The Best Queue Specifications You Will Ever See today probably

Mathias Pedersen

Aarhus University

November 2025

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Context

- Based on my Master's Thesis
- Goal of project was to prove safety of two concurrent queues
- Success! – but not too interesting

Context

- Based on my Master's Thesis
- Goal of project was to prove safety of two concurrent queues
- Success! – but not too interesting
- Today: Queue Specifications

Context

- Based on my Master's Thesis
- Goal of project was to prove safety of two concurrent queues
- Success! – but not too interesting
- Today: Queue Specifications
- In particular, three different specifications
  - Sequential specification
  - Concurrent specification
    - Doesn't track queue contents
  - HOCAP-style specification
    - Tracks queue contents with added complexity

Context

- Based on my Master's Thesis
- Goal of project was to prove safety of two concurrent queues
- Success! – but not too interesting
- Today: Queue Specifications
- In particular, three different specifications
  - Sequential specification
  - Concurrent specification
    - Doesn't track queue contents
  - HOCAP-style specification
    - Tracks queue contents with added complexity
- Uses HeapLang, but should be mostly language-agnostic

Context

- Based on my Master's Thesis
- Goal of project was to prove safety of two concurrent queues
- Success! – but not too interesting
- Today: Queue Specifications
- In particular, three different specifications
  - Sequential specification
  - Concurrent specification
    - Doesn't track queue contents
  - HOCAP-style specification
    - Tracks queue contents with added complexity
- Uses HeapLang, but should be mostly language-agnostic
- Project was advised by Amin

Specifications for Queues

Informal Queue Specification

- 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 value v at end of queue and return Some v
  - If empty, [], return None

Specifications for Queues

Informal Queue Specification

- 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 value v at end of queue and return 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

Sequential Specification

Definition (Sequential Specification)

$\exists\, \mathsf{isQueue_S} : \mathit{Val} \to \mathit{List}\ \mathit{Val} \to \mathit{SeqQgnames} \to \mathsf{Prop}.$

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

Sequential Specification

Definition (Sequential Specification)

$\exists\, \text{isQueue}_S : \mathit{Val} \to \mathit{List}\ \mathit{Val} \to \mathit{SeqQgnames} \to \text{Prop}.$

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

- The proposition $\text{isQueue}_S(v_q, xs_v, G)$, states that value v_q represents the queue, which contains elements xs_v
- $G \in \mathit{SeqQgnames}$ 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!

Sequential Specification

Definition (Sequential Specification)

$\exists\, \text{isQueue}_S : \mathit{Val} \to \mathit{List}\ \mathit{Val} \to \mathit{SeqQgnames} \to \text{Prop}.$

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

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

- The proposition $\text{isQueue}_S(v_q, xs_v, G)$, states that value v_q represents the queue, which contains elements xs_v
- $G \in \mathit{SeqQgnames}$ 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!

Sequential Specification

Definition (Sequential Specification)

$$
\begin{aligned}
&\exists\, \mathsf{isQueue_S} : \mathit{Val} \to \mathit{List}\ \mathit{Val} \to \mathit{SeqQgnames} \to \mathsf{Prop}.\\
&\quad \{\mathsf{True}\}\ \mathsf{initialize}\ ()\ \{v_q.\exists G.\ \mathsf{isQueue_S}(v_q, [], G)\}\\
\wedge\ &\quad \forall v_q, v, xs_v, G.\ \{\mathsf{isQueue_S}(v_q, xs_v, G)\}\ \mathsf{enqueue}\ \ v_q\ \ v\ \{w.\,\mathsf{isQueue_S}(v_q, (v :: xs_v), G)\}\\
\wedge\ &\quad \forall v_q, xs_v, G.\ \{\mathsf{isQueue_S}(v_q, xs_v, G)\}\\
&\qquad\qquad \mathsf{dequeue}\ \ v_q\\
&\qquad\qquad \left\{ w.\ \begin{array}{l} (xs_v = [] * w = \mathsf{None} * \mathsf{isQueue_S}(v_q, xs_v, G)) \vee \\ (\exists v, xs_v'.\ xs_v = xs_v' +\!\!+ [v] * w = \mathsf{Some}\, v * \mathsf{isQueue_S}(v_q, xs_v', G)) \end{array} \right\}
\end{aligned}
$$

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

Concurrent Specification

- To support concurrent clients, we shall require the queue predicate be persistent
- Tracking the contents of queue in the way that the sequential specification did doesn't work
- Threads will start disagreeing on contents of queue, as they have only local view of contents
- Give up on tracking contents for now
- Instead, promise that all elements satisfy client-defined predicate, Ψ

Definition (Concurrent Specification)

$\exists\, \mathsf{isQueue_C} : (\mathit{Val} \to \mathsf{Prop}) \to \mathit{Val} \to \mathit{ConcQgnames} \to \mathsf{Prop}.$

$\forall \Psi : \mathit{Val} \to \mathsf{Prop}.$

$\quad \forall v_q, G.\ \mathsf{isQueue_C}(\Psi, v_q, G) \implies \Box\, \mathsf{isQueue_C}(\Psi, v_q, G)$

Concurrent Specification

- To support concurrent clients, we shall require the queue predicate be persistent
- Tracking the contents of queue in the way that the sequential specification did doesn't work
- Threads will start disagreeing on contents of queue, as they have only local view of contents
- Give up on tracking contents for now
- Instead, promise that all elements satisfy client-defined predicate, Ψ

Definition (Concurrent Specification)

$$\exists\, \mathsf{isQueue_C} : (\mathit{Val} \to \mathsf{Prop}) \to \mathit{Val} \to \mathit{ConcQgnames} \to \mathsf{Prop}.$$

$$\forall \Psi : \mathit{Val} \to \mathsf{Prop}.$$

$$\quad \forall v_q, G.\ \mathsf{isQueue_C}(\Psi, v_q, G) \implies \Box\, \mathsf{isQueue_C}(\Psi, v_q, G)$$

$$\wedge \quad \{\mathsf{True}\}\ \mathsf{initialize}\ ()\ \{v_q.\exists G.\ \mathsf{isQueue_C}(\Psi, v_q, G)\}$$

Concurrent Specification

- To support concurrent clients, we shall require the queue predicate be persistent
- Tracking the contents of queue in the way that the sequential specification did doesn't work
- Threads will start disagreeing on contents of queue, as they have only local view of contents
- Give up on tracking contents for now
- Instead, promise that all elements satisfy client-defined predicate, Ψ

Definition (Concurrent Specification)

$$\exists\, \mathsf{isQueue_C} : (\mathit{Val} \to \mathsf{Prop}) \to \mathit{Val} \to \mathit{ConcQgnames} \to \mathsf{Prop}.$$

$$\forall \Psi : \mathit{Val} \to \mathsf{Prop}.$$

$$\quad \forall v_q, G.\ \mathsf{isQueue_C}(\Psi, v_q, G) \implies \Box\, \mathsf{isQueue_C}(\Psi, v_q, G)$$

$$\wedge \quad \{\mathsf{True}\}\ \mathsf{initialize}\ ()\ \{v_q.\exists G.\ \mathsf{isQueue_C}(\Psi, v_q, G)\}$$

$$\wedge \quad \forall v_q, v, G.\ \{\mathsf{isQueue_C}(\Psi, v_q, G) * \Psi(v)\}\ \mathsf{enqueue}\ \ v_q\ \ v\ \{w.\mathsf{True}\}$$

Concurrent Specification

- To support concurrent clients, we shall require the queue predicate be persistent
- Tracking the contents of queue in the way that the sequential specification did doesn't work
- Threads will start disagreeing on contents of queue, as they have only local view of contents
- Give up on tracking contents for now
- Instead, promise that all elements satisfy client-defined predicate, Ψ

Definition (Concurrent Specification)

$$\exists\, \mathsf{isQueue_C} : (\mathit{Val} \to \mathsf{Prop}) \to \mathit{Val} \to \mathit{ConcQgnames} \to \mathsf{Prop}.$$

$$\forall \Psi : \mathit{Val} \to \mathsf{Prop}.$$

$$\begin{aligned}
&\forall v_q, G.\ \mathsf{isQueue_C}(\Psi, v_q, G) \implies \Box\, \mathsf{isQueue_C}(\Psi, v_q, G) \\
\wedge\ &\{\mathsf{True}\}\ \mathsf{initialize}\ ()\ \{v_q.\exists G.\ \mathsf{isQueue_C}(\Psi, v_q, G)\} \\
\wedge\ &\forall v_q, v, G.\ \{\mathsf{isQueue_C}(\Psi, v_q, G) * \Psi(v)\}\ \mathsf{enqueue}\ \ v_q\ \ v\ \{w.\mathsf{True}\} \\
\wedge\ &\forall v_q, G.\ \{\mathsf{isQueue_C}(\Psi, v_q, G)\}\ \mathsf{dequeue}\ \ v_q\ \{w.w = \mathsf{None} \vee (\exists v.\ w = \mathsf{Some}\, v * \Psi(v))\}
\end{aligned}$$

HOCAP-style Specification - Abstract State RA

- We will need a construction to allow clients to track contents of queue

HOCAP-style Specification - Abstract State RA

- We will need a construction to allow clients to track contents of queue
- Idea: have two "views" of the abstract state of the queue

Authoritative view

$\gamma \Mapsto_\bullet xs_v$

Owned by queue

Fragmental view

$\gamma \Mapsto_\circ xs_v$

Owned by client

HOCAP-style Specification - Abstract State RA

- We will need a construction to allow clients to track contents of queue
- Idea: have two "views" of the abstract state of the queue

Authoritative view	**Fragmental view**
$\gamma \Mapsto_\bullet xs_v$	$\gamma \Mapsto_\circ xs_v$
Owned by queue	Owned by client

- Construction ensures:
  - authoritative and fragmental views always agree on abstract state of queue
  - views can only be updated in unison
- Implemented using the resource algebra: $\textsc{Auth}((\textsc{Frac} \times \textsc{Ag}(\mathit{List}\ \mathit{Val}))^?)$
- The desirables are captured by the following lemmas

Lemmas on the Abstract State RA

$$\vdash \Mapsto \exists \gamma.\ \gamma \Mapsto_\bullet xs_v * \gamma \Mapsto_\circ xs_v \qquad \text{(Abstract State Alloc)}$$

$$\gamma \Mapsto_\bullet xs'_v * \gamma \Mapsto_\circ xs_v \vdash xs_v = xs'_v \qquad \text{(Abstract State Agree)}$$

$$\gamma \Mapsto_\bullet xs'_v * \gamma \Mapsto_\circ xs_v \Rrightarrow \gamma \Mapsto_\bullet xs''_v * \gamma \Mapsto_\circ xs''_v \qquad \text{(Abstract State Update)}$$

HOCAP-style Specification

- Post-condition of initialize specification gives fragmental view to clients
- Hoare triples for enqueue and dequeue are conditioned on view-shifts
- Clients must show that they can supply the fragmental view, so that the abstract (and concrete) state can be updated
- View-shifts and Hoare-triples parametrised by predicates P and Q
  - Client might have resources that need to be updated as a result of enqueue/dequeue
  - P is the clients resources before enqueue/dequeue and Q the resources after

Definition (HOCAP Specification)

$\exists$ isQueue : $Val \to Qgnames \to$ Prop.

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

HOCAP-style Specification

- Post-condition of initialize specification gives fragmental view to clients
- Hoare triples for enqueue and dequeue are conditioned on view-shifts
- Clients must show that they can supply the fragmental view, so that the abstract (and concrete) state can be updated
- View-shifts and Hoare-triples parametrised by predicates P and Q
  - Client might have resources that need to be updated as a result of enqueue/dequeue
  - P is the clients resources before enqueue/dequeue and Q the resources after

Definition (HOCAP Specification)

$\exists\, \text{isQueue} : \mathit{Val} \to \mathit{Qgnames} \to \text{Prop}.$

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

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

HOCAP-style Specification

- Post-condition of initialize specification gives fragmental view to clients
- Hoare triples for enqueue and dequeue are conditioned on view-shifts
- Clients must show that they can supply the fragmental view, so that the abstract (and concrete) state can be updated
- View-shifts and Hoare-triples parametrised by predicates P and Q
  - Client might have resources that need to be updated as a result of enqueue/dequeue
  - P is the clients resources before enqueue/dequeue and Q the resources after

Definition (HOCAP Specification)

$\exists\, \text{isQueue} : \mathit{Val} \to \mathit{Qgnames} \to \text{Prop}.$

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

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

$$\wedge \quad \forall v_q, v, G, P, Q.\ \left(\forall xs_v.\ G.\gamma_{\text{Abst}} \Mapsto_\bullet xs_v * P \Rrightarrow_{\mathcal{E}\setminus\mathcal{N}.i\uparrow} \triangleright\, G.\gamma_{\text{Abst}} \Mapsto_\bullet (v :: xs_v) * Q\right) \mathrel{-\!\!*}$$
$$\{\text{isQueue}(v_q, G) * P\}\ \text{enqueue}\ v_q\ v\ \{w.Q\}$$

HOCAP-style Specification

- Post-condition of initialize specification gives fragmental view to clients
- Hoare triples for enqueue and dequeue are conditioned on view-shifts
- Clients must show that they can supply the fragmental view, so that the abstract (and concrete) state can be updated
- View-shifts and Hoare-triples parametrised by predicates P and Q
  - Client might have resources that need to be updated as a result of enqueue/dequeue
  - P is the clients resources before enqueue/dequeue and Q the resources after

Definition (HOCAP Specification)

$$
\begin{aligned}
&\exists\, \mathsf{isQueue} : \mathit{Val} \to \mathit{Qgnames} \to \mathsf{Prop}.\\
&\quad \forall v_q, G.\ \mathsf{isQueue}(v_q, G) \implies \Box\, \mathsf{isQueue}(v_q, G)\\
&\wedge \quad \{\mathsf{True}\}\ \mathsf{initialize}\ ()\ \{v_q.\exists G.\ \mathsf{isQueue}(v_q, G) * G.\gamma_{\mathrm{Abst}} \mapsto_{\circ} []\}\\
&\wedge \quad \forall v_q, v, G, P, Q.\ \left(\forall xs_v.\ G.\gamma_{\mathrm{Abst}} \mapsto_{\bullet} xs_v * P \Rrightarrow_{\mathcal{E}\setminus\mathcal{N}.i\uparrow} \triangleright\, G.\gamma_{\mathrm{Abst}} \mapsto_{\bullet} (v :: xs_v) * Q\right) \mathrel{-\!\!*}\\
&\qquad \{\mathsf{isQueue}(v_q, G) * P\}\ \mathsf{enqueue}\ v_q\ v\ \{w.Q\}\\
&\wedge \quad \forall v_q, G, P, Q.\\
&\qquad \left(\forall xs_v.\ G.\gamma_{\mathrm{Abst}} \mapsto_{\bullet} xs_v * P \Rrightarrow_{\mathcal{E}\setminus\mathcal{N}.i\uparrow} \triangleright \left(\begin{array}{l}(xs_v = [] * G.\gamma_{\mathrm{Abst}} \mapsto_{\bullet} xs_v * Q(\mathsf{None}))\\ \vee \left(\begin{array}{l}\exists v, xs'_v.\ xs_v = xs'_v +\!\!+ [v] *\\ G.\gamma_{\mathrm{Abst}} \mapsto_{\bullet} xs'_v * Q(\mathsf{Some}\ v)\end{array}\right)\end{array}\right)\right) \mathrel{-\!\!*}\\
&\qquad \{\mathsf{isQueue}(v_q, G) * P\}\ \mathsf{dequeue}\ v_q\ \{w.Q(w)\}
\end{aligned}
$$

Queue Client - A PoC Client

- Add two numbers after having two threads enqueue and subsequently dequeue them

```
unwrap w ≜ match w with None ⇒ () () | Some v ⇒ v end

enqdeq v_q c ≜ enqueue v_q c; unwrap(dequeue v_q)

queueAdd a b ≜
  let v_q = initialize () in
  let p = (enqdeq v_q a) || (enqdeq v_q b) in
  fst p + snd p
```

Queue Client - A PoC Client

- Add two numbers after having two threads enqueue and subsequently dequeue them
- Idea: a minimal client complex enough to require HOCAP-style specification

---

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

enqdeq v_q $c \triangleq$ enqueue v_q c; unwrap(dequeue v_q)

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

Queue Client - A PoC Client

- Add two numbers after having two threads enqueue and subsequently dequeue them
- Idea: a minimal client complex enough to require HOCAP-style specification
- Uses parallel composition, so sequential specification insufficient

---

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

enqdeq v_q $c \triangleq$ enqueue v_q c; unwrap(dequeue v_q)

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

Queue Client - A PoC Client

- Add two numbers after having two threads enqueue and subsequently dequeue them
- Idea: a minimal client complex enough to require HOCAP-style specification
- Uses parallel composition, so sequential specification insufficient
- Relies on dequeues not returning None, so concurrent specification insufficient

---

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

enqdeq v_q $c \triangleq$ enqueue v_q c; unwrap(dequeue v_q)

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

Queue Client - A PoC Client

- Add two numbers after having two threads enqueue and subsequently dequeue them
- Idea: a minimal client complex enough to require HOCAP-style specification
- Uses parallel composition, so sequential specification insufficient
- Relies on dequeues not returning None, so concurrent specification insufficient
- HOCAP-style specification supports consistency and tracks 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 $v_q\ c \triangleq$ enqueue $v_q\ c$; unwrap(dequeue v_q)

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

Queue Client - A PoC Client (continued)

Lemma (QueueAdd Specification)

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

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)

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

- When using the HOCAP-style Queue specification to prove the above, we will make P and Q talk about the tokens.
- E.g for enqueue:
  - $P = \text{TokA}\ Ga \vee \text{TokB}\ Ga$
  - $Q = \text{TokD1}\ Ga \vee \text{TokD2}\ Ga$