Sequential Specification

Definition (Sequential Specification)

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

- The proposition $\mathrm{isQueue}_S(v_q, xs_v, G)$, 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: $\mathrm{isQueue}_S$ *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)

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

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

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

- 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)

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

$$
\begin{aligned}
& \{\mathrm{True}\}\ \mathrm{initialize}\ ()\ \{v_q. \exists G.\ \mathrm{isQueue}_S(v_q, [], G)\} \\
\wedge\quad & \forall v_q, v, xs_v, G.\ \{\mathrm{isQueue}_S(v_q, xs_v, G)\}\ \mathrm{enqueue}\ \ v_q\ \ v\ \{w.\ \mathrm{isQueue}_S(v_q, (v :: xs_v), G)\} \\
\wedge\quad & \forall v_q, xs_v, G.\ \{\mathrm{isQueue}_S(v_q, xs_v, G)\} \\
& \qquad\qquad \mathrm{dequeue}\ \ v_q \\
& \qquad \left\{ w.\ \begin{array}{l} (xs_v = [] * w = \mathrm{None} * \mathrm{isQueue}_S(v_q, xs_v, G)) \vee \\ (\exists v, xs'_v.\ xs_v = xs'_v +\!\!+ [v] * w = \mathrm{Some}\ v * \mathrm{isQueue}_S(v_q, xs'_v, G)) \end{array} \right\}
\end{aligned}
$$

- The proposition $\mathrm{isQueue}_S(v_q, xs_v, G)$, 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: $\mathrm{isQueue}_S$ *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)

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

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

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

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

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)

$$\exists\, \mathsf{isQueue}_C : (Val \to \mathsf{Prop}) \to Val \to ConcQgnames \to \mathsf{Prop}.$$

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

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

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

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

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)

$$\exists\,\mathrm{isQueue}_C : (\mathit{Val} \to \mathrm{Prop}) \to \mathit{Val} \to \mathit{ConcQgnames} \to \mathrm{Prop}.$$

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

$$\begin{aligned}
& \forall v_q, G.\ \mathrm{isQueue}_C(\Psi, v_q, G) \implies \Box\,\mathrm{isQueue}_C(\Psi, v_q, G) \\
\land\quad & \{\mathrm{True}\}\ \mathrm{initialize}\ ()\ \{v_q.\exists G.\ \mathrm{isQueue}_C(\Psi, v_q, G)\} \\
\land\quad & \forall v_q, v, G.\ \{\mathrm{isQueue}_C(\Psi, v_q, G) * \Psi(v)\}\ \mathrm{enqueue}\ \ v_q\ \ v\ \{w.\mathrm{True}\} \\
\land\quad & \forall v_q, G.\ \{\mathrm{isQueue}_C(\Psi, v_q, G)\}\ \mathrm{dequeue}\ \ v_q\ \{w.w = \mathrm{None} \lor (\exists v.\ w = \mathrm{Some}\, v * \Psi(v))\}
\end{aligned}$$

Queue Client - A PoC Client

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

$$\text{unwrap}\ \ w \triangleq \textsf{match}\ w\ \textsf{with}\ \text{None} \Rightarrow ()\ ()\mid \text{Some}\ v \Rightarrow v\ \textsf{end}$$

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

$$\begin{aligned}&\text{queueAdd}\ \ a\ b \triangleq\\ &\quad \textsf{let}\ v_q = \text{initialize}\ ()\ \textsf{in}\\ &\quad \textsf{let}\ p = (\text{enqdeq}\ \ v_q\ \ a) \mid\mid (\text{enqdeq}\ \ v_q\ \ b)\ \textsf{in}\\ &\quad \textsf{fst}\ p + \textsf{snd}\ p\end{aligned}$$

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

$$\mathsf{unwrap}\;\; w \triangleq \mathbf{match}\; w \;\mathbf{with}\; \mathsf{None} \Rightarrow ()\;() \mid \mathsf{Some}\, v \Rightarrow v \;\mathbf{end}$$

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

$$\begin{aligned}&\mathsf{queueAdd}\;\; a\; b \triangleq\\ &\quad \mathbf{let}\; v_q = \mathsf{initialize}\;()\;\mathbf{in}\\ &\quad \mathbf{let}\; p = (\mathsf{enqdeq}\;\; v_q\;\; a) \,||\, (\mathsf{enqdeq}\;\; v_q\;\; b)\;\mathbf{in}\\ &\quad \mathbf{fst}\; p + \mathbf{snd}\; p\end{aligned}$$

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

$$\text{unwrap } w \triangleq \textbf{match } w \textbf{ with } \text{None} \Rightarrow ()\ () \mid \text{Some } v \Rightarrow v \textbf{ end}$$

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

$$\begin{aligned}
&\text{queueAdd } a\ b \triangleq \\
&\quad \textbf{let } v_q = \text{initialize } ()\ \textbf{in} \\
&\quad \textbf{let } p = (\text{enqdeq } v_q\ a) \mid\mid (\text{enqdeq } v_q\ b)\ \textbf{in} \\
&\quad \textbf{fst } p + \textbf{snd } p
\end{aligned}$$

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

$$\text{unwrap}\ \ w \triangleq \textbf{match}\ w\ \textbf{with}\ \text{None} \Rightarrow ()\ ()\ |\ \text{Some}\ v \Rightarrow v\ \textbf{end}$$

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

$$\begin{aligned}&\text{queueAdd}\ \ a\ b \triangleq\\ &\quad \textbf{let}\ v_q = \text{initialize}\ ()\ \textbf{in}\\ &\quad \textbf{let}\ p = (\text{enqdeq}\ \ v_q\ \ a)\ ||\ (\text{enqdeq}\ \ v_q\ \ b)\ \textbf{in}\\ &\quad \textbf{fst}\ p + \textbf{snd}\ p\end{aligned}$$

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

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

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

$$\begin{aligned}&\mathsf{queueAdd}\ \ a\ b \triangleq\\ &\quad \mathsf{let}\ v_q = \mathsf{initialize}\ ()\ \mathsf{in}\\ &\quad \mathsf{let}\ p = (\mathsf{enqdeq}\ \ v_q\ \ a)\ ||\ (\mathsf{enqdeq}\ \ v_q\ \ b)\ \mathsf{in}\\ &\quad \mathsf{fst}\ p + \mathsf{snd}\ p\end{aligned}$$

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)

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