Abstract Data Types

EECS 214

November 2, 2015
So you want a FIFO queue, do you?

What’s a FIFO?
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- $empty() : \text{FifoQ}$
- $empty?(\text{FifoQ}) : \mathbb{B}$
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- `empty()` : FifoQ
- `empty?(FifoQ) : ℂ`
- `enqueue(Element, FifoQ)`
- `dequeue(FifoQ) : Element`
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- `empty()` : FifoQ
- `empty?(FifoQ) : ℂĕ
- `enqueue(Element, FifoQ)
- `dequeue(FifoQ) : Element`

And it has some behavior, e.g.:

- `empty?(empty()) = ⊤`
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- \textit{empty}() : \texttt{FifoQ}
- \textit{empty}?(\texttt{FifoQ}) : \mathbb{B}
- \textit{enqueue} (\texttt{Element}, \texttt{FifoQ})
- \textit{dequeue} (\texttt{FifoQ}) : \texttt{Element}

And it has some behavior, e.g.:

- \textit{empty}?(\textit{empty}()) = \top
- \textit{enqueue}(e, q); \textit{empty}?(q) = \bot
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- \textit{empty}() : FifoQ
- \textit{empty?}(FifoQ) : \mathbb{B}
- \textit{enqueue}(\text{Element}, \text{FifoQ})
- \textit{dequeue}(\text{FifoQ}) : \text{Element}

And it has some behavior, e.g.:

- \textit{empty?}(\text{empty}()) = \top
- \textit{enqueue}(e,q); \textit{empty?}(q) = \bot
- q \leftarrow \textit{empty}()
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- \( empty() : \text{FifoQ} \)
- \( empty?(\text{FifoQ}) : \mathbb{B} \)
- \( enqueue(\text{Element, FifoQ}) \)
- \( dequeue(\text{FifoQ}) : \text{Element} \)

And it has some behavior, e.g.:

- \( empty?(empty()) = \top \)
- \( enqueue(e,q); empty?(q) = \bot \)
- \( q \leftarrow empty() \)
  \( enqueue(a,q); enqueue(b,q) \)
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- $empty() : \text{FifoQ}$
- $empty?\,(\text{FifoQ}) : \mathbb{B}$
- $enqueue(\text{Element}, \text{FifoQ})$
- $dequeue(\text{FifoQ}) : \text{Element}$

And it has some behavior, e.g.:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$
  $enqueue(a, q); enqueue(b, q)$
  $a' \leftarrow dequeue(q); b' \leftarrow dequeue(q)$
So you want a FIFO queue, do you?

What’s a FIFO? Well, it has some operations:

- $empty() : \text{FifoQ}$
- $empty?(\text{FifoQ}) : \mathbb{B}$
- $enqueue(\text{Element}, \text{FifoQ})$
- $dequeue(\text{FifoQ}) : \text{Element}$

And it has some behavior, e.g.:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$
  $enqueue(a, q); enqueue(b, q)$
  $a' \leftarrow dequeue(q); b' \leftarrow dequeue(q)$
  $a' = a \land b' = b \land empty?(q) = \top$
But what *is* it?

It doesn’t matter.
HOW CAN IT NOT MATTER?
Let’s use one and see

– adt.rkt –
ADTs can have multiple implementations

Like you saw on the exam! Two possible FIFO implementations:

- linked list
- ring buffer
Linked list FIFO

(define-struct list-fifo-cell [first rest])
(define-struct list-fifo [front back])

; A ListFifoList is one of:
;   - '()
;   - (make-list-fifo-cell Element ListFifoList)

; A ListFifo is
;   (make-list-fifo ListFifoList
;   (make-list-fifo-cell Element '(()))
;   where either
;   - both fields are '(), or
;   - the `back' is the last cell of `front'
Ring buffer FIFO

(define-struct ring-fifo [front back elements])

; A RingFifo is
; (make-ring-fifo N N [Vector-of Element])
;
; where `front' and `back' are valid indices
; for `elements', and one of:
; − front = back means it's empty
; − front < back means the FIFO comprises
;   elements [front, back)
; − front > back means the FIFO comprises
;   elements [front, size) then [0, back)