# Hood-Melville Queue

Alejandro Gómez-Londoño

May 26, 2024

**Abstract**

This is a verified implementation of a constant time queue. The original design is due to Hood and Melville [1]. This formalization follows the presentation by Okasaki [2].

**theory** *Hood-Melville-Queue*
**imports**
*HOL−Data-Structures.Queue-Spec*
**begin**

**datatype** *'a status* =

*Idle*

| *Rev nat 'a list 'a list 'a list 'a list*

| *App nat 'a list 'a list*

| *Done 'a list*

**record** *'a queue* = *lenf* :: *nat*
*front* :: *'a list*
*status* :: *'a status*
*rear* :: *'a list*
*lenr* :: *nat*

**fun** *exec* :: *'a status* ⇒ *'a status* **where**
*exec* (*Rev ok* (*x*#*f*) *f'* (*y*#*r*) *r'*) = *Rev* (*ok*+*1*) *f* (*x*#*f'*) *r* (*y*#*r'*)
| *exec* (*Rev ok* [] *f'* [*y*] *r'*) = *App ok f'* (*y*#*r'*)
| *exec* (*App 0 f' r'*) = *Done r'*
| *exec* (*App ok* (*x*#*f'*) *r'*) = *App* (*ok*−*1*) *f'* (*x*#*r'*)
| *exec s* = *s*

**fun** *invalidate* **where**

$invalidate\ (Rev\ ok\ f\ f'\ r\ r') = Rev\ (ok{-}1)\ f\ f'\ r\ r'$
$|\ invalidate\ (App\ 0\ f'\ (x\#r')) = Done\ r'$
$|\ invalidate\ (App\ ok\ f'\ r') = App\ (ok{-}1)\ f'\ r'$
$|\ invalidate\ s = s$

**fun** *exec2* :: *'a queue* ⇒ *'a queue* **where**
*exec2 q* =
  (*case exec* (*exec* (*status q*)) *of*
    *Done newf* ⇒ *q*⦇*status* := *Idle*,*front* := *newf*⦈ |
    *newstate* ⇒ *q*⦇*status* := *newstate*⦈)

**definition** *check* :: *'a queue* ⇒ *'a queue* **where**
*check q* = (*if lenr q* ≤ *lenf q*
      *then exec2 q*
      *else let newstate* = *Rev 0* (*front q*) [] (*rear q*) []
        *in exec2* (*q*⦇*lenf* := *lenf q* + *lenr q*, *status* := *newstate*, *rear* := [],
*lenr* := *0*⦈))

**definition** *empty* :: *'a queue* **where**
  *empty* = *queue.make 0* [] *Idle* [] *0*

**fun** *enq* **where**
  *enq x q* = *check* (*q*⦇*rear* := *x*#(*rear q*), *lenr* := *lenr q* + *1*⦈)

**fun** *deq* **where**
  *deq q* = *check* (*q*⦇ *lenf* := *lenf q* − *1*
      , *front* := *tl* (*front q*)
      , *status* := *invalidate* (*status q*)⦈)

**fun** *front-list* :: *'a queue* ⇒ *'a list* **where**
  *front-list q* = (*case status q of*
      *Idle* ⇒ *front q*
      | *Done f* ⇒ *f*
      | *Rev ok f f' r r'* ⇒ *rev* (*take ok f'*) @ *f* @ *rev r* @ *r'*
      | *App ok f' r'* ⇒ *rev* (*take ok f'*) @ *r'*)

**definition** *rear-list* :: *'a queue* ⇒ *'a list* **where**
  *rear-list* = *rev o rear*

**fun** *list* :: *'a queue* ⇒ *'a list* **where**
  *list q* = *front-list q* @ *rear-list q*

**fun** *first* :: *'a queue* ⇒ *'a* **where**
*first q* = *hd* (*front q*)

**fun** *rem-steps* :: *'a status* ⇒ *nat* **where**
*rem-steps* (*Rev ok f f' r r'*) = *2*∗*length f* + *ok* + *2*
| *rem-steps* (*App ok f' r'*) = *ok* + *1*
| *rem-steps* - = *0*

**fun** *inv-st* :: *'a status* ⇒ *bool* **where**
*inv-st* (*Rev ok f f' r r'*) = (*length f* + *1* = *length r* ∧
*length f'* = *length r'* ∧
*ok* ≤ *length f'*)
| *inv-st* (*App ok f' r'*) = (*ok* ≤ *length f'* ∧ *length f'* < *length r'*)
| *inv-st* - = *True*

**fun** *steps* :: *nat* ⇒ *'a status* ⇒ *'a status* **where**
*steps n st* = (*exec* ^^ *n*) *st*

**lemma** *rev-steps-app*:
**assumes** *inv*: *inv-st* (*Rev ok f f' r r'*)
**shows** *steps* (*length f* + *1*) (*Rev ok f f' r r'*) = *App* (*length f* + *ok*) (*rev f* @ *f'*) (*rev r* @ *r'*)
⟨*proof*⟩

**lemma** *inv-st-steps*:
**assumes** *inv* : *inv-st s*
**assumes** *not-idle* : *s* ≠ *Idle*
**shows** ∃ *x*. *steps* (*rem-steps s*) *s* = *Done x* (**is** *?reach-done s*)
⟨*proof*⟩

**lemma** *inv-st-exec*:
**assumes** *inv-st*: *inv-st s*
**shows** *inv-st* (*exec s*)
⟨*proof*⟩

**lemma** *inv-st-exec2*:
**assumes** *inv-st*: *inv-st s*
**shows** *inv-st* (*exec* (*exec s*))
⟨*proof*⟩

**lemma** *inv-st-invalidate*:

**assumes** *inv-st*: *inv-st s*
**shows** *inv-st* (*invalidate s*)
⟨*proof*⟩

**definition** *invar* **where**
*invar q* = (*lenf q* = *length* (*front-list q*) ∧
*lenr q* = *length* (*rear-list q*) ∧
*lenr q* ≤ *lenf q* ∧
(*case status q of*
*Rev ok f f′ r r′* ⇒ *2*∗*lenr q* ≤ *length f′* ∧ *ok* ≠ *0* ∧ *2*∗*length f* + *ok* + *2* ≤ *2*∗*length* (*front q*)
| *App ok f r* ⇒ *2*∗*lenr q* ≤ *length r* ∧ *ok* + *1* ≤ *2*∗*length* (*front q*)
| - ⇒ *True*) ∧
(∃ *rest*. *front-list q* = *front q* @ *rest*) ∧
(¬(∃ *fr*. *status q* = *Done fr*)) ∧
*inv-st* (*status q*))

**lemma** *invar-empty*: *invar empty*
⟨*proof*⟩

**lemma** *tl-rev-take*: ⟦*0* < *ok*; *ok* ≤ *length f*⟧ ⟹ *rev* (*take ok* (*x* # *f*)) = *tl* (*rev* (*take ok f*)) @ [*x*]
⟨*proof*⟩

**lemma** *tl-rev-take-Suc*:
*n* + *1* ≤ *length l* ⟹ *rev* (*take n l*) = *tl* (*rev* (*take* (*Suc n*) *l*))
⟨*proof*⟩

**lemma** *invar-deq*:
**assumes** *inv*: *invar q*
**shows** *invar* (*deq q*)
⟨*proof*⟩

**lemma** *invar-enq*:
**assumes** *inv*: *invar q*
**shows** *invar* (*enq x q*)
⟨*proof*⟩

**lemma** *queue-correct-deq* :
**assumes** *inv*: *invar q*
**shows** *list* (*deq q*) = *tl* (*list q*)
⟨*proof*⟩

**lemma** *queue-correct-enq* :
  **assumes** *inv*: *invar q*
  **shows** *list* (*enq x q*) = (*list q*) @ [*x*]
⟨*proof*⟩

**datatype** *'a action* = *Deq* | *Enq 'a*

**type-synonym** *'a actions* = *'a action list*

**fun** *do-act* :: *'a action* ⇒ *'a queue* ⇒ *'a queue* **where**
  *do-act Deq q* = *deq q*
| *do-act* (*Enq x*) *q* = *enq x q*

**definition** *qfa* :: *'a actions* ⇒ *'a queue* **where**
  *qfa* = (λ*acts*. *foldr do-act acts empty*)

**lemma** *invar-qfa* : *invar* (*qfa l*)
⟨*proof*⟩

**lemma** *qfa-deq-correct*: *list* (*deq* (*qfa l*)) = *tl* (*list* (*qfa l*))
  ⟨*proof*⟩

**lemma** *qfa-enq-correct*: *list* (*enq x* (*qfa l*)) = (*list* (*qfa l*)) @ [*x*]
  ⟨*proof*⟩

**lemma** *first-correct* :
  **assumes** *inv*: *invar q*
  **assumes** *not-nil* : *list q* ≠ []
  **shows** *first q* = *hd* (*list q*)
⟨*proof*⟩

**fun** *is-empty* :: *'a queue* ⇒ *bool* **where**
  *is-empty q* = (*list q* = [])

**interpretation** *HMQ*: *Queue* **where**
  *empty* = *empty* **and**
  *enq* = *enq* **and**
  *first* = *first* **and**
  *deq* = *deq* **and**
  *is-empty* = *is-empty* **and**
  *list* = *list* **and**
  *invar* = *invar*

⟨*proof*⟩

**end**

# References


[1] R. Hood and R. Melville. Real-time queue operation in pure LISP. *Inf. Process. Lett.*, 13(2):50–54, 1981.

[2] C. Okasaki. *Purely Functional Data Structures.* Cambridge University Press, 1998.