

Master's Thesis Exam

Verification of the Blocking and Non-Blocking Michael-Scott Queue Algorithms

Mathias Pedersen, 201808137

Advisor: Amin Timany

Aarhus University

June 2024



AARHUS
UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Overview of the Project and Contributions

► create slide◄

- 1 Queue Specifications
- 2 The Two-Lock Michael-Scott Queue
- 3 Proving that the Two-Lock Michael-Scott Queue Satisfies the HOCAP-style Specification
- 4 The Lock-Free Michael-Scott Queue
- 5 Proving that the Lock-and-CC-Free Michael-Scott Queue Satisfies the HOCAP-style Specification

Queue Specifications

Specifications for Queues

Assumptions on Queues

- Queues consists of initialize, enqueue, and dequeue
- initialize creates an empty queue: $[]$
- enqueue adds a value, v , to the beginning of the queue $xs_v: v :: xs_v$
- dequeue depends on whether queue is empty:
 - If non-empty, $xs_v ++ v$, remove v and return $\text{Some } v$
 - If empty, $[]$, return None

Nature of Specifications

- Specifications written in Iris, a higher order CSL
- Expressed in terms of *Hoare triples*: $\{P\} e \{v. \Phi \ v\}$
- Hoare triples prove partial correctness of programs, e
- In particular: safety
- Idea: clients can use Hoare triples to prove results about their own code

Sequential Specification

Definition (Sequential Specification)

$\exists \text{isQueue}_S : \text{Val} \rightarrow \text{List Val} \rightarrow \text{SeqQgnames} \rightarrow \text{Prop.}$

$\{\text{True}\} \text{ initialize } () \{v_q. \exists G. \text{isQueue}_S(v_q, [], G)\}$

$\wedge \quad \forall v_q, v, xs_v, G. \{\text{isQueue}_S(v_q, xs_v, G)\} \text{ enqueue } v_q \ v \{w. \text{isQueue}_S(v_q, (v :: xs_v), G)\}$

$\wedge \quad \forall v_q, xs_v, G. \{\text{isQueue}_S(v_q, xs_v, G)\}$

$\text{ dequeue } v_q$

$\left\{ w. \begin{array}{l} (xs_v = [] * w = \text{None} * \text{isQueue}_S(v_q, xs_v, G)) \vee \\ (\exists v, xs'_v. xs_v = xs'_v ++ [v] * w = \text{Some } v * \text{isQueue}_S(v_q, xs'_v, G)) \end{array} \right\}$

- The proposition $\text{isQueue}_S(v_q, xs_v, G)$, states that value v_q represents the queue, which contains elements xs_v
- $G \in \text{SeqQgnames}$ is a collection of ghost names (depends on specific queue)
- Specification consists of three Hoare triples – one for each queue function
- Important: isQueue_S not required to be persistent!

Definition (Concurrent Specification)

$\exists \text{isQueue}_C : (Val \rightarrow \text{Prop}) \rightarrow Val \rightarrow \text{ConcQnames} \rightarrow \text{Prop}.$

$\forall \Psi : Val \rightarrow \text{Prop}.$

$\forall v_q, G. \text{isQueue}_C(\Psi, v_q, G) \implies \Box \text{isQueue}_C(\Psi, v_q, G)$

$\wedge \{ \text{True} \} \text{ initialize } () \{ v_q. \exists G. \text{isQueue}_C(\Psi, v_q, G) \}$

$\wedge \forall v_q, v, G. \{ \text{isQueue}_C(\Psi, v_q, G) * \Psi(v) \} \text{ enqueue } v_q \ v \{ w. \text{True} \}$

$\wedge \forall v_q, G. \{ \text{isQueue}_C(\Psi, v_q, G) \} \text{ dequeue } v_q \{ w. w = \text{None} \vee (\exists v. w = \text{Some } v * \Psi(v)) \}$

HOCAP-style Specification

Definition (HOCAP Specification)

$\exists \text{isQueue} : \text{Val} \rightarrow \text{Qgnames} \rightarrow \text{Prop}.$

$\forall v_q, G. \text{isQueue}(v_q, G) \implies \Box \text{isQueue}(v_q, G)$

$\wedge \{ \text{True} \} \text{ initialize } () \{ v_q. \exists G. \text{isQueue}(v_q, G) * G.\gamma_{\text{Abst}} \mapsto_{\circ} [] \}$

$\wedge \forall v_q, v, G, P, Q. \left(\forall xs_v. G.\gamma_{\text{Abst}} \mapsto_{\bullet} xs_v * P \Rightarrow_{\mathcal{E} \setminus \mathcal{N}.i \uparrow} \triangleright G.\gamma_{\text{Abst}} \mapsto_{\bullet} (v :: xs_v) * Q \right) \multimap$
 $\{ \text{isQueue}(v_q, G) * P \} \text{ enqueue } v_q \ v \{ w.Q \}$

$\wedge \forall v_q, G, P, Q.$

$\left(\forall xs_v. G.\gamma_{\text{Abst}} \mapsto_{\bullet} xs_v * P \Rightarrow_{\mathcal{E} \setminus \mathcal{N}.i \uparrow} \triangleright \left(\begin{array}{l} (xs_v = [] * G.\gamma_{\text{Abst}} \mapsto_{\bullet} xs_v * Q(\text{None})) \\ \vee \left(\begin{array}{l} \exists v, xs'_v. xs_v = xs'_v ++ [v] * \\ G.\gamma_{\text{Abst}} \mapsto_{\bullet} xs'_v * Q(\text{Some } v) \end{array} \right) \end{array} \right) \right)$
 $\{ \text{isQueue}(v_q, G) * P \} \text{ dequeue } v_q \{ w.Q(w) \}$

Queue Client - A PoC Client

- Idea: a minimal client complex enough to require HOCAP specification
- Concurrent, so sequential specification insufficient
- Relies on dequeues not returning None, so concurrent specification insufficient
- HOCAP specification supports consistency and allows us to track queue contents, allowing us to exclude cases where dequeue returns None

```
unwrap w  $\triangleq$  match w with None  $\Rightarrow$  () () | Some v  $\Rightarrow$  v end
```

```
enqdeq vq c  $\triangleq$  enqueue vq c; unwrap(dequeue vq)
```

```
queueAdd a b  $\triangleq$   
  let vq = initialize () in  
  let p = (enqdeq vq a) || (enqdeq vq b) in  
  fst p + snd p
```

Queue Client - A PoC Client (continued)

Lemma (QueueAdd Specification)

$$\forall a, b \in \mathbb{Z}. \{True\} \text{ queueAdd } a \ b \{v.v = a + b\}$$

- Proof idea: Create invariant capturing possible states of queue contents
- Tokens are used to reason about which state we are in

Definition (Invariant for QueueAdd)

$$\begin{aligned} I_{QA}(G, Ga, a, b) \triangleq & G.\gamma_{\text{Abst}} \Rightarrow_{\circ} [] * \text{TokD1 } Ga * \text{TokD2 } Ga \vee \\ & G.\gamma_{\text{Abst}} \Rightarrow_{\circ} [a] * \text{TokA } Ga * (\text{TokD1 } Ga \vee \text{TokD2 } Ga) \vee \\ & G.\gamma_{\text{Abst}} \Rightarrow_{\circ} [b] * \text{TokB } Ga * (\text{TokD1 } Ga \vee \text{TokD2 } Ga) \vee \\ & G.\gamma_{\text{Abst}} \Rightarrow_{\circ} [a; b] * \text{TokA } Ga * \text{TokB } Ga \vee \\ & G.\gamma_{\text{Abst}} \Rightarrow_{\circ} [b; a] * \text{TokB } Ga * \text{TokA } Ga \vee \end{aligned}$$

The Two-Lock Michael-Scott Queue

Implementation: initialize

► create slide◄

Implementation: enqueue

► create slide ◀

Implementation: dequeue

► create slide ◀

Observations on Behaviour of the Two-Lock M&S Queue

► create slide ◀

Proving that the Two-Lock Michael-Scott Queue Satisfies the HOCAP-style Specification

Invariant

► create slide ◀

The isLL Predicate

► create slide ◀

Queue Predicate

► create slide◄

Proof of Initialise

► create slide ◀

Proof of ►enqueue **xor** dequeue◄

►create slide◄

The Lock-Free Michael-Scott Queue

Implementation: initialize

► create slide ◀

Implementation: enqueue

► create slide ◀

Implementation: dequeue

► create slide ◀

Prophecies

►create slide◄

The Lock-and-CC-Free Michael-Scott Queue

►format◀

```
initialize  $\triangleq$ 
  let node = ref (None, ref (None)) in
  ref (ref (node), ref (node))

enqueue Q value  $\triangleq$ 
  let node = ref (Some value, ref (None)) in
  (rec loop_ =
    let tail = !(snd(! Q)) in
    let next = !(snd(! tail)) in
    if next = None then
      if CAS (snd(! tail)) next node then
        CAS (snd(! Q)) tail node
      else loop ()
    else CAS (snd(! Q)) tail next; loop ()
  ) ()

dequeue Q  $\triangleq$ 
  (rec loop_ =
    let head = !(fst(! Q)) in
    let tail = !(snd(! Q)) in
    let next = !(snd(! head)) in
    if head = tail then
      if next = None then
        None
      else
        CAS(snd(! Q)) tail next; loop ()
    else
      let value = fst(! next) in
      if CAS (fst(! Q)) head next then
        value
```

Proving that the Lock-and-CC-Free Michael-Scott Queue Satisfies the HOCAP-style Specification

► create slide ◀

Invariant

► create slide ◀

Queue Predicate

► create slide◄

In Coq!

