mapsto 2 + (2 + 0)$

$\mapsto 2 + 2 \mapsto 4$

To skip a case

- ① step the thing being cased on [until value]
- ② If its value is 0,  
then go step 1<sup>st</sup> branch
- ③ If its value is not 0,  
then go step 2<sup>nd</sup> branch

int in SML is an imperfect representation of natural numbers

- ① int is fixed size  $2^{64}$   
[maybe  $2^{63}$ ]
- ② int includes negatives ints

$\sim 1 \quad \sim 2 \quad \dots$   
 $\sim$

double (~1)

$\mapsto^* 2 + \text{double } (\sim 2)$

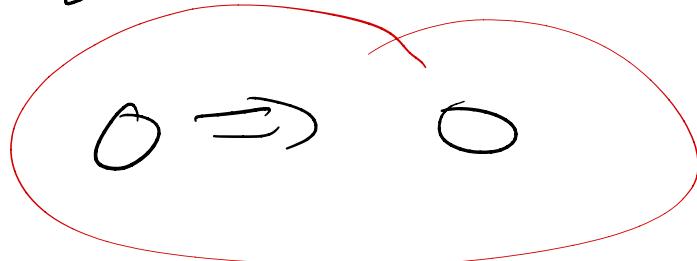
$\mapsto^* 2 + 2 + \text{double } (\sim 3)$

$\mapsto^*$   
;  
;  
)  
(  
(

infinite loop!

fun double (n:int):int =  
    

case n of



base case,

1 ->

$2 + \text{double}(n-1)$

recursive  
call

Recursive  
Input

is  
Smaller

double(0)

$\mapsto 2 + \text{double}(0-1)$

$\mapsto 2 + \text{double}(-1) \rightarrow 2 + (2 + \text{double}(-2))$

double(2)

→ 2 + double(z)

→ 2 + 2 + double(2)

→ 2 + 2 + 2 + double(z)

1

1

i

Infinite loop

Case  $n$  of

$0 \Rightarrow [e_0]$  has type

$1 \Rightarrow [e_1] T = \text{int}$

If  $n$  has type int

and  $e_0$  has type  ~~$T$~~   $= \text{int}$

$e_1$  has type  $T = \text{int}$

fun double(a:int):int =

{ double has type int → int

double(y) has type int

if y has type int

# Factorial

---

$$n!_o = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

$$5!_o = 5 * 4 * 3 * 2 * 1 = 120$$

(\* Purpose: compute  $n!$ )

$$n * (n - 1) * (n - 2) * (n - 3) * \dots$$

Example:  $\text{Factorial}(\underline{\underline{n}}) = 120^*$

fun fact (n : int) : int =

{case 0 of

$$0 \Rightarrow \underline{\underline{1}}$$

$$1 - \Rightarrow \underline{\underline{n * \text{fact}(n - 1)}}$$

fact(5)

→ case 5 of 0 => 1 | - => 5 \* fact(5-1)

→ 5 \* fact(5-1)

→ 5 \* fact(4)

→ 5 \* 4 \* fact(3)

→ 5 \* 4 \* 3 \* fact(2)

→ 5 \* 4 \* 3 \* 2 \* fact(1)

→ 5 \* 4 \* 3 \* 2 \* 1 \* fact(0)

→ 5 \* 4 \* 3 \* 2 \* 1 \* 1

→ 120

(\* Purpose: Say "aaaaa ... a"  
with  $\underbrace{n}_{\text{nat}}$  a's \*)

(\* E.g. doctor(4) = "aaaa"

doctor(5) = "aaaaa"

:  
:  
\*)

fun doctor(n: int) : string =

case n of

0 => ""

1 -> "a" ^ doctor(n-1)

doctor(2)

→ Case 2 of  $O \Rightarrow O$   $\rightarrow$  "a"  $\wedge$  doctor  
(2+1)

→ "a"  $\wedge$  doctor(2-1)

→ "a"  $\wedge$  doctor(1)

→ "a"  $\wedge$  "a"  $\wedge$  doctor(0)

→ "a"  $\wedge$  "a"  $\wedge$  " "

→ "aa"

fun double(n:int):int =

case n of

0 => 0

base case

1 ->

$$2 + \text{double}(n-1)$$

fun fact(n:int):int =

case 0 of

0 => 1

1 -> 
$$n * \text{fact}(n-1)$$

fun doctor(n:int):string =

case n of

0 => ""

1 -> "a" ^ doctor(n-1)

# Template

fun f(n:int):T =

case n of

0 => [ ]

1 ->

use

n

and

f(n-1)

# Methodology

3

- ① Give name and names of inputs and types
- ② Purpose
- ③ Examples
- ④ Write body using templates
- ⑤ Run examples as tests ✓  
X

# Induction

To prove something for all  
natural numbers,

- prove it for 0
- prove it for  $l+k$ ,  
assuming it is true  
for  $k$

(\* Purpose:  $\text{exp}(n)$  computes  $2^n$

E.g.  $\text{exp}(2) = 2^2 = 4$

$\text{exp}(3) = 2^3 = 8 \dots *$

fun  $\text{exp}(n : \text{int}) : \text{int} =$

case  $n$  of

$0 \Rightarrow 1$

$1 - =$

$2 * \text{exp}(n - 1)$

$1 - = \Rightarrow$

Theorem for all natural numbers  
 $n,$

$$\exp(n) = 2^n$$

code      ↗      math value

## Programs

- ① return a value
- ② infinite loop
- ③ raise an exception

$c = c'$  means

- ① same value
- ② both infinite loop
- ③ both raise the same exception

① equivalence relation

a)  $e = e$

b)  $e_1 = e_2$  if  $e_2 = e_1$

c)  $e_1 = e_3$  if  $e_1 = e_2$  and  $e_2 = e_3$

② congruence: replace equals  
with equals in a  
bigger program

③  $e_1 = e_2$  if

$$e_1 \xrightarrow{\quad} e_2$$

for all  $n$ ,  $\exp(n) = 2^n$

① Case for 0:

To show:  $\exp(0) = 2^0$

$\exp(0)$

→ case 0 of 0  $\Rightarrow 1 \mid - - -$

→ 1

$= 2^0$  math def

Case for  $l+k$ :

To show:  $\exp(l+k) = 2^{l+k}$

Inductive hypothesis: assume

$\exp(k) = 2^k$

$$\exp(l+k)$$

$\mapsto$  case  $l+k$  of  $0 \Rightarrow 1 \mapsto 2 * \exp((l+k)-1)$

$$\mapsto 2 * \exp((l+k)-1)$$

$$\mapsto 2 * \exp(k)$$

$$= 2 * 2^k$$

by inductive hypothesis

$$= 2^{l+k}$$

math