

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

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

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

$$\text{pi } [X_1; \dots; X_n] = X_1 \otimes \dots \otimes X_n \otimes e = \text{List.fold_right } (\otimes) [X_1; \dots; X_n] e$$

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

5

Interleave

DEFINITION

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

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]]
```

6

Definition of Permutations

DEFINITION

using **list comprehension**

$\text{permutation } [] = [[]]$

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

EXAMPLE

$\text{permutation } [2; 3] = [[2; 3]; [3; 2]]$

$\text{permutation } [1; 2; 3] = [zs \mid ys \leftarrow [[2; 3]; [3; 2]]; zs \leftarrow \text{interleave } 1 \text{ } ys]$

$= \text{interleave } 1 [2; 3] \text{ @ } \text{interleave } 1 [3; 2]$

$= [[1; 2; 3]; [2; 1; 3]; [2; 3; 1]; [1; 3; 2]; [3; 1; 2]; [3; 2; 1]]$

how to implement list comprehension?

7

Transformation Rules for **List Comprehension**

EXAMPLE

$[x + 1 \mid x \leftarrow [1; 2; 3]] = \text{map } (\text{fun } x \rightarrow x + 1) [1; 2; 3]$

$[x \mid x \leftarrow [1; 2; 3]; \text{odd } x] = \text{filter } \text{odd } [1; 2; 3]$

$[(x, y) \mid x \leftarrow [1; 2]; y \leftarrow [\"a\"; \"b\"]] =$

$\text{concat } [[(x, y) \mid y \leftarrow [\"a\"; \"b\"]] \mid x \leftarrow [1; 2]]$

RULES

$[e \mid x \leftarrow xs] = \text{map } (\text{fun } x \rightarrow e) xs$

$[e \mid x \leftarrow xs; p; \dots] = [e \mid x \leftarrow \text{filter } (\text{fun } x \rightarrow p) xs; \dots]$

$[e \mid x \leftarrow xs; y \leftarrow ys; \dots] = \text{concat } [[e \mid y \leftarrow ys; \dots] \mid x \leftarrow xs]$

8

Translating List Comprehension: `permutation`

```
permutation (x :: xs)
= [zs | ys ← permutation xs; zs ← interleave x ys]
= concat [ [zs | zs ← interleave x ys ]           | ys ← permutation xs ]
= concat [ map (fun zs → zs) (interleave x ys) | ys ← permutation xs ]
= concat [ interleave x ys                         | ys ← permutation xs ]
= concat (map (fun ys → interleave x ys) (permutation xs))
= concat (map (interleave x)                (permutation xs))
```

9

Homework: Slowsort

Slowsort

- ▶ `ordered`

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

- ▶ `slowsort`

```
slowsort [2; 3; 1]
= List.find ordered (permutation [1; 2; 3])
= List.find ordered [[2; 3; 1]; [3; 2; 1]; [3; 1; 2]; [2; 1; 3]; [1; 2; 3]; [1; 3; 2]]
= [1; 2; 3]
```

11

Applications of Combinatorial Functions

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

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

12

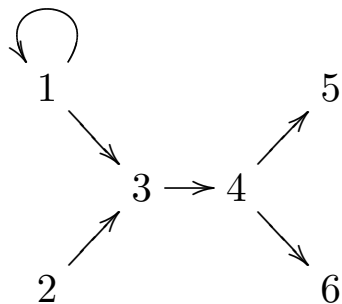
Homework: Graphs

13

Graphs and Adjacency Lists

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

```
g = [(1, [1; 3]); (2, [3]); (3, [4]); (4, [5; 6]); (5, []); (6, [])]
```

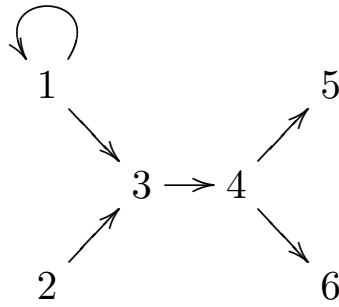


GOAL

```
# 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]
```

14

Idea: **Fixpoint**



define

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

$$f \{3\} = \{3, 4\}$$

$$f \{3, 4\} = \{3, 4, 5, 6\}$$

$$f \{3, 4, 5, 6\} = \{3, 4, 5, 6\} \text{ fixpoint}$$

15

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]
```

16

Translating List Comprehension: `pred`

```
pred g x
= [ src | (src, dsts) ← g; mem x dsts ]
= [ src | (src, dsts) ← filter (fun (src, dsts) → mem x dsts) g ]
= map (fun (src, dsts) → src) (filter (fun (src, dsts) → mem x dsts) g)
= map fst (filter (fun (src, dsts) → mem x dsts) g)
```