Herlihy and Wing Queue

Pooja Verma, 203050072

Indian Institute of Technology Bombay

February 16, 2021

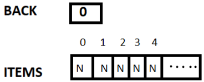
Algorithm

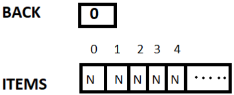
- Given a queue implementation containing
  - an integer, back
  - an array of values, items

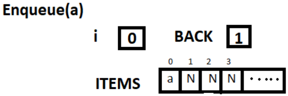
```
Enq = proc (q: queue, x: item)          // ignoring buffer overflow
 i: int = INC(q.back)      // allocate new slot, atomic
 STORE(q.items[i], x)  // fill it

Deq = proc (q: queue) returns (item)
 while true do
  range: int = READ(q.back) - 1
  for i: int in 1.. range do
    x: item = SWAP (q.items[i], null)
    if x!= null then return
```

Enque Operation



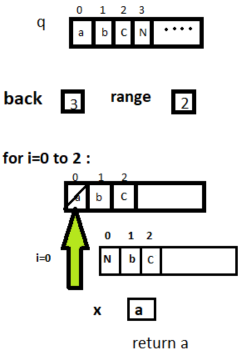


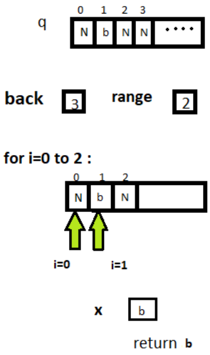


var $q.back : int \leftarrow 0$
var $q.items$: **array of** val
$\leftarrow \{\texttt{NULL}, \texttt{NULL}, \ldots\}$

procedure $\text{enq}(x : val)$
$\langle i \leftarrow \texttt{INC}(q.back) \rangle$ $\triangleright E_1$
$\langle q.items[i] \leftarrow x \rangle$ $\triangleright E_2$

Deque Operation





procedure deq() : *val*
  while true **do**
    $\langle range \leftarrow q.back - 1\rangle$
    for $i = 0$ **to** *range* **do**
      $\langle x \leftarrow \mathtt{SWAP}(q.items[i], \mathtt{NULL})\rangle$
      if $x \neq \mathtt{NULL}$ **then return** x

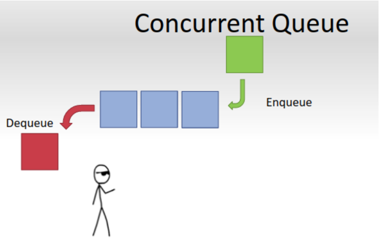
Is the algorithm correct ?

What does it mean for a concurrent algorithm to be correct ?

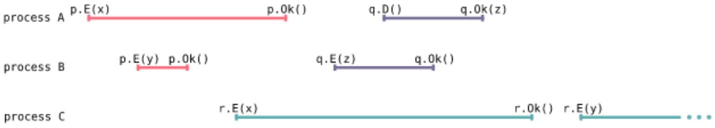
Linearizability :

Some definitions :

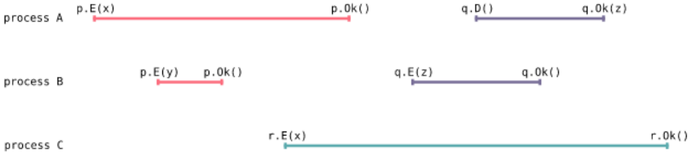
- Objects and methods :
- Method invocations are divided into two distinct parts:
  1. the invocation
     Example - q.Enqueue(x)
  2. the response
     Example - q.OK()
- Event -
  Example - A.p.E(x) or A.q.OK()



- history -
  A.p.E(x), B.p.D(), B.p.Ok(x)
- Sequential history -
  example- [A.p.E(x),A.p.Ok()] or [A.p.E(x).A.p.Ok(),A.p.E(y)]
- Pending invocation -



- Complete history -



Subsequence History

Figure 4. $H_1|A$

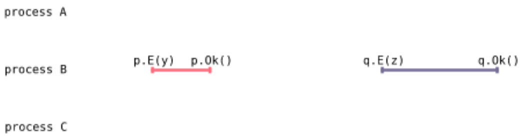


Figure 5. $H_1|B$

Equivalent Histories

Two histories H and H' are said to be **equivalent** if they satisfy the following property.

$$\forall P.\ H|P = H'|P$$

Linearizability-

- History H is linearizable if it can be extended to history G so that G is equivalent to legal sequential history S where ->G (subset sign)->S
- G is same as H without pending invocations.

Example

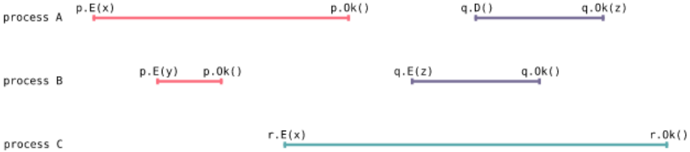


Figure: H

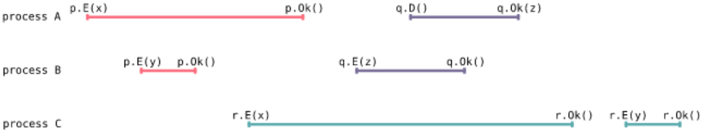


Figure: H'

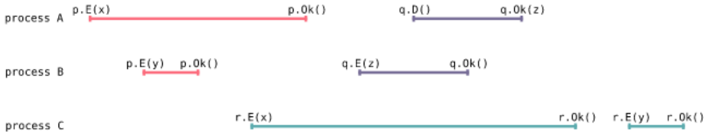


Figure: H'

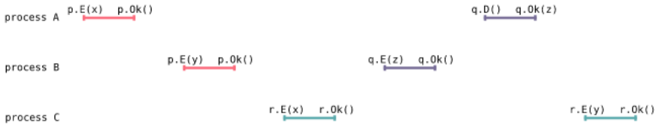


Figure: S

Identifying Linearization point is difficult

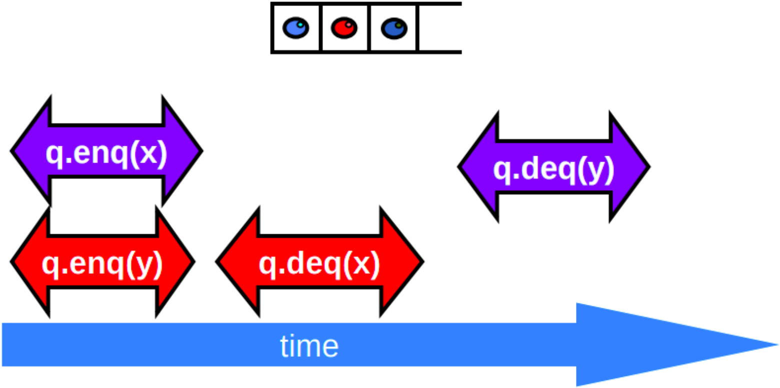
- it is difficult or even impossible to statically determine all linearization points.
- Linearization points differ depending on the execution history.
- Example ??

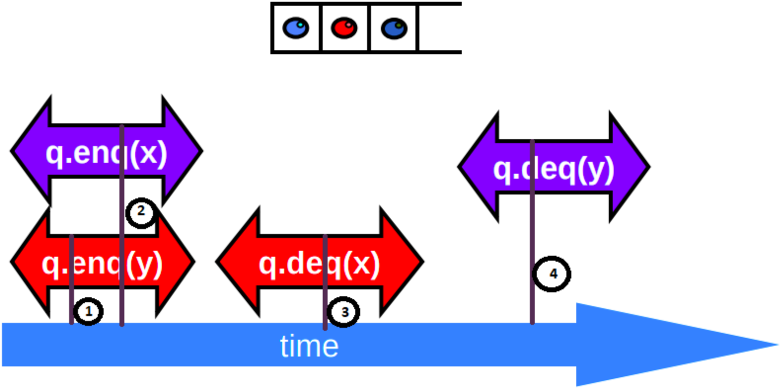
Key idea in Queue Proof

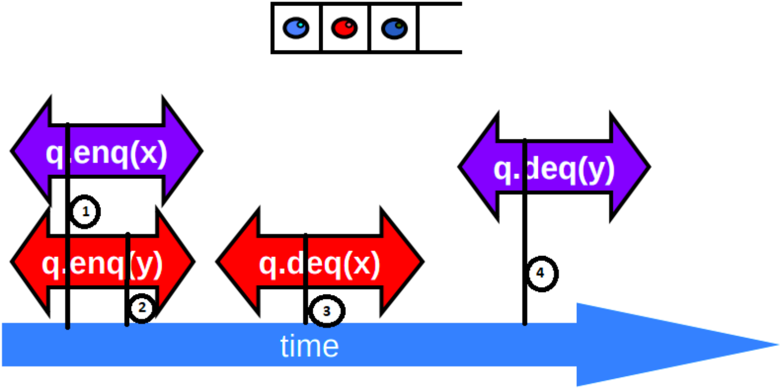
- Lemma: If Enq(x), Enq(y), Deq(x) and Deq(y) are complete operations of H such that x's Enq precedes y's Enq, then y's Deq does not precede x's Deq.

- Proof: Suppose this is not true. Pick a linearization and let q_i and q_j be queue values following the Deq operation of x and y respectively. From the assumption that $j < i$, $q_{j-1} = [y,\ldots,x,\ldots]$ which implies that y is enqueued before x, a contradiction.

Example




q.enq(x)
2
q.enq(y)
1
q.deq(x)
3
q.deq(y)
4
time



q.enq(x)
1
q.enq(y)
2
q.deq(x)
3
q.deq(y)
4
time

A correct (i.e., linearizable) concurrent queue:

- must not allow dequeuing an element that was never enqueued
- must not allow the same element to be dequeued twice
- must not allow elements to be dequeued out of order; and
- must correctly report whether the queue is empty or not

References

- https://link.springer.com/chapter/10.1007/978-3-642-31424-7_21
- https://mwhittaker.github.io/blog/visualizing_linearizability/
- https://www.coursera.org/lecture/concurrent-programming-in-java/4-3-linearizability-2ZxOt

Thankyou