Recursion and Induction

driven by the structure of the data

A natural number is

either

- 0, or

- 1 + k, where k is a natural number

self-reference

0 is a nat ✓

1 is a nat

$1 = 1 + 0$

2 is a nat

$2 = 1 + 1$

$= 1 + 1 + 0$

Recursion

Methodology

1) Type + name
2) Purpose
3) Examples
4) Code
5) Test

Goal: double a natural number

$\boxed{=>}$

2 (* Purpose: *)

3 (* Examples:
double(2) should be 4
double(3) should be 6 *)

1

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

4

shift

recursive call

case n of

  0 => ________

| 1 => ________

| 2 => ________

| 3 => ________

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

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

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

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

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

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

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

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

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

$$\mapsto 2 + 2$$

$$\mapsto 4$$

To step

case n of

$0 \Rightarrow \boxed{e_0}$

$1 \_ \Rightarrow \boxed{e_1}$

1. step n until value
2. if value is 0,
   step to e_0
3. if value is not 0,
   step to e_1

(* Purpose
Compute 2^n
*)

(* Eg.
exp (2) = 4
exp (3) = 8
exp (4) = 16 . . .
*)

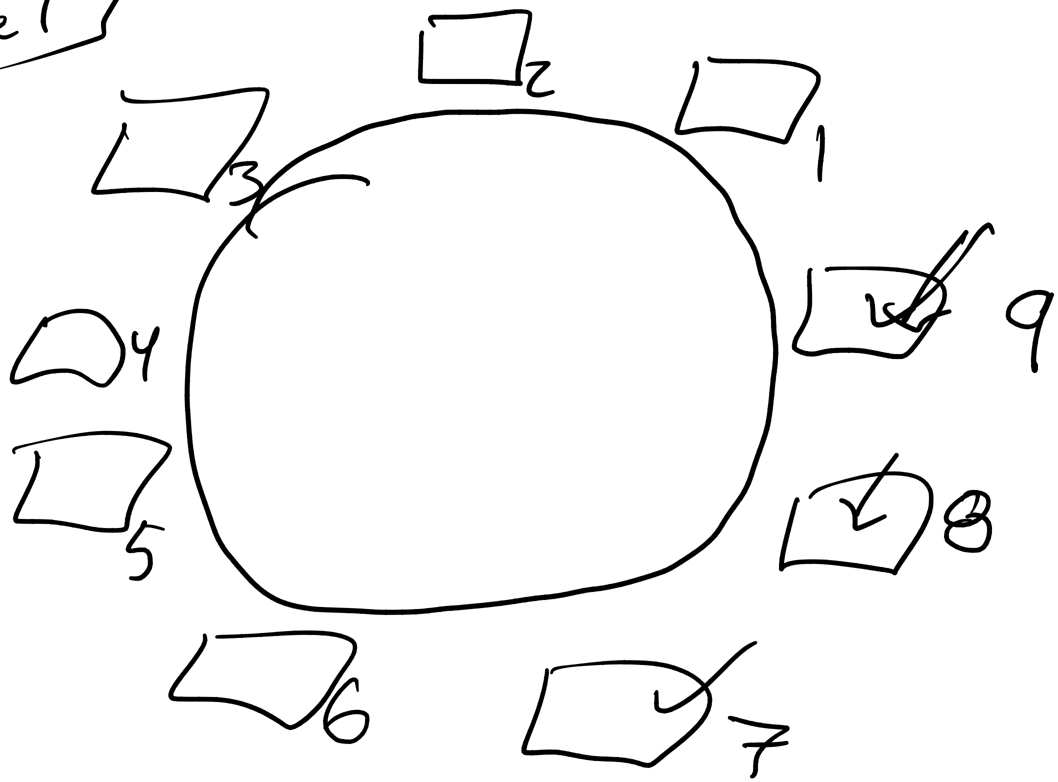
```
fun exp (n: int): int =
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

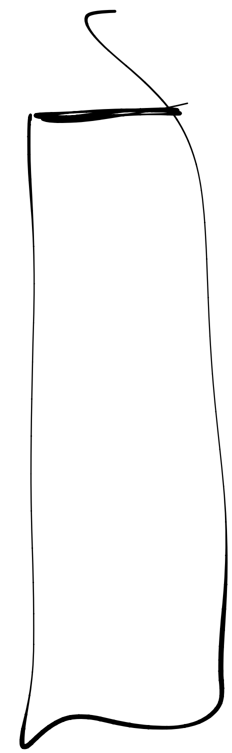
$$\overbrace{2 * 2 \times 2 \times 2 \times 2 \ldots\ldots \times 2}^{n \text{ times}}$$

Use Purpose + e.g.

$$9! = 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$$

9 factorial





Q: how many arrangements of 9 people?

*)

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

```
fact(5)
↦ case 5 of 0 => 1 | _ => 5 * fact(4)
↦ 5 * fact(4)
↦ 5 * (4 * 3 * 2 * 1)
```

$fact(5)$

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

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

$\mapsto$ 5 * fact(4)

$\mapsto$ 5 * case 4 of 0 => 1 _ => 4 * fact(4-1)

Problem: make the doctor say "aaaa.....a" with n letters

Examples

doctor 2 = "aa"

doctor 4 = "aaaa"

doctor 5 = "aaaaa"

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

doctor(5) should be "aaaaa"

doctor(4) should be "aaaa"

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

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

fun f(x : T_1) : T =

: T [: int, : string] [
case n : int of
  0 => $\boxed{e_0}$: int : string : T
| _ => $\boxed{e_1 \quad f(\_) : T}$: int : string : T
]

To prove something about

all natural numbers,

you can

1 prove it for 0

2 prove it for $1+k$

assuming it is true

for k

```
fun exp(n:int):int =
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

(* Purpose: for all natural numbers n,

$$\exp(n) = \underbrace{2^n}_{\text{math}} \quad *)$$

Proof by induction on n

1. Prove $\exp(0) = 2^0$

2. Prove $\exp(1+k) = 2^{1+k}$ assuming $\exp(k) = 2^k$] for any k

```
fun exp(n: int): int =
    case n of
      0 => 1
    | _ => 2 * exp(n-1)
```

1 **Base case**

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

$exp(0)$

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

$\mapsto 1$

$= 2^0$

(2) <u>To show:</u>

$$\exp(1+k) = 2^{1+k}$$

```
fun exp (n: int): int =
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

<u>Inductive hypothesis:</u>

$$\boxed{\exp(k) = 2^k}$$

$\exp(1+k)$

$\mapsto$ 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

$= 2^{1+k}$ by math

Programs aren't exactly math

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

"-1"

$$\mapsto 2 * \mathrm{exp}(-2)$$

$$\mapsto 2 * 2 * \mathrm{exp}(-3)$$

$$\mapsto 2 * 2 * 2 * \mathrm{exp}(-4)$$

$$\mapsto \vdots$$

Infinite loop

$e_1 = e_2$ $\quad$ e_1, e_2 programs

either

1. both e_1 and e_2 return the same value

2. both raise the same exception (5 div 0 raise Div)

3. both ∞-loop

Rules:

1. If $e \mapsto e'$ then $e = e'$

2. equivalence relation

   1. $e = e$
   2. if $e_1 = e_2$ then $e_2 = e_1$
   3. if $e_1 = e_2$ and $e_2 = e_3$ then $e_1 = e_3$

3. replace equals with equals inside an expression