

System Modelling and Design

Modelling a Queue

Towards Implementation

Revision: March 28 2011

Ken Robinson

March 28, 2011

©Ken Robinson 2005-2010

[mailto::k.robinson@unsw.edu.au](mailto:k.robinson@unsw.edu.au)

This model will involve data refinement towards what could be called a *pointer implementation*.

We will specify a simple *Queue* machine that models a queue manager. A *queue*, of course, is a *first in first out* structure.

The items in the queue are represented by the set *ITEM* and it should be noted that we allow the same item to appear more than once in the queue. We are never concerned about the identity of the items, we are only concerned with the queue tokens that are taken from the set *QUEUE*. The queue tokens are unique.

0.1 Queue Events

The machine has the following events:

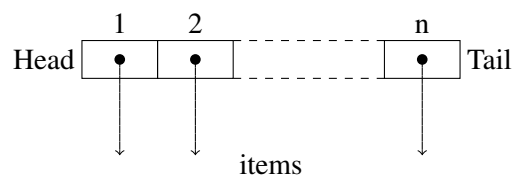
Enqueue(item) an event that places an item on the end of the queue. The event creates a unique queue identifier for this item. A unique item identifier is also generated for the item that is queued. A queue can contain multiple instances of the same item value.

Dequeue an event that removes the item that is at the head of the queue.

Unqueue(qid) removes the item from the queue identified by the queue identifier, *qid*. This is not a strict queue event; it is used to remove an item outside the queue discipline.

0.2 Modelling of queue

The queue is first modelled by a sequence with the head of the queue being the first element of the sequence; the end of the queue is the last element of the sequence.



Because a sequence is a monolithic structure the coherence of the queue structure is trivially guaranteed.

The Unqueue event requires unique identification of items in the queue. Since the position of an item in the queue changes as the queue changes, the initial position of an item in the queue cannot be used to uniquely identify the item. For that reason the elements of the queue will be unique identifiers, *queuetokens*. A function, *queueitem*, maps from *queuetokens* to the actual items.

0.3 Context Machines

Context Machines

Since EventB does not have a sequence type we need to define our own sequence type and accompanying functions for managing queues represented as sequences.

QueueContext contains *QUEUE*, which models the set of all injective queues of *TOKEN*.

QueueContext also contains the carrier set *ITEM*, to represent the items that are contained in a queue.

CONTEXT QueueContext

SETS

TOKEN

ITEM

CONSTANTS

MAXQUEUE

QUEUE

AXIOMS

axm1: $finite(TOKEN)$

axm2: $finite(ITEM)$

axm3: $MAXQUEUE \in \mathbb{N}_1$

axm4: $QUEUE \subseteq 1 .. MAXQUEUE \mapsto TOKEN$

axm5: $finite(QUEUE)$

axm6: $QUEUE = \{q | q \in 1 .. card(q) \mapsto TOKEN\}$

thm1: $\forall q \cdot q \in QUEUE \Rightarrow dom(q) = 1 .. card(q)$

thm1: $\emptyset \in QUEUE$

END

0.4 The QueueA machine

The first model we build uses rather ad-hoc queue operations.

MACHINE QueueA

SEES QueueContext

VARIABLES *queuetokens* tokens for currently queued items *queue* the queue of tokens
queueitems a function that binds the item associated with a token *qsize* current size
of queue

INVARIANTS

inv1: $queuetokens \subseteq TOKEN$

inv2: $queue \in QUEUE$

inv3: $qsize \in \mathbb{N}$

inv4: $queue \in 1..qsize \mapsto queuetokens$

inv5: $\forall i, j \cdot i \in dom(queue) \wedge j \in dom(queue) \wedge i \neq j$
 \Rightarrow
 $queue(i) \neq queue(j)$

inv6: $queuetokens = ran(queue)$

inv7: $queueitems \in queuetokens \rightarrow ITEM$

inv8: $card(queue) = qsize$

inv9: $queue^{-1} \in queuetokens \mapsto 1..qsize$

thm1: $queuetokens \neq \emptyset \Leftrightarrow qsize \neq 0$

EVENTS

Initialisation

begin

act1: $queuetokens := \emptyset$

act2: $queue := \emptyset$

act3: $qsize := 0$

act4: $queueitems := \emptyset$

end

Event *Enqueue* $\hat{=}$

any

item

qid

when

grd1: $item \in ITEM$

grd2: $qid \in TOKEN \setminus queuetokens$

then

act1: $queuetokens := queuetokens \cup \{qid\}$

act2: $queue(qsize + 1) := qid$

act3: $queueitems(qid) := item$

act4: $qsize := qsize + 1$

end

Event *Dequeue* $\hat{=}$

when

grd1: $qsize \neq 0$

then

act1: $queue : |queue' \in QUEUE$
 $\wedge queue' \in 1 .. qsize - 1 \mapsto queuetokens \setminus \{queue(1)\}$
 $\wedge (\forall i \cdot i \in 1 .. qsize - 1 \Rightarrow queue'(i) = queue(i + 1))$

act2: $queueitems := \{queue(1)\} \triangleleft queueitems$

act3: $queuetokens := queuetokens \setminus \{queue(1)\}$

act4: $qsize := qsize - 1$

end

Event *Unqueue* $\hat{=}$

any

qid

when

grd1: $qid \in queuetokens$

grd2: $qsize \neq 0$

then

act1: $queue : |queue' \in 1..(qsize - 1) \mapsto queuetokens \setminus \{qid\}$
 $\wedge (qsize = 1 \Rightarrow queue' = \emptyset)$
 $\wedge (qsize > 1 \Rightarrow$
 $(\forall i \cdot i \in 1..queue^{-1}(qid) - 1 \Rightarrow queue'(i) = queue(i))$
 \wedge
 $(\forall j \cdot j \in queue^{-1}(qid) + 1..qsize$
 $\Rightarrow queue'(j - 1) = queue(j)))$

act2: $queueitems := \{qid\} \triangleleft queueitems$

act3: $queuetokens := queuetokens \setminus \{qid\}$

act4: $qsize := qsize - 1$

end

END

Next we will define a QueueType with queue operations.

CONTEXT QueueType

EXTENDS QueueContext

CONSTANTS

ENQUEUE

DEQUEUE

DELETE

AXIOMS

axm1: $ENQUEUE \in QUEUE \times TOKEN \mapsto QUEUE$

axm2: $\forall q, t \cdot q \in QUEUE \wedge t \notin ran(q)$
 \Rightarrow
 $q \mapsto t \in dom(ENQUEUE)$

axm3: $\forall q, t \cdot q \in QUEUE \wedge t \notin ran(q)$
 \Rightarrow
 $ENQUEUE(q \mapsto t) = q \triangleleft \{card(q) + 1 \mapsto t\}$

axm4: $\forall q, t \cdot q \in QUEUE \wedge t \in TOKEN \wedge t \notin ran(q)$
 $\Rightarrow card(ENQUEUE(q \mapsto t)) = card(q) + 1$

axm5: $\forall q, t \cdot q \in QUEUE \wedge t \in TOKEN \wedge t \notin ran(q)$
 $\Rightarrow dom(ENQUEUE(q \mapsto t)) = 1..card(q) + 1$

axm6: $\forall q, t, i \cdot q \in QUEUE \wedge t \notin ran(q) \Rightarrow$
 $(i \in dom(q) \Rightarrow ENQUEUE(q \mapsto t)(i) = q(i))$
 \wedge
 $(i = card(q) + 1 \Rightarrow ENQUEUE(q \mapsto t)(i) = t)$

- axm7:** $DEQUEUE \in QUEUE \rightarrow QUEUE$
- axm8:** $dom(DEQUEUE) = QUEUE \setminus \{\emptyset\}$
- axm9:** $\forall q \cdot q \in QUEUE \wedge q \neq \emptyset$
 \Rightarrow
 $DEQUEUE(q) \in 1 .. card(q) - 1 \mapsto ran(q) \setminus \{q(1)\}$
- axm10:** $\forall q \cdot q \in dom(DEQUEUE)$
 \Rightarrow
 $card(DEQUEUE(q)) = card(q) - 1$
- axm11:** $\forall q \cdot q \in dom(DEQUEUE)$
 \Rightarrow
 $dom(DEQUEUE(q)) = 1 .. card(q) - 1$
- axm12:** $\forall q, i \cdot q \in dom(DEQUEUE) \wedge i \in dom(DEQUEUE(q))$
 \Rightarrow
 $DEQUEUE(q)(i) = q(i + 1)$
- axm13:** $DELETE \in QUEUE \times \mathbb{N}_1 \rightarrow QUEUE$
- axm14:** $\forall q, i \cdot q \in QUEUE \wedge i \in dom(q)$
 \Leftrightarrow
 $q \mapsto i \in dom(DELETE)$
- axm15:** $\forall q, i \cdot q \mapsto i \in dom(DELETE)$
 \Rightarrow
 $card(DELETE(q \mapsto i)) = card(q) - 1$
- axm16:** $\forall q, i \cdot q \mapsto i \in dom(DELETE)$
 \Rightarrow
 $dom(DELETE(q \mapsto i)) = 1 .. card(q) - 1$
- axm17:** $\forall q, i, j \cdot q \mapsto i \in dom(DELETE)$
 \Rightarrow
 $(j < i \wedge j \in dom(q) \Rightarrow DELETE(q \mapsto i)(j) = q(j))$
 \wedge
 $(j \geq i \wedge j + 1 \in dom(q) \Rightarrow DELETE(q \mapsto i)(j) = q(j + 1))$

END

0.5 The QueueB machine

Then refine QueueA to QueueB using QueueType

MACHINE QueueB

REFINES QueueA

SEES QueueType

VARIABLES

- queuetokens* tokens for currently queued items
- queue* the queue of tokens
- queueitems* a function for fetching the item associated with a token
- qsize* current size of queue

INVARIANTS

- inv1:* $queuetokens \subseteq TOKEN$
- inv2:* $queue \in QUEUE$
- inv3:* $qsize = card(queue)$
- inv4:* $queue \in 1 .. qsize \mapsto queuetokens$
- inv5:* $\forall i, j \cdot i \in dom(queue) \wedge j \in dom(queue) \wedge i \neq j$
 \Rightarrow
 $queue(i) \neq queue(j)$
- inv6:* $queuetokens = ran(queue)$
- inv7:* $queueitems \in queuetokens \rightarrow ITEM$
- inv8:* $queue^{-1} \in queuetokens \mapsto 1 .. qsize$
- inv9:* $(\forall qid \cdot qid \in TOKEN \setminus queuetokens$
 \Rightarrow
 $ENQUEUE(queue \mapsto qid) = queue \Leftarrow \{qsize + 1 \mapsto qid\})$
- inv10:* $\forall qid \cdot qid \in queuetokens$
 \Rightarrow
 $queue \mapsto queue^{-1}(qid) \in dom(DELETE)$
- inv11:* $qsize \neq 1$
 \Rightarrow
 $(\forall qid, i \cdot qid \in queuetokens \wedge i \in 1 .. (queue^{-1}(qid) - 1)$
 \Rightarrow
 $(DELETE(queue \mapsto queue^{-1}(qid)))(i) = queue(i))$
- inv12:* $qsize \neq 1$
 \Rightarrow
 $(\forall qid, i \cdot qid \in queuetokens \wedge i \in queue^{-1}(qid) + 1 .. qsize$
 \Rightarrow
 $(DELETE(queue \mapsto queue^{-1}(qid)))(i - 1) = queue(i))$
- inv13:* $\forall qid \cdot qid \in queuetokens$
 \Rightarrow
 $queue^{-1}(qid) \leq qsize$

EVENTS

Initialisation

begin

act1: $queuetokens := \emptyset$

act2: $queue := \emptyset$

act3: $qsize := 0$

act4: $queueitems := \emptyset$

end

Event *Enqueue* $\hat{=}$

refines *Enqueue*

any

item

qid

when

grd1: $item \in ITEM$

grd2: $qid \in TOKEN \setminus queuetokens$

then

act1: $queuetokens := queuetokens \cup \{qid\}$

act2: $queue := ENQUEUE(queue \mapsto qid)$

act3: $queueitems(qid) := item$

act4: $qsize := qsize + 1$

end

Event *Dequeue* $\hat{=}$

refines *Dequeue*

when

grd1: $qsize \neq 0$

then

act1: $queue := DEQUEUE(queue)$

act2: $queueitems := \{queue(1)\} \triangleleft queueitems$

act3: $queuetokens := queuetokens \setminus \{queue(1)\}$

act4: $qsize := qsize - 1$

```

end
Event Unqueue  $\hat{=}$ 
refines Unqueue
any
  qid
when
  grd1:  $qid \in \text{queuetokens}$ 
then
  act1:  $queue := DELETE(queue \mapsto queue^{-1}(qid))$ 
  act2:  $queueitems := \{qid\} \triangleleft queueitems$ 
  act3:  $queuetokens := queuetokens \setminus \{qid\}$ 
  act4:  $qsize := qsize - 1$ 
end
END

```

0.6 Refining the Queue machine

The refinement replaces the monolithic sequence model by a list model, in which the discrete elements of the set *queuetokens* are organised as a list using the following variables:

qfirst the first element of the list;

qlast the last element of the list;

qnext a function that links an element of the list to the next element in the list —relevant only to lists with more than one item;

qsize the size of the list.

picture required

Additionally, the refinement uses the variable *queueitem* in the same role as in the *Queue* machine. Although this variable has the same name it is a new variable that is related by equivalence to the variable in the refined machine.

A refinement relation relates the list model to the queue model.

Data refinements may not use variables of the refined machine except in invariants. Complete hiding is enforced.

Relational composition and iteration

Since we are modelling a list structure we will use *relational composition* on the *qnext* function to describes paths along the list, and we will use transitive *closure* of *qnext* to describe reachability.

Suppose we have a list with at least 2 elements, then

<i>qfirst</i>	gives the identity of the first item in the list
<i>qnext(qfirst)</i>	gives the identity of the second item in the list
<i>(qnext ; qnext)(qfirst)</i>	gives the identity of the third item in the list
...	etc

Multiple composition is expressed by *iteration*: $qnext^n$ (provided by the constant function $iterate(qnext \mapsto n)$), is the result of composing *qnext* with itself n times.

If $r \in X \leftrightarrow X$, then $r^0 = id(X)$ and $r^{n+1} = r^n ; r$.

Closure

Reflexive transitive closure of a relation r , written r^* , is the union of all iterations of r , that is

$$r^* = \bigcup_{n \in \mathbb{N}} (n \mid r^n)^\dagger$$

Irreflexive transitive closure of a relation, written r^+ , does not explicitly include r^0 from the union

$$r^+ = \bigcup_{n \in \mathbb{N}_1} (n \mid r^n),$$

but it may be present, depending on r . EventB (RODIN) does not supply *closure*; it has to be defined as a constant function.

Relational composition of functions

It should be clear that if f is a function then $f ; f$ is also a function and by extrapolation f^n is a function.

Further, if f is an injective function then f^n is also an injective function.

Thus, $qnext^n$ is an injective function that gives all paths of length n within the list.

$qnext^+$ is a set of injective functions representing all paths, of all lengths from 0 to the length of the list, within the list.

It follows that $qnext^+[\{qfirst\}]$, the image of the first node in the list under $qnext^+$, is the set of all nodes in the list.

CONTEXT Iteration

EXTENDS QueueType

CONSTANTS

`iterate`

`iclosure`

AXIOMS

[†]It should be clear that continuous composition of a relation with itself will eventually reach a stationary relation.

axm1: $iterate \in (TOKEN \leftrightarrow TOKEN) \times \mathbb{N} \rightarrow (TOKEN \leftrightarrow TOKEN)$

axm2: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN \Rightarrow iterate(r \mapsto 0) = dom(r) \triangleleft id$

axm3: $\forall r, n \cdot r \in TOKEN \leftrightarrow TOKEN \wedge n \in \mathbb{N}_1$
 \Rightarrow
 $iterate(r \mapsto n) = iterate(r \mapsto n - 1); r$

thm1: $\forall s \cdot s \subseteq \mathbb{N} \wedge 0 \in s \wedge (\forall n \cdot n \in s \Rightarrow n + 1 \in s) \Rightarrow \mathbb{N} \subseteq s$

thm2: $\forall r, n \cdot r \in TOKEN \leftrightarrow TOKEN \wedge n \in \mathbb{N}_1$
 \Rightarrow
 $dom(iterate(r \mapsto n)) \subseteq dom(r)$

thm3: $\forall r, n \cdot r \in TOKEN \leftrightarrow TOKEN \wedge n \in \mathbb{N}_1$
 \Rightarrow
 $ran(iterate(r \mapsto n)) \subseteq ran(r)$

thm4: $iclosure \in (TOKEN \leftrightarrow TOKEN) \rightarrow (TOKEN \leftrightarrow TOKEN)$

axm5: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN$
 \Rightarrow
 $iclosure(r) = (\bigcup n \cdot n \in \mathbb{N}_1 | iterate(r \mapsto n))$

thm5: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN$
 \Rightarrow
 $dom(iclosure(r)) \subseteq dom(r)$

END

0.7 The QueueR invariant

The list consists of the elements of *queuetokens* hence

$$qsize = card(queuetokens)$$

For non-empty lists, *qfirst* and *qlast* are elements of *queuetokens*

$$queuetokens \neq \emptyset \implies qfirst \in queuetokens$$
$$queuetokens \neq \emptyset \implies qlast \in queuetokens$$

The list is linear and connected, hence *qnext* is injective, but it is also surjective and therefore bijective:

$$qnext \in queuetokens \setminus \{qlast\} \rightsquigarrow queuetokens \setminus \{qfirst\}$$

0.8 The Refinement relation

Each element of the queue model can be retrieved from the list model

$$\forall i \cdot i \in 1 .. qsize \implies queue(i) = qnext^{i-1}(qfirst)$$

0.9 QueueR Theorems

The following should follow from the invariant:

1. Any element of the list that is not *qfirst* must be in $\text{dom}(qnext)$

$$\forall t. t \in \text{queuetokens} \wedge \text{qsize} > 1 \wedge t \neq \text{qlast} \Rightarrow t \in \text{dom}(qnext)$$

2. Any element of the list that is not *qlast* must be in $\text{ran}(qnext)$

$$\forall t. t \in \text{queuetokens} \wedge \text{qsize} > 1 \wedge t \neq \text{qfirst} \Rightarrow t \in \text{ran}(qnext)$$

3. Following all sequences of *qnext* from *qfirst* should give all tokens in *queuetokens*

$$\text{closure1}(qnext)[\{\text{qfirst}\}] = \text{queuetokens}$$

4. The following should also follow from the refinement relation:

$$\text{qsize} \neq 0 \Rightarrow \text{queue}(1) = \text{qfirst}$$

$$\text{qsize} \neq 0 \Rightarrow \text{queue}(\text{qsize}) = \text{qlast}$$

$$\text{qsize} \neq 0 \Rightarrow \forall i. i \in 1 .. \text{qsize} - 1 \Rightarrow \text{queue}(i + 1) = \text{qnext}(\text{queue}(i))$$

Loops

There must be no loops. When moving from a monolithic structure to a list it is clear that loops are possible. It is easy to see by informal induction on the way the list is built that there will be no loops, but it follows from the type of *qnext*, so the following should be a theorem:

$$qnext^+ \cap \text{id}(\text{queuetokens}) = \emptyset$$

Traversing the list from *qfirst* should cover all the elements of *queuetokens*

$$qnext^+[\{\text{qfirst}\}] = \text{queuetokens}$$

0.9.1 The QueueR machine

MACHINE QueueR

REFINES QueueB

SEES Iteration

VARIABLES

queuetokens tokens currently in queue

queueitems a function for fetching the item associated with a token

qsize current size of queue

qfirst first item, if any, in queue

qlast last item, if any, in queue

qnext link to next item, if any, in queue

INVARIANTS

inv1: $qfirst \in TOKEN$

inv2: $qlast \in TOKEN$

inv3: $qsize \neq 0 \Rightarrow qfirst = queue(1)$

inv4: $qsize \neq 0 \Rightarrow qlast = queue(qsize)$

inv5: $qnext \in queuetokens \leftrightarrow queuetokens$

inv6: $dom(qnext) = queuetokens \setminus \{qlast\}$

inv7: $qnext \cap id = \emptyset$

inv8: $ran(qnext) = queuetokens \setminus \{qfirst\}$

inv9: $qsize = 1 \Rightarrow qfirst = qlast$

inv10: $\forall i \cdot i \in 1 .. qsize \wedge i < qsize$
 \Rightarrow
 $qnext(queue(i)) = queue(i + 1)$

inv11: $qsize \geq 1$
 $\Rightarrow iterate(qnext \mapsto 0)[\{qfirst\}] = \{queue(1)\}$

inv12: $qsize \geq 1$
 $\Rightarrow (\forall n \cdot n \in 1 .. qsize - 1$
 $\wedge iterate(qnext \mapsto n - 1)[\{qfirst\}] = \{queue(n)\}$
 \Rightarrow
 $iterate(qnext \mapsto n)[\{qfirst\}] = \{queue(n + 1)\})$

inv13: $qsize \geq 1$
 $\Rightarrow (\forall n \cdot n \in 1 .. qsize - 1$
 $\Rightarrow iterate(qnext \mapsto n - 1)[\{qfirst\}] = \{queue(n)\})$

inv14: $qsize \geq 1 \Rightarrow iclosure(qnext)[\{qfirst\}] = queuetokens$

EVENTS

Initialisation

begin

act1: $queuetokens := \emptyset$

act2: $qsize := 0$

act3: $queueitems := \emptyset$

act4: $qfirst := TOKEN$

act5: $qlast := TOKEN$

act6: $qnext := \emptyset$
end
Event $Enqueue0 \hat{=}$
refines $Enqueue$
any
 $item$
 qid
when
grd1: $item \in ITEM$
grd2: $qid \in TOKEN \setminus queuetokens$
grd3: $qsize = 0$
then
act1: $queuetokens := queuetokens \cup \{qid\}$
act2: $queueitems(qid) := item$
act3: $qsize := qsize + 1$
act4: $qfirst := qid$
act5: $qlast := qid$
end
Event $Enqueue1 \hat{=}$
refines $Enqueue$
any
 $item$
 qid
when
grd1: $item \in ITEM$
grd2: $qid \in TOKEN \setminus queuetokens$
grd3: $qsize \neq 0$
then
act1: $queuetokens := queuetokens \cup \{qid\}$
act2: $queueitems(qid) := item$

act3: $qsize := qsize + 1$

act4: $qnext(qlast) := qid$

act5: $qlast := qid$

end

Event *Dequeue0* $\hat{=}$

refines *Dequeue*

when

grd1: $qsize = 1$

then

act1: $qsize := qsize - 1$

act2: $queuetokens := queuetokens \setminus \{qfirst\}$

act3: $queueitems := \{qfirst\} \triangleleft queueitems$

act4: $qnext := \{qfirst\} \triangleleft qnext$

end

Event *Dequeue1* $\hat{=}$

refines *Dequeue*

when

grd1: $qsize > 1$

then

act1: $qsize := qsize - 1$

act2: $queuetokens := queuetokens \setminus \{qfirst\}$

act3: $queueitems := \{qfirst\} \triangleleft queueitems$

act4: $qfirst := qnext(qfirst)$

act5: $qnext := \{qfirst\} \triangleleft qnext$

end

Event *Unqueue0* $\hat{=}$

refines *Unqueue*

any

qid

when

grd1: $qid \in \text{queuetokens}$
grd2: $qsize = 1$
then
act1: $\text{queueitems} := \{qid\} \triangleleft \text{queueitems}$
act2: $\text{queuetokens} := \text{queuetokens} \setminus \{qid\}$
act3: $qsize := qsize - 1$
end
Event *Unqueue1* $\hat{=}$
refines *Unqueue*
any
qid
when
grd1: $qid \in \text{queuetokens}$
grd2: $qsize > 1$
grd3: $qid = qfirst$
then
act1: $\text{queueitems} := \{qid\} \triangleleft \text{queueitems}$
act2: $\text{queuetokens} := \text{queuetokens} \setminus \{qid\}$
act3: $qsize := qsize - 1$
act4: $qfirst := qnext(qid)$
act5: $qnext := \{qid\} \triangleleft qnext$
end
Event *Unqueue2* $\hat{=}$
refines *Unqueue*
any
qid
when
grd1: $qid \in \text{queuetokens}$
grd2: $qsize > 1$
grd3: $qlast = qid$

then

act1: $queueitems := \{qid\} \triangleleft queueitems$

act2: $queuetokens := queuetokens \setminus \{qid\}$

act3: $qsize := qsize - 1$

act4: $qlast := qnext^{-1}(qid)$

act5: $qnext := qnext \triangleright \{qid\}$

end

Event $Unqueue3 \hat{=}$

refines $Unqueue$

any

qid

when

grd1: $qid \in queuetokens$

grd2: $qsize > 1$

grd3: $qfirst \neq qid$

grd4: $qlast \neq qid$

then

act1: $queueitems := \{qid\} \triangleleft queueitems$

act2: $queuetokens := queuetokens \setminus \{qid\}$

act3: $qsize := qsize - 1$

act4: $qnext(qnext^{-1}(qid)) := qnext(qid)$

end

END

0.9.2 Refinement of Unqueue3

Refinement of Unqueue3

The event $Unqueue3$ deletes an item from within the queue, that is neither the first or last items on the queue.

Implementing $prev$

Until now we got $prev$ for free because $qnext$ is an injective function, so $prev$ has been obtained by simply inverting $qnext$. In an implementation we have no such luxury. In the refinement of $Unqueue3$ we implement $prev$ by using a loop to search from the beginning of the queue (list) for the predecessor of the item to be deleted. This, of course, is inefficient. If efficiency is important, we could implement a doubly linked list, ie implement $qprev$.

Preventing Interference

The refinement of Unqueue3 consists of three events:

Unqueue3I: initiates the computation of $qprev$. This event sets $qprev$ to $qfirst$ and sets a flag, $deleting$, to $TRUE$.

Unqueue3M: an event that represents *still searching*. It advances $qprev$ to $qnext(qprev)$.

Unqueue3F: the final step. The item to be deleted has been found, so the current value of $qprev$ is the value we want. This event does the deletion and sets $deleting$ to $FALSE$.

The purpose of *deleting*

Until the deletion is complete the other queue events must not run as the state of the queue is not yet correct. Until now *Unqueue3* was an atomic event; in this refinement the actions of that event are spread across three events.

QueueRR

MACHINE QueueRR

REFINES QueueR

SEES Iteration

VARIABLES

queuetokens tokens currently in queue

queueitems a function for fetching the item associated with a token

qsize current size of queue

qfirst first item, if any, in queue

qlast last item, if any, in queue

qnext link to next item, if any, in queue

deleting Unqueue deletion in progress

qprev concrete version of queue

qidv copy of qid

INVARIANTS

inv1: $deleting \in \text{BOOL}$

inv2: $qprev \in \text{TOKEN}$

inv3: $qidv \in \text{TOKEN}$

inv4: $deleting = \text{TRUE} \Rightarrow qidv \in \text{queuetokens}$

inv5: $deleting = \text{TRUE} \Rightarrow qidv \neq qfirst$

inv6: $deleting = TRUE \Rightarrow qsize > 1$

inv7: $deleting = TRUE \Rightarrow qprev \in dom(qnext)$

inv8: $deleting = TRUE \Rightarrow qidv \in iclosure(qnext)[\{qprev\}]$

EVENTS

Initialisation *extended*

begin

act1: $queuetokens := \emptyset$

act2: $qsize := 0$

act3: $queueitems := \emptyset$

act4: $qfirst \in TOKEN$

act5: $qlast \in TOKEN$

act6: $qnext := \emptyset$

act7: $deleting := FALSE$

act8: $qprev \in TOKEN$

act9: $qidv \in TOKEN$

end

Event *Enqueue0* $\hat{=}$

extends *Enqueue0*

any

item

qid

when

grd1: $item \in ITEM$

grd2: $qid \in TOKEN \setminus queuetokens$

grd3: $qsize = 0$

grd4: $deleting = FALSE$

then

act1: $queuetokens := queuetokens \cup \{qid\}$

act2: $queueitems(qid) := item$

act3: $qsize := qsize + 1$

```

act4:  $qfirst := qid$ 
act5:  $qlast := qid$ 
end

Event Enqueue1  $\hat{=}$ 
extends Enqueue1
any
item
qid
when
grd1:  $item \in ITEM$ 
grd2:  $qid \in TOKEN \setminus queuetokens$ 
grd3:  $qsize \neq 0$ 
grd4:  $deleting = FALSE$ 
then
act1:  $queuetokens := queuetokens \cup \{qid\}$ 
act2:  $queueitems(qid) := item$ 
act3:  $qsize := qsize + 1$ 
act4:  $qnext(qlast) := qid$ 
act5:  $qlast := qid$ 
end

Event Dequeue0  $\hat{=}$ 
extends Dequeue0
when
grd1:  $qsize = 1$ 
grd2:  $deleting = FALSE$ 
then
act1:  $qsize := qsize - 1$ 
act2:  $queuetokens := queuetokens \setminus \{qfirst\}$ 
act3:  $queueitems := \{qfirst\} \triangleleft queueitems$ 
act4:  $qnext := \{qfirst\} \triangleleft qnext$ 

```

end

Event *Dequeue1* $\hat{=}$

extends *Dequeue1*

when

grd1 : $qsize > 1$

grd2 : $deleting = FALSE$

then

act1 : $qsize := qsize - 1$

act2 : $queuetokens := queuetokens \setminus \{qfirst\}$

act3 : $queueitems := \{qfirst\} \triangleleft queueitems$

act4 : $qfirst := qnext(qfirst)$

act5 : $qnext := \{qfirst\} \triangleleft qnext$

end

Event *Unqueue0* $\hat{=}$

extends *Unqueue0*

any

qid

when

grd1 : $qid \in queuetokens$

grd2 : $qsize = 1$

grd3 : $deleting = FALSE$

then

act1 : $queueitems := \{qid\} \triangleleft queueitems$

act2 : $queuetokens := queuetokens \setminus \{qid\}$

act3 : $qsize := qsize - 1$

end

Event *Unqueue1* $\hat{=}$

extends *Unqueue1*

any

qid

when

grd1 : $qid \in queuetokens$

grd2 : $qsize > 1$

grd3 : $qid = qfirst$

grd4 : $deleting = FALSE$

then

act1 : $queueitems := \{qid\} \triangleleft queueitems$

act2 : $queuetokens := queuetokens \setminus \{qid\}$

act3 : $qsize := qsize - 1$

act4 : $qfirst := qnext(qid)$

act5 : $qnext := \{qid\} \triangleleft qnext$

end

Event *Unqueue2* $\hat{=}$

refines *Unqueue2*

when

grd1 : $deleting = TRUE$

grd2 : $qnext(qprev) = qidv$

grd3 : $qlast = qidv$

with

qid : $qid = qidv$

then

act1 : $queueitems := \{qidv\} \triangleleft queueitems$

act2 : $queuetokens := queuetokens \setminus \{qidv\}$

act3 : $qsize := qsize - 1$

act4 : $qlast := qprev$

act5 : $qnext := qnext \triangleright \{qidv\}$

act6 : $deleting := FALSE$

end

Event *Unqueue3* $\hat{=}$

refines *Unqueue3*

when

grd1: $deleting = TRUE$

grd2: $qnext(qprev) = qidv$

grd3: $qidv \neq qlast$

with

qid: $qid = qidv$

then

act1: $queueitems := \{qidv\} \triangleleft queueitems$

act2: $queuetokens := queuetokens \setminus \{qidv\}$

act3: $qsize := qsize - 1$

act4: $qnext(qprev) := qnext(qidv)$

act5: $deleting := FALSE$

end

Event $UnqueueI \hat{=}$

any

qid

when

grd1: $qid \in queuetokens$

grd2: $qsize > 1$

grd3: $qfirst \neq qid$

grd4: $deleting = FALSE$

then

act1: $qprev := qfirst$

act2: $qidv := qid$

act3: $deleting := TRUE$

end

Event $UnqueueS \hat{=}$

Status convergent

when

grd1: $deleting = TRUE$

grd2: $qnext(qprev) \neq qidv$

then

act1: $qprev := qnext(qprev)$

end

VARIANT

$iclosure(qnext)[\{qprev\}]$

END

0.9.3 Notes on the Variant

The variant for the search event is the set of remaining items in the queue from the current item pointed to by *prev*. Clearly we expect that the number of remaining items in that set is finite and decreasing. The set of items is obtained by applying $closure(qnext)$ to *prev*.