Lect 3: $\boxed{\text{Recursion}}$

+

$\boxed{\text{Induction}}$

A natural number is

either

- 0, or

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

→ and that's it!

$\underline{0}$ is a nat ✓

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

$\underline{2}$ is a nat $2 = 1 + 1$

$\vdots$

(* Purpose: double the input natural number *)

comment start

comment end

(* Examples: double(2) = 4
double(3) = 6 *)

fun double(n: int): int = 2 * n

fun double(n:int):int =
case n of
0 => $\boxed{0}$
| _ => $\boxed{2 + \text{double}(n-1)}$

recursive call

$$\mathrm{double}(2)$$

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

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

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

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

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

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

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

$$\mapsto 2 + (2 + 0) \mapsto 2 + 2 \mapsto 4$$

To step a case

1. step the thing being cased on [until value]

2. If its value is 0, then go step 1st branch

3. If its value is <u>not</u> 0, then go step 2nd branch

int in SML is an imperfect representation of natural numbers

1. int is fixed size 2^{64}

[maybe 2^{63}] ints

2. int includes negatives

~1 ~2 . . . . .

$$\mathrm{double}(\sim 1)$$

$$\longmapsto^* 2 + \mathrm{double}(\sim 2)$$

$$\longmapsto^* 2 + 2 + \mathrm{double}(\sim 3)$$

$$\longmapsto^* \vdots$$

infinite loop!

```
fun double (n : int) : int =
  case n of
    0 => 0
  | 1 => 2 + double (n - 1)
```

base case

recursive call

recursive input is smaller

double(0)

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

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

double(2)

$\mapsto$ 2 + double(2)

$\mapsto$ 2 + 2 + double(2)

$\mapsto$ 2 + 2 + 2 + double(2)

$\vdots$

infinite loop

case n of

$0 \Rightarrow \boxed{e_0}$ has type $T_{=int}$

$| \_ \Rightarrow \boxed{e_1}$

if n has type <u>int</u>

and e_0 has type $T = int$

e_1 has type $T = int$

fun double(n: int): int =

double has type int → int

double(y) has type int

if y has type int

Factorial

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

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

(* Purpose: compute n!

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

Example: factorial(5) = 120 *)

```
fun fact (n : int) : int =
  case n of
    0 => 1
  | _ => n * fact(n-1)
```

fact(5)

$\mapsto$ case 5 of 0 => 1 | _ => 5 * fact(5-1)

$\mapsto$ 5 * fact(5-1)

$\mapsto$ 5 * fact(4)

$\mapsto$ 5 * 4 * fact(3)

$\mapsto$ 5 * 4 * 3 * fact(2)

$\mapsto$ 5 * 4 * 3 * 2 * fact(1)

$\mapsto$ 5 * 4 * 3 * 2 * 1 * fact(0)

$\mapsto$ 5 * 4 * 3 * 2 * 1 * 1

$\mapsto$ 120

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

(* E.g. doctor(4) = "aaaa"
doctor(5) = "aaaaa"

⋮

*)

```
fun doctor(n: int) : string =
  case n of
    0 => ""
  | _ => "a" ^ doctor(n-1)
```

doctor(2)

$\mapsto$ case 2 of $0 \Rightarrow 0 \mid \_ \Rightarrow$ "a" ^ doctor (2 − 1)

$\mapsto$ "a" ^ doctor (2 − 1)

$\mapsto$ "a" ^ doctor (1)

$\mapsto$ "a" ^ "a" ^ doctor (0)

$\mapsto$ "a" ^ "a" ^ ""

$\mapsto$ "aa"

```
fun double(n:int):int =
    case n of
        0 => 0          base case
      | _ => 2 + double(n-1)
```

```
fun fact(n:int):int =
    case o of
        0 => 1
      | _ => n * fact(n-1)
```

```
fun doctor(n:int):string =
    case n of
        0 => ""
      | _ => "a" ^ doctor(n-1)
```

Template

```
fun f(n:int): T =
    case n of
        0 => 
      | _ => 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 <u>all</u> natural numbers,

- prove it for 0
- prove it for 1+k, <u>assuming</u> it is true for k

(* Purpose: exp(n) computes 2^n

E.g. $exp(2) = 2^2 = 4$

$exp(3) = 2^3 = 8$ ... *)

fun exp (n : int) : int =

case n of

0 => $\boxed{1}$

| _ => $\boxed{2 * exp(n-1)}$

| _ =>

Theorem for all natural numbers n,

$$\underbrace{\exp(n)}_{\text{code}} = \underbrace{2^n}_{\text{math (value)}}$$

Programs

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

$e = e'$ 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 $\boxed{e_1 \mapsto e_2}$

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

1 $\boxed{\text{Case for } 0:}$

<u>To show:</u> $exp(0) = 2^0$

$exp(0)$

$\mapsto$ case 0 of 0 => 1 | - - - .

$\mapsto$ 1

$= 2^0$ math def

$\boxed{\text{Case for } 1+k:}$

To show: $exp(1+k) = 2^{1+k}$

Inductive hypothesis: assume $exp(k) = 2^{k}$

$exp(1+k)$

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

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

$\mapsto 2 * exp(k)$

$= 2 * 2^{k}$ by inductive hypothesis

$= 2^{1+k}$ math