Relaxation for Efficient Asynchronous Queues

Samuel Baldwin
Bucknell University, USA
Cole Hausman
Bucknell University, USA
Mohamed Bakr
Bucknell University, USA
Edward Talmage
Bucknell Univserity, USA

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 1 Code for each process p_i to implement a Queue with out-of-order k-relaxed *Dequeue*, where $k \geq n$ and $l = [k/n]$

```
function ENQUEUE(val)
    EnqCount = 0
    updateTS(v_i)
    enq_timestamp = v_i
    send (EnqReq, val, i, enq_timestamp) to all processes
end function
function RECEIVE(EnqReq, val, j, enq_timestamp) from p_j
    updateTS(v_i, v_j)
    if enq_timestamp not in Pending_Enqueues then
        Pending_Enqueues.insertByTS(enq_timestamp, val)
    end if
    send (EnqAck, i) to p_j
end function
function RECEIVE(EnqAck from p_j)
    EnqCount+ = 1
    if EnqCount == n then
        if localQueue.size < k then
            send (EnqConfirm, enq_timestamp) to all processes
        end if
    end if
    return EnqResponse
end function
function RECEIVE(EnqConfirm, enq_timestamp from p_j)
    localQueue.insertByTS(Pending_Enqueues.getByTS(enq_timestamp))
    if clean == true and localQueue.size() ≤ k then ▷ localQueues agree by this point
        let procNum = (localQueue.size() − 1  mod n
        localQueue.label(p_procNum, localQueue.tail)        ▷ I may have mangled this line
    end if
end function
```

Algorithm 2 Continued, part 2

```
function DEQUEUE
    v_i += 1
    let Deq_ts = v_i
    if localQueue.peekByLabel(p_i) ≠ ⊥ then          ▷ Check that I didn't change this
        let ret = localQueue.deqByLabel(p_i)
        send (Deq_f, ret, Deq_ts) to all processes
    else
        send (Deq_s, null, Deq_ts) to all processes
    end if
end function
function RECEIVE(deq_f, val, Deq_ts) from p_j)
    if j ≠ i then localQueue.remove(val)
    end if
end function
function RECEIVE(deq_s, val, Deq_ts from p_j)
    UpdateTs(v_i, Deq_ts)
    if Deq_ts is not in PendingDequeues then
        PendingDequeues.insertByTs(createList(Deq_ts, p_invoker))          ▷ Check line
    end if
    let p_invoker = p_j ▷ This doesn't make sense to me? What are you doing on this line?
    if Deq_ts ≠ 0 and Deq_ts < v_i then
        send (Unsafe, Deq_ts, i, p_invoker) to all processes
    else
        send (Safe, Deq_ts, i, p_invoker) to all processes
    end if
end function
```

Algorithm 3 Continued, part 3

```
function RECEIVE(Safe/Unsafe, Deq_ts, j, p_invoker)
    if Deq_ts not in PendingDequeues then
        PendingDequeues.insertByTs(createList(Deq_ts, p_invoker))
    end if
    for confirmationList in PendingDequeues do
        if confirmationList.ts == Deq_ts then
            if Unsafe then
                response = 2
            else
                response = 1
            end if
            confirmationList.list[j] = response
        end if
        propagateEarlierResponses(PendingDequeues)
    end for
    for (index, confirmationList) in PendingDequeues do
        if not confirmationList.contains(0) and not confirmationList.handled then
            pos = 0
            for response in confirmationList.list do
                if response == 2 then
                    pos+ = 1
                end if
            end for
            confirmationList.handled = True
            updateUnsafes(Lists, index)
            ret = localQueue.deqByIndex(pos)
            labelElements(p_invoker)
            if i == p_invoker then
                return ret
            end if
        end if                                  ▷ Not sure I left the nesting right on these.
    end for
end function
```