

The Best Queue Specifications You Will Ever See today probably

Mathias Pedersen

Aarhus University

November 2025

..../thesis/logo-eps-converted-to.pdf

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**, $xs_v \text{ ++ } [v]$, 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)

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

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

Sequential Specification

Definition (Sequential Specification)

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

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

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

Sequential Specification

Definition (Sequential Specification)

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

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

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

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

Sequential Specification

Definition (Sequential Specification)

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

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

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

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

dequeue v_q

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

- The proposition $\text{isQueues}_S(v_q, xs_v, G)$, states that value v_q represents the queue, which contains elements xs_v
- $G \in SeqQgnames$ is a collection of ghost names (depends on specific queue)
- Specification consists of three Hoare triples – one for each queue function
- Important:** isQueues_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)

$$\exists \text{isQueue}_C : (\text{Val} \rightarrow \text{Prop}) \rightarrow \text{Val} \rightarrow \text{ConcQnames} \rightarrow \text{Prop}.$$
$$\forall \Psi : \text{Val} \rightarrow \text{Prop}.$$
$$\forall v_q, G. \text{ isQueue}_C(\Psi, v_q, G) \implies \square \text{ isQueue}_C(\Psi, v_q, G)$$

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)

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

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

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

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

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)

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

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

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

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

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

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)

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

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

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

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

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

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

HOCP-style Specification - Abstract State RA

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

HOCP-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

$$\gamma \Rightarrow_{\bullet} xs_v$$

Owned by queue

Fragmental view

$$\gamma \Rightarrow_{\circ} xs_v$$

Owned by client

HOCP-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

$\gamma \Rightarrow_{\bullet} xs_v$
Owned by queue

Fragmental view

$\gamma \Rightarrow_{\circ} xs_v$
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**: $AUTH((FRAC \times AG(List\ Val))^?)$
- The **desirables** are captured by the following **lemmas**

Lemmas on the Abstract State RA

$$\vdash \Rightarrow \exists \gamma. \gamma \Rightarrow_{\bullet} xs_v * \gamma \Rightarrow_{\circ} xs_v \quad (\text{Abstract State Alloc})$$

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

$$\gamma \Rightarrow_{\bullet} xs'_v * \gamma \Rightarrow_{\circ} xs_v \Rightarrow \gamma \Rightarrow_{\bullet} xs''_v * \gamma \Rightarrow_{\circ} xs''_v \quad (\text{Abstract State Update})$$

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)

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

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

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)

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

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

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

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)

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

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

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

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

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)

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

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

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

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

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

Queue Client - A PoC Client

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

```
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

- 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 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

- 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 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

- 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 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

- 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 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)

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

Queue Client - A PoC Client (continued)

Lemma (QueueAdd Specification)

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

- 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)

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

- 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:
 - $P = \text{TokA } Ga \vee \text{TokB } Ga$
 - $Q = \text{TokD1 } Ga \vee \text{TokD2 } Ga$