Recitation 24: Proofs about Programs (2 of 2)

Data structures

(** [push x s] is the stack [s] with
[x] pushed on top *)

(x₁; x₂; ...; xₙ)

↑ top

(** [push x (y₁; ...; yₙ)] is [(x; y₁; ...; yₙ)] *)

```
val push : 'a -> 'a t -> 'a t
```

peek(push x s) = x

Equational Specifications

empty	1	is_empty empty = true
is_empty	2	is_empty (push x s) = false
pop	3	pop (push x s) = s
peek	4	peek (push x s) = x
push		

peek (pop (push 3 (push 5 empty))) $\stackrel{(3)}{=}$

peek (push 5 empty) $\stackrel{4}{=}$

5

$(x_1; x_2; \dots; x_n)$

↑

push x_1 (push x_2 (... (push x_n empty) ...))

<u>canonical</u> form   <u>generators</u> push, empty

empty
push } generators
is_empty
peek } queries
pop } manipulators

Notice

equations are queries / manipulators acting on generators

1 is_empty empty = true
2 is_empty (push x s) = false
3 pop (push x s) = s
4 peek (push x s) = x

Proofs!

```
module ListStack = struct
  type 'a t = 'a list
  let empty = []
  let is_empty s = (s = [])
  let peek = List.hd
  let pop = List.tl
  let push = List.cons
end
```

Show: pop (push x s) = s

pop (push x s)

= { eval pop, push }

List.tl x :: s

= { eval tl }

s

Queues!

is_empty
empty

enq
front
deq

is_empty empty = true

is_empty (enq x q) = false

front (enq x q) =

x	if is_empty q
front q	if not

deq (enq x q) =

empty	if is_empty q
enq x (deq q)	if not

Simplify

deq (enc 3 (enc 4 (enc 5 empty)))
= enc 3 (deq (enc 4 (enc 5 empty)))
= enc 3 (enc 4 (dec (enc 5 empty)))
= enc 3 (enc 4 (empty)))

ListQueue : eval

```
module TwoListQueue = struct
                                          front        back
  (* AF : (f, b) represent the queue  f @ rev b
     RI: if f = [] then b = [] *)
  type 'a t = 'a list * 'a list
  let empty = [], []
  let is_empty (f, _) = f = []
  ...
end
```

Proofs use 2 additional techniques

RI(q) : if q = (f, b) and f = [] then b = []

if is_empty q then q = [], []

If AF(e) = AF(e') ⇒ e' = e

$([x_1; x_2; \dots x_n], [y])$

$([x_1; x_2; \dots x_n; y], [])$