```
module QUEUE
  global variables
  Q      :array indexed by 0:m − 1 of n-bit numbers
  HEAD:integer initially 0
  TAIL  :integer initially 0

begin subroutine PUT
     declare I integer;
     while a: ⟨TAIL − HEAD = m⟩ do b: ⟨skip⟩ od;
  c: ⟨I := 0⟩;
     while d: ⟨I < N⟩
           do e: ⟨shift.left 1 (Q[TAIL mod m], PUT.PAR)⟩;
              f: ⟨I := I + 1⟩                                    od;
  g: ⟨TAIL := TAIL + 1⟩
  h:
end subroutine

begin subroutine GET
     declare J integer;
     while r: ⟨TAIL − HEAD = 0⟩ do s: ⟨skip⟩ od;
  t: ⟨J := 0⟩;
     while u: ⟨J < N⟩
           do v: ⟨shift.right 1 (Q[HEAD mod m], GET.PAR)⟩
              w: ⟨J := J + 1⟩                                    od;
  x: ⟨HEAD := HEAD + 1⟩
  y:
end subroutine

end module
```

Fig. 6. An implementation of the QUEUE module.

to the element holding its tail. For simplicity, we let HEAD and TAIL be integers, and use their values modulo m as pointers. In a more realistic implementation, they would be integers modulo $2m$, but this makes the reasoning slightly more complicated. To emphasize that the adding and removing of elements from the queue need not be atomic, these operations are performed by shifting the elements one bit at a time out of or into the ".PAR" variable. This is done with *shift.left* and *shift.right* operations, whose meaning should be obvious. Atomic operations are enclosed by angle brackets. Note that the queue has space for only m elements, and a PUT operation must wait until there is room to add the element.

To prove that this implementation meets the specification of Figure 5, we must first define the state functions *queue*, *parg*, and *gval* in terms of the program state. This requires deciding at what point during the execution of each subroutine the change to the queue is considered to have taken place. It is most convenient to consider the queue to change when HEAD or TAIL is incremented. This leads to the following definitions, where

right.half(shift.right i (p, q))

denotes the right half of the double-length word obtained by applying the *shift.right* i operation to the double-length word (p, q), and

left.half(shift.left i (p, q))

has the analogous meaning.