

An Algebraic Specification of a simple queue.
 (as thought up impromptu during the Tuesday 9th December lecture)
 (Following the axioms, I have supplied three separate proofs for your scrutiny)

Consider the following instance of a queue:

→ | 9 | 12 | 1 | →

Operations:

create (create a new queue)
 put (place an element at the back of the queue)
 get (remove an element from the front of the queue)
 next (check which element is at the front of the queue)
 count (count how many elements are in the queue)
 is_empty (check whether or not a queue contains any elements)

Signatures:

create () -> queue	[basic]
put (q, e) -> queue	[basic]
get (q) -> queue	[extra]
next (q) -> e	[inspector]
count (q) -> N	[inspector]
is_empty (q) -> Boolean	[inspector]

Axioms:

[basics x (extras + inspectors) => 2x(1+3) = 8 axioms]

- 1) get (create ()) = error
- 2) get (put (q,e)) = if count(q) = 0 then create () else put (get (q),e)
- 3) next (create ()) = error
- 4) next (put (q,e)) = if count(q) = 0 then e else next (q)
- 5) count (create ()) = 0
- 6) count (put (q,e)) = if count (q) = 0 then 1 else count (q) + 1
- 7) is_empty (create ()) = TRUE
- 8) is_empty (put (q,e)) = FALSE

Proofs:

(Proof 1) To prove that, from the original queue, removing the element at the front of the queue will result in the value 12 as the next element.

The initial queue:

put (put (put (create (),1),12),9)

Operations to be proven on the initial queue:

next (get (put (put (put (create (),1),12),9)))

Using axiom 2 else clause on the initial queue and operations results in:

next (put (get (put (put (create (),1),12),9)))

Using axiom 2 if clause on the above:

next (put (put (get (put (create (),1)),12),9))

Using axiom 2 if clause on the above:

next (put (put (create (),12),9))

Using axiom 4 else clause on the above:

next (put (create (),12))

Using axiom 4 if clause on the above:

12 (proven as required)

(Proof 2) To prove that, from the original queue, removing two consecutive elements from the front of the queue will result in a queue of length 1.

The initial queue:

put (put (put (create (),1),12),9)

Operations to be proven on the initial queue:

count (get (get (put (put (put (create (),1),12),9))))

Using axiom 2 else clause on the initial queue and operations results in:

count (get (put (get (put (put (create (),1),12),9)))

Using axiom 2 else clause on the above:

count (get (put (put (get (put (create (),1),12),9)))

Using axiom 2 if clause on the above:

count (get (put (put (create (),12),9)))

Using axiom 2 else clause on the above:

count (put (get (put (create (),12),9)))

* Using axiom 2 if clause on the above:

count (put (create (),9))

Using axiom 6 if clause on the above:

1 (proven as required)

* You could take an equally valid alternative approach at this point and onwards, as follows...

Using axiom 6 else clause on the above:

count (get (put (create (), 12))) + 1

Using axiom 2 if clause on the above:

count (create ()) + 1

Using axiom 5 on the above:

0 + 1 = 1 (proven as required)

(Proof 3) To prove that, from the original queue, removing one consecutive element from the front of the queue will not result in an empty queue.

The initial queue:

put (put (put (create (),1),12),9)

Operations to be proven on the initial queue:

is_empty (get (put (put (put (create (),1),12),9)))

Using axiom 2 else clause on the initial queue and operations results in:

is_empty (put (get (put (put (create (),1),12),9)))

Using axiom 2 else clause on the above:

is_empty (put (put (get (put (create (),1),12),9)))

Using axiom 2 if clause on the above:

is_empty (put (put (create (),12),9))

Using axiom 8 on the above:

FALSE (proven as required)