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. lists, lists of lists
2. graphs, adjacency lists, fixpoint

Products of Lists

Products of Lists

$$\text{product } f\ [x_1; \cdots; x_m]\ [y_1; \cdots; y_n] = [f\ x_1\ y_1; \cdots; f\ x_m\ y_n]$$

GOAL

```
# pi [[1;2];[3;4];[5;6]];;
- : int list list =
[[1; 3; 5]; [1; 3; 6]; [1; 4; 5]; [1; 4; 6];
 [2; 3; 5]; [2; 3; 6]; [2; 4; 5]; [2; 4; 6]]
```

IDEA

$$\text{pi } [X_1; \ldots; X_n] = X_1 \otimes \cdots \otimes X_n \otimes e = \text{List.fold\_right } (\otimes)\ [X_1; \ldots; X_n]\ e$$

CODE

```
let cons x xs = x :: xs
let pi lists = List.fold_right (product cons) lists [[]]
```


Permutations

Permutations

GOAL

```
# permutation [1;2;3];;
- : int list list =
[[1; 2; 3]; [2; 1; 3]; [2; 3; 1];
 [1; 3; 2]; [3; 1; 2]; [3; 2; 1]]
```

HOW TO IMPLEMENT

1. `interleave`
2. `permutation`

Interleave

DEFINITION

$$\text{List.map2}\ f\ [x_1; \ldots; x_n]\ [y_1; \ldots; y_n] = [f\ x_1\ y_1; \ldots; f\ x_n\ y_n]$$

EXERCISES

```
# prefix [2;3];;
- : int list list = [[]; [2]; [2; 3]]
# suffix [2;3];;
- : int list list = [[2; 3]; [3]; []]
# interleave 1 [2;3];;
- : int list list = [[1; 2; 3]; [2; 1; 3]; [2; 3; 1]]
```


Definition of Permutations

DEFINITION using list comprehension

$$\text{permutation}\ [\,] = [\ [\,]\ ]$$
$$\text{permutation}\ (x :: xs) = [zs \mid ys \leftarrow \text{permutation}\ xs; zs \leftarrow \text{interleave}\ x\ ys]$$

EXAMPLE

$$\text{permutation}\ [2;3] = [[2;3];[3;2]]$$
$$\begin{aligned}\text{permutation}\ [1;2;3] &= [zs \mid ys \leftarrow [[2;3];[3;2]]; zs \leftarrow \text{interleave}\ 1\ ys] \\ &= \text{interleave}\ 1\ [2;3]\ @\ \text{interleave}\ 1\ [3;2] \\ &= [[1;2;3];[2;1;3];[2;3;1];[1;3;2];[3;1;2];[3;2;1]]\end{aligned}$$

how to implement list comprehension?

Transformation Rules for List Comprehension

EXAMPLE

$$[\ x+1 \mid x \leftarrow [1;2;3]\ ] = \text{map}\ (\text{fun}\ x \rightarrow x+1)\ [1;2;3]$$
$$[\ x \mid x \leftarrow [1;2;3];\ \text{odd}\ x\ ] = \text{filter odd}\ [1;2;3]$$
$$[\ (x,y) \mid x \leftarrow [1;2]; y \leftarrow [\text{"}a\text{"};\text{"}b\text{"}]\ ] =$$
$$\text{concat}\ [\ [\ (x,y) \mid y \leftarrow [\text{"}a\text{"};\text{"}b\text{"}]\ ] \mid x \leftarrow [1;2]\ ]$$

RULES

$$[\ e \mid x \leftarrow xs\ ] = \text{map}\ (\text{fun}\ x \rightarrow e)\ xs$$
$$[\ e \mid x \leftarrow xs; p; \cdots\ ] = [\ e \mid x \leftarrow \text{filter}\ (\text{fun}\ x \rightarrow p)\ xs; \cdots]$$
$$[\ e \mid x \leftarrow xs; y \leftarrow ys; \cdots\ ] = \text{concat}\ [\ [\ e \mid y \leftarrow ys; \cdots\ ] \mid x \leftarrow xs\ ]$$

Translating List Comprehension: `permutation`

$$
\begin{aligned}
& \text{permutation } (x :: xs) \\
= & \ [zs \mid ys \leftarrow \text{permutation } xs; zs \leftarrow \text{interleave } x \ ys] \\
= & \ \text{concat } [\ [zs \mid zs \leftarrow \text{interleave } x \ ys\ ] \mid ys \leftarrow \text{permutation } xs\ ] \\
= & \ \text{concat } [\ \text{map (fun } zs \rightarrow zs) \ (\text{interleave } x \ ys) \mid ys \leftarrow \text{permutation } xs\ ] \\
= & \ \text{concat } [\ \text{interleave } x \ ys \mid ys \leftarrow \text{permutation } xs\ ] \\
= & \ \text{concat (map (fun } ys \rightarrow \text{interleave } x \ ys) \ (\text{permutation } xs)) \\
= & \ \text{concat (map (interleave } x) \ (\text{permutation } xs))
\end{aligned}
$$

Homework: Slowsort

Slowsort

- `ordered`

```
# ordered [1;2;3]
- : bool = true
# ordered [3;2;1]
- : bool = false
```

- `slowsort`

$$
\begin{aligned}
& \text{slowsort } [2;3;1] \\
= & \ \text{List.find ordered (permutation } [1;2;3]) \\
= & \ \text{List.find ordered } [[2;3;1];[3;2;1];[3;1;2];[2;1;3];[1;2;3];[1;3;2]] \\
= & \ [1;2;3]
\end{aligned}
$$

Applications of Combinatorial Functions

- n-queens
- slowsort
- travel salesman problem (TSP)
- satisfiability problem (SAT)
- (in)equation solving
- ...

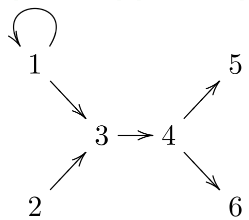
- trivial implementation
- very inefficient: $n!$ elements for $[x_1; \ldots; x_n]$

Homework: Graphs

Graphs and Adjacency Lists

```
type 'a graph = ('a * 'a list) list
```

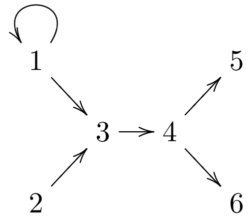
$$g = [(1,[1;3]);\quad (2,[3]);\quad (3,[4]);\quad (4,[5;6]);\quad (5,[]);\quad (6,[])]$$



GOAL

```
# succ g 3;;
- : int list = [4]
# reachable_from g 3;;
- : int list = [3; 4; 5; 6]
# pred g 3;;
- : int list = [1;2]
# reachable_to g 3;;
- : int list = [1; 2; 3]
```


Idea: Fixpoint



define

$$\mathsf{f}\ X\ = X \cup \{\ y \mid x \in X, y \in \mathsf{succ}\ g\ x\ \}$$

$$\begin{aligned} \mathsf{f}\ \{3\} &= \{3,4\} \\ \mathsf{f}\ \{3,4\} &= \{3,4,5,6\} \\ \mathsf{f}\ \{3,4,5,6\} &= \{3,4,5,6\} \quad \text{fixpoint} \end{aligned}$$

Implementation

- set operators

```
# union [1;2;3] [2;3;4]
- : int list = [1; 2; 3; 4]
# equal [1;2;3] [2;3;1]
- : bool = true
```

- `fixpoint`

$$\text{fixpoint}\ f\ X = \begin{cases} X & \text{if } f\ X = X \\ \text{fixpoint}\ f\ (f\ X) & \text{otherwise} \end{cases}$$

- `reachable_from`

```
let reachable_from g x =
  let f xs = ... in
  fixpoint f [x]
```

Translating List Comprehension: `pred`

$$
\begin{aligned}
& \mathsf{pred}\ g\ x \\
=\ & [\ src \mid (src, dsts) \leftarrow g; \mathsf{mem}\ x\ dsts\ ] \\
=\ & [\ src \mid (src, dsts) \leftarrow \mathsf{filter}\ (\mathsf{fun}\ (src, dsts) \rightarrow \mathsf{mem}\ x\ dsts)\ g\ ] \\
=\ & \mathsf{map}\ (\mathsf{fun}\ (src, dsts) \rightarrow src)\ (\mathsf{filter}\ (\mathsf{fun}\ (src, dsts) \rightarrow \mathsf{mem}\ x\ dsts)\ g) \\
=\ & \mathsf{map}\ \mathsf{fst}\ (\mathsf{filter}\ (\mathsf{fun}\ (src, dsts) \rightarrow \mathsf{mem}\ x\ dsts)\ g)
\end{aligned}
$$