Schedule

SCHEDULE

w	date	topic
~~7~~	~~November 25~~	~~introduction~~
~~8~~	~~December 2~~	~~higher-order functions, lists, trees~~
~~9~~	~~December 9~~	~~graphs, combinatorics~~
10	December 16	program reasoning
11	January 13	λ and interpreter
12	January 20	type system
13	January 27	exam part 2

CONTENTS

1. quiz
2. tail recursion
3. induction proofs
4. n-queens problem

Quiz

Q1

use recursion to implement

- fold_left
- fold_right

$$\text{List.fold\_left } (\circ)\ e\ [x_1; \cdots; x_n] = (e \circ x_1) \circ \cdots \circ x_n$$
$$\text{List.fold\_right } (\circ)\ [x_1; \cdots; x_n]\ e = x_1 \circ \cdots \circ (x_n \circ e)$$

Q2

use recursion to implement

- for_all
- exists

$$\text{List.for\_all } p\ [x_1; \cdots; x_n] = p\ x_1\ \&\&\ \cdots\ \&\&\ p\ x_n$$
$$\text{List.exists } p\ [x_1; \cdots; x_n] = p\ x_1\ ||\ \cdots\ ||\ p\ x_n$$

Q3

use fold_left or fold_right to implement

- for_all
- exists

$$\text{List.for\_all}\ p\ [x_1; \cdots ; x_n] \ = \ p\ x_1\ \&\&\ \cdots\ \&\&\ p\ x_n$$
$$\text{List.exists}\ p\ [x_1; \cdots ; x_n] \ = \ p\ x_1\ ||\ \cdots\ ||\ p\ x_n$$

Tail Recursion

Stack Problem of Recursion

```
# let rec sum n =
    if n = 0 then 0 else n + sum (n - 1)

# sum 100000;;
Stack overflow during evaluation (looping recursion?).
```

recursion may cause stack overflow. why?

$$
\begin{array}{rll}
 & \text{stack} & \text{register} \\
 & & \text{sum } 3 \\
= & 3+ & \text{sum } 2 \\
= & 3+(2+ & \text{sum } 3) \\
= & 3+(2+(1+ & \text{sum } 0)) \\
= & \cdots & \\
= & & 6
\end{array}
$$

Tail Recursion

- $\boxed{\text{tail call}}$ is outermost function call in expression
- tail recursion consumes no stack

```
let rec sum n =
  if n = 0 then 0 else n + sum (n - 1)
```

$\Downarrow$ tail recursive version of sum

```
let rec sum_aux m n =
  if n = 0 then m else sum_aux (m + n) (n - 1)
let sum' n = sum_aux 0 n
```

$$
\begin{array}{lll}
 & \text{register} & \\
\text{sum} & 3 & \\
= \text{sum\_aux} & 0 & 3 \\
= \text{sum\_aux} & 3 & 2 \\
= \text{sum\_aux} & 5 & 1 \\
= \text{sum\_aux} & 6 & 0 \\
= 6 & &
\end{array}
$$

Naive Version of Reversing

$$
\begin{aligned}
(@)\ []\ ys &= ys \\
(@)\ (x :: xs)\ ys &= x :: (xs@ys) \\
\mathsf{rev}\ [] &= [] \\
\mathsf{rev}\ (x :: xs) &= \mathsf{rev}\ xs@[x]
\end{aligned}
$$

$$
\begin{aligned}
& \mathsf{rev}\ [1;2;3] \\
= & \mathsf{rev}\ [2;3]\ @\ [1] \\
= & (\mathsf{rev}\ [3]\ @\ [2])\ @\ [1] \\
= & ((\mathsf{rev}\ [\,]\ @\ [3])\ @\ [2])\ @\ [1] \\
= & (([\,]\ @\ [3])\ @\ [2])\ @\ [1] \\
= & ([3]\ @\ [2])\ @\ [1]
\end{aligned}
$$

$$
\begin{aligned}
= & (3 :: ([]\ @\ [2]))\ @\ [1] \\
= & [3;2]@\ [1] \\
= & 3 :: ([2]@\ [1]) \\
= & 3 :: 2 :: ([]@\ [1]) \\
= & 3 :: 2 :: [1]
\end{aligned}
$$

Tail-recursive Version of Reversing

$$
\begin{aligned}
\mathsf{rev\_append}\ []\ \mathit{list} &= \mathit{list} \\
\mathsf{rev\_append}\ (x :: xs)\ \mathit{list} &= \mathsf{rev\_append}\ xs\ (x :: \mathit{list}) \\
\mathsf{rev}\ \mathit{list} &= \mathsf{rev\_append}\ \mathit{list}\ []
\end{aligned}
$$

$$
\begin{aligned}
& \mathsf{rev}\ [1;2;3] \\
= & \mathsf{rev\_append}\ [1;2;3]\ \ [] \\
= & \mathsf{rev\_append}\ [2;3]\ \ [1] \\
= & \mathsf{rev\_append}\ [3]\ \ [2;1] \\
= & \mathsf{rev\_append}\ []\ \ [3;2;1] \\
= & [3;2;1]
\end{aligned}
$$

Induction and Recursion

Induction on lists

THEOREM

$$\text{length } (xs@ys) = \text{length } xs + \text{length } ys$$

PROOF by induction on xs

- base case $xs = [\,]$

$$\begin{aligned} \text{length } ([\,]@ys) &= \text{length } ys && \text{def of } @ \\ &= \text{length } [\,] + \text{length } ys && \text{def of length} \end{aligned}$$

- inductive step $xs = x :: xs'$

$$\begin{aligned} \text{length } ((x :: xs')@ys) &= \text{length } (x :: (xs'@ys)) && \text{def of } @ \\ &= 1 + \text{length } (xs'@ys) && \text{def of length} \\ &= 1 + \text{length } xs' + \text{length } ys && \text{I.H.} \\ &= \text{length } (x :: xs') + \text{length } ys && \text{def of length} \end{aligned}$$

Mirroring Property: List

THEOREM

$$\text{rev}\ (\text{rev}\ l) = l$$

PROOF by induction on l

- base case $l = [\,]$

$$\begin{aligned} \text{rev}\ (\text{rev}\ [\,]) &= \text{rev}\ [\,] && \text{def of rev} \\ &= [\,] && \text{def of rev} \end{aligned}$$

- inductive step $l = x :: xs$

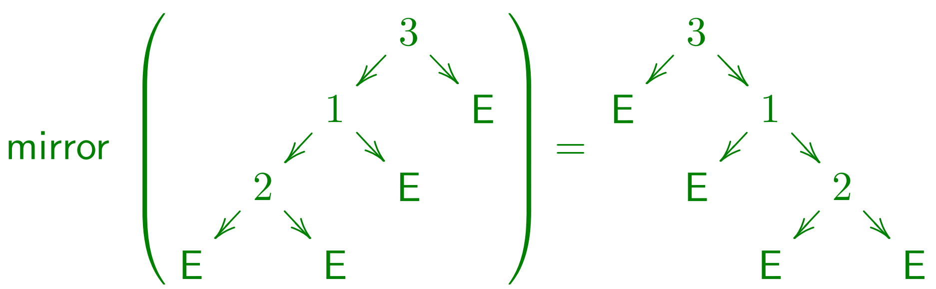
$$\begin{aligned} \text{rev}\ (\text{rev}\ (x :: xs)) &= \text{rev}\ (\text{rev}\ xs\ @\ [x]) && \text{def of rev} \\ &= x :: \text{rev}\ (\text{rev}\ xs) && \text{lemma} \\ &= x :: xs && \text{I.H.} \end{aligned}$$

LEMMA $\text{rev}\ (ys@[x]) = x :: \text{rev}\ ys$

Mirroring

```
type 'a tree = Empty | Node of 'a tree * 'a * 'a tree

let rec mirror = function
  | Empty -> Empty
  | Node (l, x, r) -> Node (mirror r, x, mirror l)
```



Mirroring Property: Trees

THEOREM

$$\text{mirror (mirror } t) = t$$

PROOF by induction on t

- base case $t = \text{Empty}$

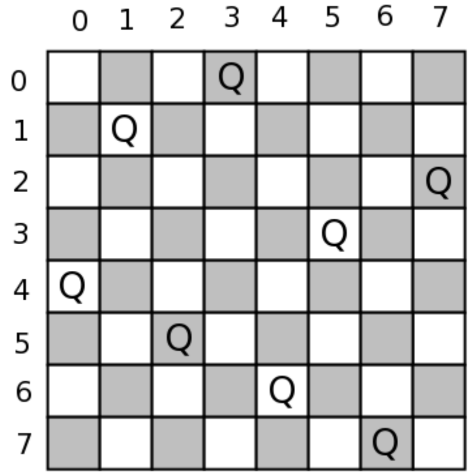
$$\begin{aligned} \text{mirror (mirror Empty)} &= \text{mirror Empty} && \text{def of mirror} \\ &= \text{Empty} && \text{def of mirror} \end{aligned}$$

- inductive step $t = \text{Node } (l, x, r)$

$$\begin{aligned} & \text{mirror (mirror (Node } (l, x, r))) \\ = {} & \text{mirror (Node (mirror } r, x, \text{mirror } l)) && \text{def of mirror} \\ = {} & \text{Node (mirror (mirror } l), x, \text{mirror (mirror } r)) && \text{I.H. (twice)} \\ = {} & \text{Node } (l, x, r) \end{aligned}$$

N-Queens Problem

N-Queens Problem: Generate and Test




```
let safe ((x1, y1) as q1) ((x2, y2) as q2) =
  (x1 <> x2 && x1 + y1 <> x2 + y2 &&
   y1 <> y2 && x1 - y1 <> x2 - y2   ) ||
  q1 = q2
```

```
let ok qs = List.for_all (fun q1 -> (List.for_all (safe q1) qs)) qs
```


N-Queens Problem: Generate and Test

```
# permutation (range 0 7);;
- : int list list =
[[0; 1; 2; 3; 4; 5; 6; 7]; [1; 0; 2; 3; 4; 5; 6; 7];
 [1; 2; 0; 3; 4; 5; 6; 7]; [1; 2; 3; 0; 4; 5; 6; 7]; .. ]
# List.combine (range 0 7) [4;1;5;0;6;3;7;2];;
- : (int * int) list =
[(0, 4); (1, 1); (2, 5); (3, 0); (4, 6); (5, 3); (6, 7); (7, 2)]
```

```
let solve n =
  let l = range 0 (n - 1) in
  List.find ok (List.map (List.combine l) (permutation l))
```