Functional correctness specifications for concurrent data structures

Logical Atomicity in Iris

Ralf Jung, MIT, USA

ASL 2022

Functional correctness of a queue

$\{ \quad \}$

$\texttt{enqueue}(q, v)$

$\{ \quad \}$

$\{ \quad \}$

$\texttt{dequeue}(q)$

$\{ \quad \}$

Functional correctness of a queue

$\text{Queue}(q, l)$:
q points to a queue storing list of values l

$$\{\text{Queue}(q, l)\}$$
$$\quad \texttt{enqueue}(q, v)$$
$$\{\text{Queue}(q, l +\!\!+ [v])\}$$

Functional correctness of a queue

$\text{Queue}(q, l)$:
q points to a queue storing list of values l

$$\{\text{Queue}(q, l)\}\quad \texttt{enqueue}(q, v)\quad \{\text{Queue}(q, l + [v])\}$$

$$\{\text{Queue}(q, v :: l)\}\quad \texttt{dequeue}(q)\quad \{w.\, w = v * \text{Queue}(q, l)\}$$

Functional correctness of a queue

$\text{Queue}(q, l)$:
q points to a queue storing list of values l

$$\{\text{Queue}(q, l)\} \quad \texttt{enqueue}(q, v) \quad \{\text{Queue}(q, l +\!\!+ [v])\}$$

$$\{\text{Queue}(q, v :: l)\} \quad \texttt{dequeue}(q) \quad \{w.\, w = v * \text{Queue}(q, l)\}$$

Binder for return value.

Specifying a sequential queue
is easy.

What about a concurrent queue?

Reuse the sequential specification?

$$\{\mathrm{Queue}(q, l)\}$$
$$\quad \texttt{enqueue}(q, v)$$
$$\{\mathrm{Queue}(q, l +\!\!+ [v])\}$$

$$\{\mathrm{Queue}(q, v :: l)\}$$
$$\quad \texttt{dequeue}(q)$$
$$\{w.\, w = v * \mathrm{Queue}(q, l)\}$$

Reuse the sequential specification?

Precondition consumes queue ownership. Impossible to call `enqueue` or `dequeue` concurrently with other queue operations.

$$\{\mathrm{Queue}(q, l)\}\ \ \mathtt{enqueue}(q, v)\ \ \{\mathrm{Queue}(q, l +\!\!+ [v])\}$$

$$\{\mathrm{Queue}(q, v :: l)\}\ \ \mathtt{dequeue}(q)\ \ \{w.\ w = v * \mathrm{Queue}(q, l)\}$$

Reuse the sequential specification?

Precondition consumes queue ownership.

Even a non-thread-safe queue would satisfy this specification.

$\{\text{Queue}(q, l ++ [v])\}$ $\{w.\, w = v * \text{Queue}(q, l)\}$

Common solution:

- Use contextual refinement as spec
- Use linearizability to prove it

$$\frac{\frac{\texttt{enqueue} \precsim \texttt{seq\_enqueue\_locked}}{???}}{\{P\}\ \texttt{client}[\texttt{enqueue}]\ \{Q\}}$$

$$\frac{\frac{\texttt{enqueue} \precsim \texttt{seq\_enqueue\_locked}}{???}}{\{P\}\ \texttt{client[enqueue]}\ \{Q\}}$$

What about heap accesses?

These look just as sequential:

$$\{\ell \mapsto v\}\ !\,\ell\ \{w.\ w = v * \ell \mapsto v\}$$

$$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$$

What about heap accesses?

These look just as sequential:

$$\{\ell \mapsto v\} \, ! \, \ell \, \{w. \, w = v * \ell \mapsto v\}$$

$$\{\ell \mapsto v\} \, \ell \leftarrow w \, \{\ell \mapsto w\}$$

Concurrent use is possible thanks to the invariant rule!

What about heap accesses?

Invariant Rule

$$\frac{\{P * I\}\ \ell \leftarrow w\ \{Q * I\}}{\boxed{I} \vdash \{P\}\ \ell \leftarrow w\ \{Q\}}$$

(consider $I \triangleq \exists v.\ \ell \mapsto v$)

Invariant rule!

What about heap accesses?

Invariant Rule

$$\frac{\{P * I\}\; e\; \{Q * I\} \qquad \text{phy\_atomic(e)}}{\boxed{I} \vdash \{P\}\; e\; \{Q\}}$$

Invariant rule!

An operation is atomic if we can open invariants around it.

An operation is atomic if we can open invariants around it.

Logical atomicity

lets us open invariants around non-physically-atomic operations.

Outline

1. How to specify and use basic logically atomic operations in Iris
2. Advanced logically atomic patterns: aborting, helping
3. Summary and case studies

Outline

1. How to specify and use basic logically atomic operations in Iris
2. Advanced logically atomic patterns: aborting, helping
3. Summary and case studies

Logically Atomic Queue

Logically Atomic Hoare triples inspired by TaDA

(Pinto, Dinsdale-Young, Gardner; 2014)

$$\langle l.\, \mathrm{Queue}(q, l) \rangle$$
$$\quad \mathtt{enqueue}(q, v)$$
$$\langle \mathrm{Queue}(q, l \mathbin{+\!\!+} [v]) \rangle$$

Logically Atomic Queue

Logically Atomic Hoare triples inspired by TaDA

(Pinto, Dinsdale-Young, Gardner; 2014)

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\quad \texttt{enqueue}(q, v)$$
$$\langle \mathsf{Queue}(q, l +\!\!+ [v])\rangle$$

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\quad \texttt{dequeue}(q)$$
$$\langle v.\ v = \mathsf{head}(l) * \mathsf{Queue}(q, \mathsf{tail}(l))\rangle$$

Logically Atomic Queue

Logically Atomic Hoare triples inspired by TaDA
(Pinto, Dinsdale-Young, Gardner; 2014)

l is picked at the linearization point

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\quad \mathtt{enqueue}(q, v)$$
$$\langle \mathsf{Queue}(q, l +\!\!+ [v])\rangle$$

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\quad \mathtt{dequeue}(q)$$
$$\langle v.\ v = \mathsf{head}(l) * \mathsf{Queue}(q, \mathsf{tail}(l))\rangle$$

Logically Atomic Queue

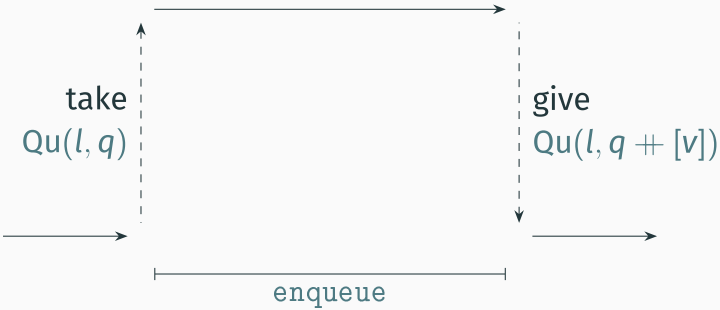
Logically Atomic Hoare triples inspired by TaDA

(Pinto, Dinsdale-Young, Gardner; 2014)

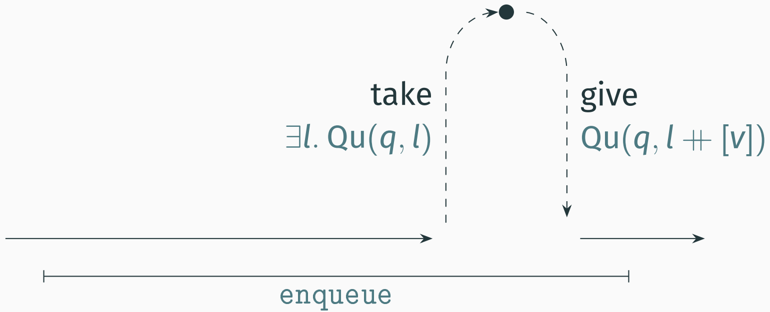
$$\mathrm{IsQueue}(q) \vdash \langle l.\, \mathrm{Queue}(q, l) \rangle \; \texttt{enqueue}(q, v) \; \langle \mathrm{Queue}(q, l +\!\!+ [v]) \rangle$$

$$\mathrm{IsQueue}(q) \vdash \langle l.\, \mathrm{Queue}(q, l) \rangle \; \texttt{dequeue}(q) \; \langle v.\, v = \mathrm{head}(l) * \mathrm{Queue}(q, \mathrm{tail}(l)) \rangle$$

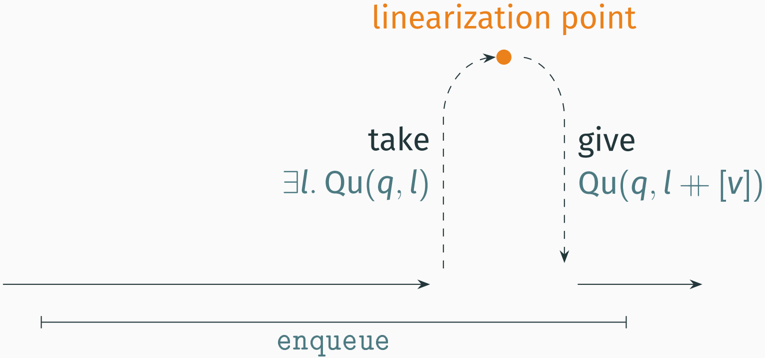
$$\forall l.\ \{\mathrm{Qu}(q, l)\}\ \texttt{enqueue}(q, v)\ \{\mathrm{Qu}(q, l +\!\!+ [v])\}$$



$\mathsf{IsQu}(q) \vdash \langle l.\ \mathsf{Qu}(q, l) \rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l \mathbin{+\!\!+} [v]) \rangle$

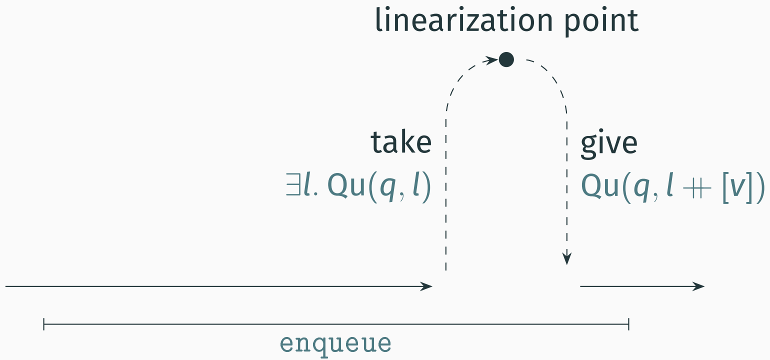


$$\mathsf{IsQu}(q) \vdash \langle l.\ \mathsf{Qu}(q, l)\rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l \mathbin{+\!\!+} [v])\rangle$$



$$\mathsf{IsQu}(q) \vdash \langle l.\, \mathsf{Qu}(q, l)\rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l \mathbin{+\!\!+} [v])\rangle \triangleq$$

$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$



Interlude: Masks and view shifts in Iris

$$\frac{\{P * I\}\; e\; \{Q * I\} \qquad \text{phy\_atomic(e)}}{\boxed{I} \;\vdash \{P\}\; e\; \{Q\}}$$

Interlude: Masks and view shifts in Iris

$$\frac{\{P * I\}\; e\; \{Q * I\}_{\mathcal{E}\setminus\mathcal{N}} \qquad \mathcal{N} \subseteq \mathcal{E} \qquad \text{phy\_atomic(e)}}{\boxed{I}^{\mathcal{N}} \vdash \{P\}\; e\; \{Q\}_{\mathcal{E}}}$$

Namespace of the invariant

Mask of the Hoare triple

Interlude: Masks and view shifts in Iris

View shift: "linear ghost step" (without code)

$$\frac{P * I \equiv\!\!\ast_{\mathcal{E}\setminus\mathcal{N}} Q * I \qquad \mathcal{N} \subseteq \mathcal{E}}{\boxed{I}^{\mathcal{N}} \vdash P \equiv\!\!\ast_{\mathcal{E}} Q}$$

Namespace of the invariant

Mask of the view shift

Interlude: Masks and view shifts in Iris

View shift: "linear ghost step" (without code)

$$\frac{P * I \Rrightarrow_{\mathcal{E} \setminus \mathcal{N}} Q * I \qquad \mathcal{N} \subseteq \mathcal{E}}{\boxed{I}^{\mathcal{N}} \vdash P \Rrightarrow_{\mathcal{E}} Q}$$

$P_1 \Rrightarrow_{\mathcal{E}} P_2$: view shift from P_1 to P_2
using only invariants $\mathcal{N} \subseteq \mathcal{E}$

Interlude: Masks and view shifts in Iris

View shift: "linear ghost step" (without code)

$$\frac{P * I \Rrightarrow_{\mathcal{E}\setminus\mathcal{N}} Q * I \qquad \mathcal{N} \subseteq \mathcal{E}}{\boxed{I}^{\mathcal{N}} \vdash P \Rrightarrow_{\mathcal{E}} Q}$$

$P_1 \Rrightarrow_{\mathcal{E}} P_2$: view shift from P_1 to P_2 using only invariants $\mathcal{N} \subseteq \mathcal{E}$

$P_1 \;{}^{\mathcal{E}_1}{\Rrightarrow}^{\mathcal{E}_2}\; P_2$: mask-changing view shift from P_1 to P_2

Interlude: Mask-changing view shifts

$$\frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash P \; {}^{\mathcal{E}}{\equiv\!\!\!\ast}^{\mathcal{E}\setminus\mathcal{N}} \; P * I}$$

Interlude: Mask-changing view shifts

$$\frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash P \; {}^{\mathcal{E}}{\Rrightarrow}^{\mathcal{E} \setminus \mathcal{N}} \; P * I}$$

$$\frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash Q * I \; {}^{\mathcal{E} \setminus \mathcal{N}}{\Rrightarrow}^{\mathcal{E}} \; Q}$$

Interlude: Mask-changing view shifts

$$\frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash P \;{}^{\mathcal{E}}{\Rrightarrow}\!\!\ast^{\mathcal{E}\setminus\mathcal{N}}\; P * I} \qquad \frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash Q * I \;{}^{\mathcal{E}\setminus\mathcal{N}}{\Rrightarrow}\!\!\ast^{\mathcal{E}}\; Q}$$

$$\frac{\begin{array}{c}\text{phy\_atomic(e)} \\ P \;{}^{\mathcal{E}}{\Rrightarrow}\!\!\ast^{\mathcal{E}'}\; P' \qquad \{P'\}\; e \;\{Q'\}_{\mathcal{E}'} \qquad Q' \;{}^{\mathcal{E}'}{\Rrightarrow}\!\!\ast^{\mathcal{E}}\; Q\end{array}}{\{P\}\; e \;\{Q\}_{\mathcal{E}}}$$

Interlude: Mask-changing view shifts

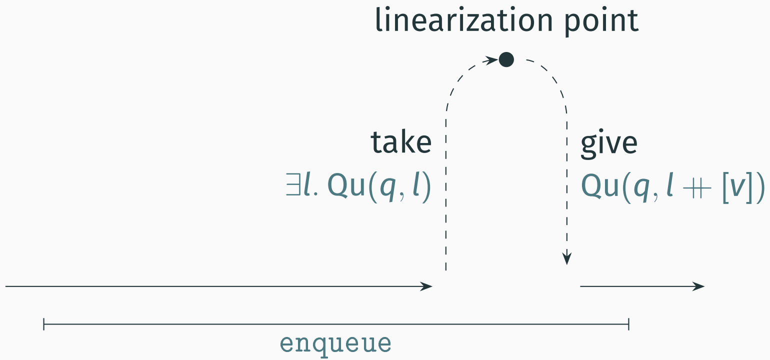
$$\frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash P \mathrel{{}^{\mathcal{E}}\!\!\Rrightarrow\!\!\ast^{\mathcal{E}\setminus\mathcal{N}}} P * I} \qquad \frac{\mathcal{E} \subseteq \mathcal{N}}{\boxed{I}^{\mathcal{N}} \vdash Q * I \mathrel{{}^{\mathcal{E}\setminus\mathcal{N}}\!\!\Rrightarrow\!\!\ast^{\mathcal{E}}} Q}$$

$$\frac{P \mathrel{{}^{\mathcal{E}}\!\!\Rrightarrow\!\!\ast^{\mathcal{E}'}} P' \qquad \begin{array}{c}\text{phy\_atomic(e)}\\ \{P'\}\, e\, \{Q'\}_{\mathcal{E}'}\end{array} \qquad Q' \mathrel{{}^{\mathcal{E}'}\!\!\Rrightarrow\!\!\ast^{\mathcal{E}}} Q}{\{P\}\, e\, \{Q\}_{\mathcal{E}}}$$

Together, these three rules imply the invariant open rule!

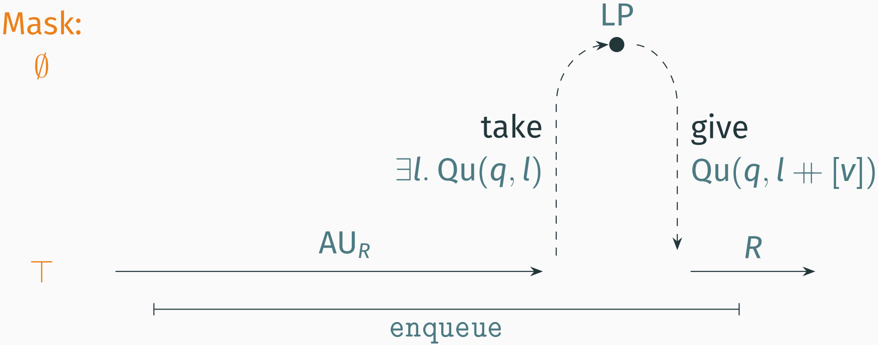
$$\mathsf{IsQu}(q) \vdash \langle l.\ \mathsf{Qu}(q, l)\rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l \mathbin{+\!\!+} [v])\rangle \triangleq$$

$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$



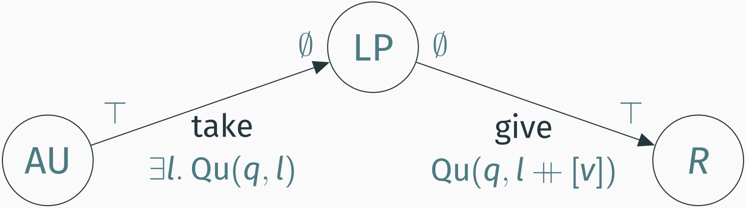
$$\langle l.\ \mathrm{Qu}(q, l) \rangle\ \mathtt{enqueue}(q, v)\ \langle \mathrm{Qu}(q, l +\!\!+ [v]) \rangle \triangleq$$

$$\forall R.\ \{\mathrm{AU}_R\}\ \mathtt{enqueue}(q, v)\ \{R\}$$



$$\langle l.\ \mathsf{Qu}(q, l)\rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l \mathbin{+\!\!+} [v])\rangle \triangleq$$
$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

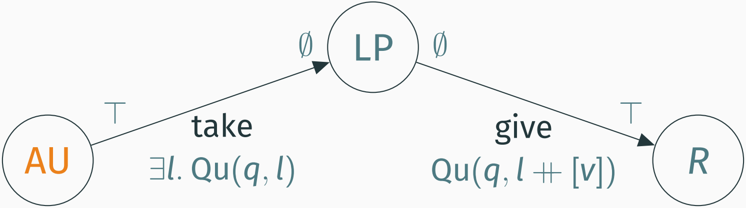
Meaning of atomic update AU_R:



$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$\mathsf{AU}_R \triangleq$

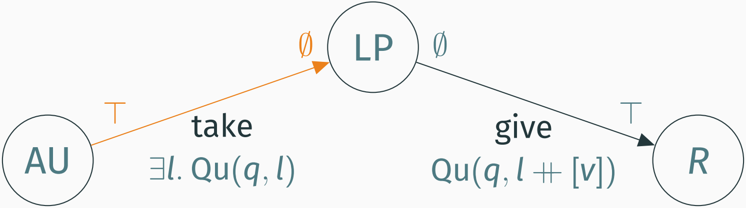
Meaning of atomic update AU_R:



$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}$$

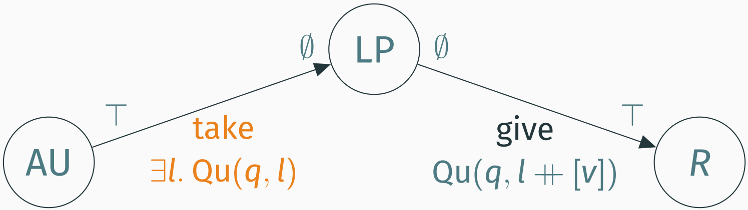
Meaning of atomic update AU_R:



$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l)$$

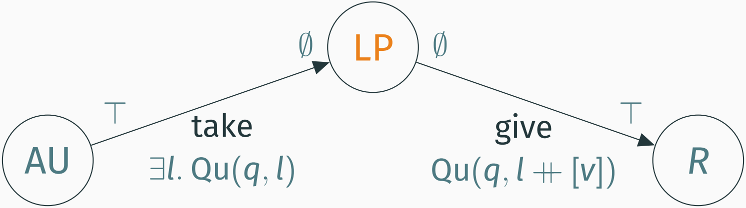
Meaning of atomic update AU_R:



$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) * \mathsf{LP}$$

Meaning of atomic update AU_R:

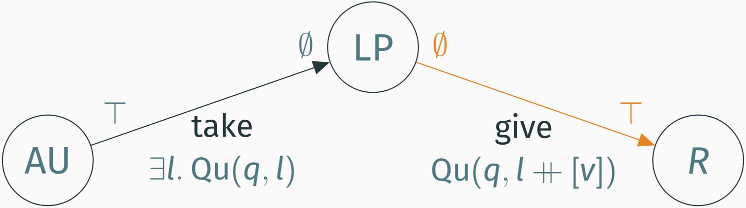


$$\forall R.\ \{\mathsf{AU}_R\}\ \mathtt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\Rrightarrow\!\!*^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) * \mathsf{LP}$$

$$\mathsf{LP} \triangleq \qquad\qquad\qquad {}^{\emptyset}\!\Rrightarrow\!\!*^{\top}$$

Meaning of atomic update AU_R:

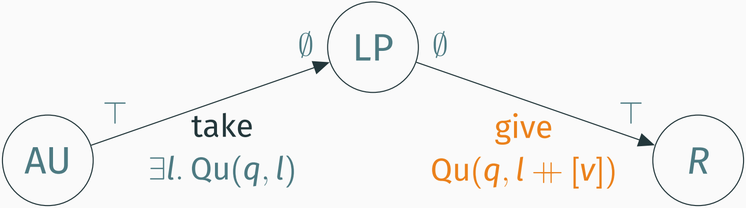


$$\forall R.\ \{\mathsf{AU}_R\}\ \mathtt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\equiv\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) \ast \mathsf{LP}$$

$$\mathsf{LP} \triangleq \mathsf{Qu}(q, l +\!\!+ [v])\ {}^{\emptyset}\!\!\equiv\!\!\ast^{\top}$$

Meaning of atomic update AU_R:

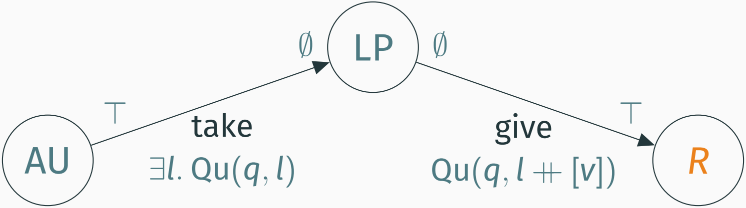


$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) * \mathsf{LP}$$

$$\mathsf{LP} \triangleq \mathsf{Qu}(q, l +\!\!+ [v])\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ R$$

Meaning of atomic update AU_R:

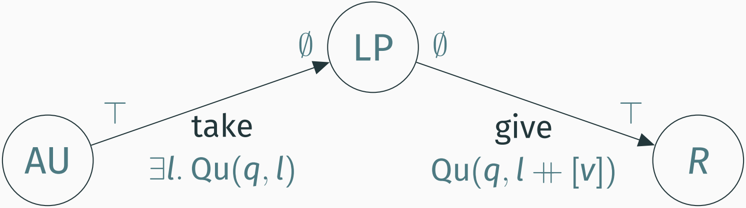


$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\equiv\!\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) * \mathsf{LP}$$

$$\mathsf{LP} \triangleq \mathsf{Qu}(q, l \mathbin{+\!\!+} [v])\ {}^{\emptyset}\!\!\equiv\!\!\!\ast^{\top}\ R$$

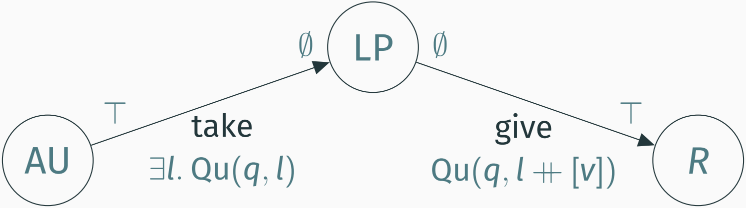
Meaning of atomic update AU_R:



$$\forall R.\ \{\mathrm{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

$$\mathrm{AU}_R \triangleq \mathrm{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathrm{Qu}(q, l) \ast \left(\mathrm{Qu}(q, l \mathbin{+\!\!+} [v])\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ R\right)$$

Meaning of atomic update AU_R:



$$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{enqueue}(q, v)\ \{R\}$$

Specification of logically atomic `enqueue`:

$$\mathsf{IsQu}(l) \vdash \langle l.\ \mathsf{Qu}(q, l)\rangle\ \texttt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l +\!\!+ [v])\rangle$$

$$\forall R.\ \{\mathsf{AU}_R\}\ \mathtt{enqueue}(q, v)\ \{R\}$$

Specification of logically atomic `enqueue`:

$$\mathsf{IsQu}(l) \vdash \langle l.\ \mathsf{Qu}(q, l)\rangle\ \mathtt{enqueue}(q, v)\ \langle \mathsf{Qu}(q, l +\!\!+ [v])\rangle$$

which expands to:

$$\mathsf{IsQu}(l) \vdash \forall R.\ \{\mathsf{AU}_R\}\ \mathtt{enqueue}(q, v)\ \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Qu}(q, l) \ast \left(\mathsf{Qu}(q, l +\!\!+ [v])\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ R\right)$$

How do we use a logically atomic triple?

Let's say we have a shared queue that contains only even numbers:

$$\exists l.\, \mathsf{Qu}(q, l) * \forall n \in l.\, \mathsf{even}(n)$$

How do we use a logically atomic triple?

Let's say we have a shared queue that contains only even numbers:

$$\boxed{\exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)}^{\mathcal{N}}$$

How do we use a logically atomic triple?

Let's say we have a shared queue that contains only even numbers:

$$\boxed{\exists l.\ \mathsf{Qu}(q, l) * \forall n \in l.\ \mathsf{even}(n)}^{\mathcal{N}}$$

How can we `enqueue` and `dequeue` on it?

`enqueue` on a shared queue

Assume: $\mathsf{IsQu}(q)$, $\boxed{\exists l.\, \mathsf{Qu}(q, l) * \forall n \in l.\, \mathsf{even}(n)}^{\mathcal{N}}$

Goal: $\{\mathsf{True}\}\ \texttt{enqueue}(q, 2)\ \{\mathsf{True}\}$

`enqueue` on a shared queue

Assume: $\mathsf{IsQu}(q)$, $\boxed{\exists l.\, \mathsf{Qu}(q, l) * \forall n \in l.\, \mathsf{even}(n)}^{\mathcal{N}}$

Goal: $\{\mathsf{True}\}\ \mathtt{enqueue}(q, 2)\ \{\mathsf{True}\}$

Remember that we have:

$$\mathsf{IsQu}(l) \vdash \forall R.\, \{\mathsf{AU}_R\}\ \mathtt{enqueue}(q, 2)\ \{R\}$$

So it suffices to show:

$$\boxed{\exists l.\, \mathsf{Qu}(q, l) * \forall n \in l.\, \mathsf{even}(n)}^{\mathcal{N}} \vdash \mathsf{AU}_{\mathsf{True}}$$

`enqueue` on a shared queue

Assume: $\boxed{\exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)}^{\mathcal{N}}$

Goal:

$\mathrm{AU}_{\mathrm{True}}$

`enqueue` on a shared queue

Assume: $\boxed{\exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)}^{\mathcal{N}}$

Goal:

$$\mathrm{True} \;{}^{\top}{\Rrightarrow\!\!*}^{\emptyset}\; \exists l.\, \mathrm{Qu}(q, l) * \left(\mathrm{Qu}(q, l +\!\!+ [2]) \;{}^{\emptyset}{\Rrightarrow\!\!*}^{\top}\; \mathrm{True}\right)$$

`enqueue` on a shared queue

Assume: $\boxed{\exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)}^{\mathcal{N}}$

Goal:

$$\mathrm{True}\ {}^{\top}\!\Rrightarrow^{\emptyset} \exists l.\, \mathrm{Qu}(q, l) * \left(\mathrm{Qu}(q, l \mathbin{+\!\!+} [2])\ {}^{\emptyset}\!\Rrightarrow^{\top} \mathrm{True}\right)$$

The invariant rules give us:

$$\boxed{I}^{\mathcal{N}} \vdash \mathrm{True}\ {}^{\top}\!\Rrightarrow^{\emptyset} I * \left(I\ {}^{\emptyset}\!\Rrightarrow^{\top} \mathrm{True}\right)$$

for our $I \triangleq \exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)$

`enqueue` on a shared queue

Assume: $\boxed{\exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)}^{\mathcal{N}}$

Goal:

$\mathrm{True} \;{}^{\top}{\Rrightarrow}^{\emptyset}\; \exists l.\, \mathrm{Qu}(q, l) * \left(\mathrm{Qu}(q, l +\!\!+ [2]) \;{}^{\emptyset}{\Rrightarrow}^{\top}\; \mathrm{True}\right)$

The invariant rules give us:

$\boxed{I}^{\mathcal{N}} \vdash \mathrm{True} \;{}^{\top}{\Rrightarrow}^{\emptyset}\; I * \left(I \;{}^{\emptyset}{\Rrightarrow}^{\top}\; \mathrm{True}\right)$

for our $I \triangleq \exists l.\, \mathrm{Qu}(q, l) * \forall n \in l.\, \mathrm{even}(n)$

Now all we need is

$I -\!\!* \; \exists l.\, \mathrm{Qu}(q, l) * \left(\mathrm{Qu}(q, l +\!\!+ [2]) -\!\!* \; I\right)$

which is trivial.

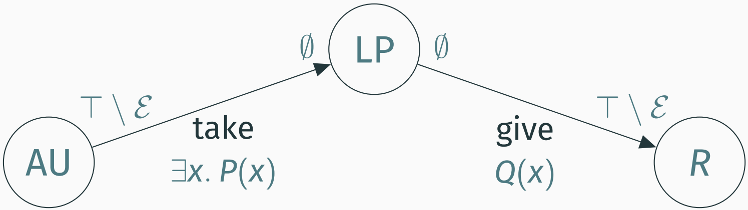
We have seen logical atomicity for `enqueue`.

This can be generalized!

$$\langle x.\, P(x)\rangle\; e\; \langle Q(x)\rangle_{\mathcal{E}} \triangleq \forall R.\, \{\mathsf{AU}_R\}\; e\; \{R\}$$

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top\setminus\mathcal{E}}{\Rrightarrow\!\!\ast}^{\emptyset}\ \exists x.\, P(x) * \left(Q(x)\ {}^{\emptyset}{\Rrightarrow\!\!\ast}^{\top\setminus\mathcal{E}}\ R\right)$$

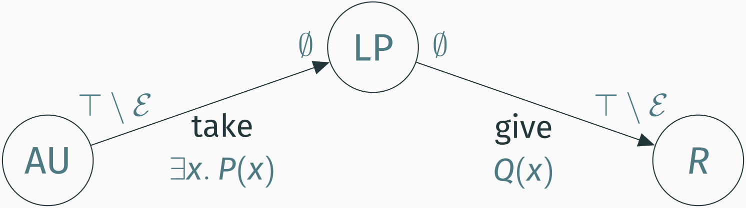
Meaning of atomic update AU_R:



$$\langle x.\ P(x)\rangle\ e\ \langle Q(x)\rangle_{\mathcal{E}} \triangleq \forall R.\ \{\mathsf{AU}_R\}\ e\ \{R\}$$

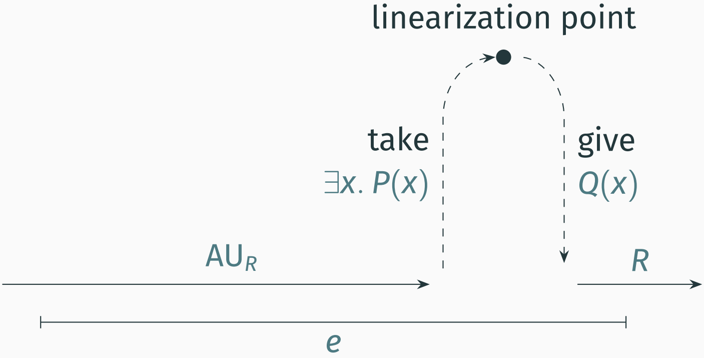
$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top\setminus\mathcal{E}}\!\Rrightarrow\!\!*^{\emptyset}\ \exists x.\ P(x) * \left(Q(x)\ {}^{\emptyset}\!\Rrightarrow\!\!*^{\top\setminus\mathcal{E}}\ R\right)$$

Meaning of atomic update AU_R:



$$\langle x.\, P(x)\rangle\ e\ \langle Q(x)\rangle_{\mathcal{E}} \triangleq \forall R.\ \{\mathrm{AU}_R\}\ e\ \{R\}$$

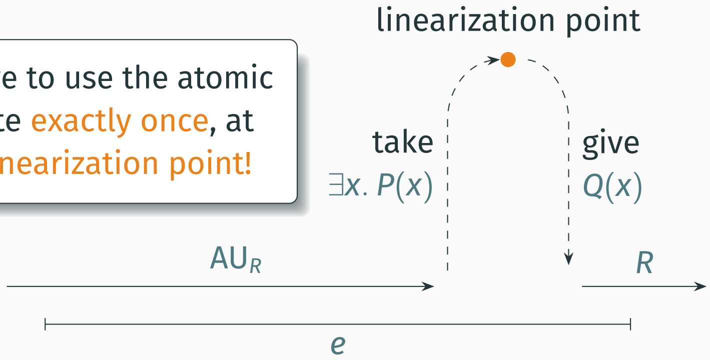
$$\mathrm{AU}_R \triangleq \mathrm{True}\ {}^{\top\setminus\mathcal{E}}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists x.\, P(x) \ast \left(Q(x)\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top\setminus\mathcal{E}}\ R\right)$$



$$\langle x.\, P(x)\rangle\; e\; \langle Q(x)\rangle_{\mathcal{E}} \triangleq \forall R.\, \{\mathrm{AU}_R\}\; e\; \{R\}$$

$$\mathrm{AU}_R \triangleq \mathrm{True}\; {}^{\top\setminus\mathcal{E}}{\Rrightarrow\!\!\ast}^{\emptyset}\; \exists x.\, P(x) * \left(Q(x)\; {}^{\emptyset}{\Rrightarrow\!\!\ast}^{\top\setminus\mathcal{E}}\; R\right)$$

We have to use the atomic update exactly once, at the linearization point!



Logically atomic triples enjoy the Invariant Rule:

$$\frac{\langle x.\, P * I \rangle\ e\ \langle Q * I \rangle_{\mathcal{E} \setminus \mathcal{N}} \qquad \mathcal{N} \subseteq \mathcal{E}}{\boxed{I}^{\mathcal{N}} \vdash \langle x.\, P \rangle\ e\ \langle Q \rangle_{\mathcal{E}}}$$

"An operation is atomic
if we can open invariants around it."

Outline

1. How to specify and use basic logically atomic operations in Iris
2. Advanced logically atomic patterns: aborting, helping
3. Summary and case studies

Can we specify and prove a blocking `dequeue`?

Implementation:

```
blocking_dequeue(q) ≜
    match dequeue(q) with
     Some(x) ⇒ x
    | None    ⇒ blocking_dequeue(q)
    end
```

Implementation:

$$\texttt{blocking\_dequeue}(q) \triangleq$$

```
match dequeue(q) with
  Some(x) ⇒ x
| None    ⇒ blocking_dequeue(q)
end
```

Specification:

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\texttt{blocking\_dequeue}(q)$$
$$\langle v.\ \exists l'.\ l = v :: l' * \mathsf{Queue}(q, l')\rangle$$

Specification:

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\texttt{blocking\_dequeue}(q)$$
$$\langle v.\ \exists l'.\ l = v :: l' * \mathsf{Queue}(q, l')\rangle$$

expands to

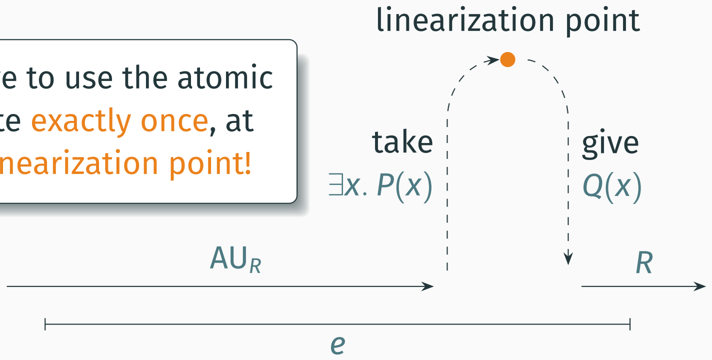
$\forall R.\ \{\mathsf{AU}_R\}\ \texttt{blocking\_dequeue}(q)\ \{v.\ R(v)\}$ where

$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}{\Rrightarrow}\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Queue}(q, l) * \left(\forall v.\ (\exists l'.\ l = v :: l' * \mathsf{Queue}(q, l'))\ {}^{\emptyset}{\Rrightarrow}\!\!\ast^{\top}\ R(v)\right)$$

$$\langle x.\, P(x)\rangle\; e\; \langle Q(x)\rangle_{\mathcal{E}} \triangleq \forall R.\, \{\mathrm{AU}_R\}\; e\; \{R\}$$

$$\mathrm{AU}_R \triangleq \mathrm{True}\ {}^{\top\setminus\mathcal{E}}{\Rrightarrow\!\!\ast}^{\emptyset}\ \exists x.\, P(x) \ast \left(Q(x)\ {}^{\emptyset}{\Rrightarrow\!\!\ast}^{\top\setminus\mathcal{E}}\ R\right)$$

We have to use the atomic update exactly once, at the linearization point!



Implementation:

```
blocking_dequeue(q) ≜
    match dequeue(q) with
     Some(x) ⇒ x
    | None   ⇒ blocking_dequeue(q)
    end
```

The first call to `dequeue` will consume AU!

Implementation:

$\texttt{blocking\_dequeue}(q) \triangleq$

$\texttt{match dequeue}(q)\ \texttt{with}$

To be able to derive `blocking_dequeue` (without breaking the abstraction), we have to adjust our definition of logical atomicity.

consume AU!

Implementation:

```
blocking_dequeue(q) ≜
    match dequeue(q) with
      Some(x) ⇒ x
    | None    ⇒ blocking_dequeue(q)
    end
```

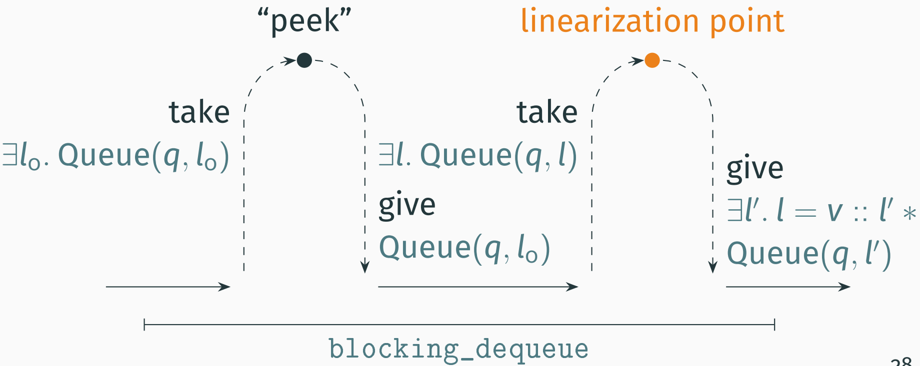
Specification:

$$\langle l.\ \mathrm{Queue}(q, l)\rangle$$

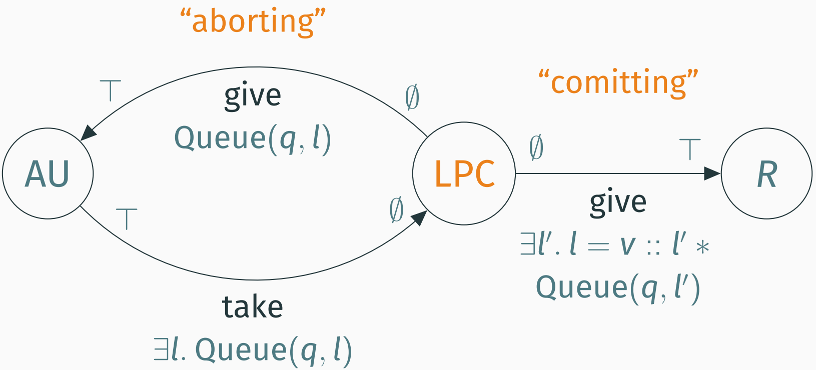
$$\texttt{blocking\_dequeue}(q)$$

$$\langle v.\ \exists l'.\ l = v :: l' * \mathrm{Queue}(q, l')\rangle$$

$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\texttt{blocking\_dequeue}(q)$$
$$\langle v.\ \exists l'.\ l = v :: l' * \mathsf{Queue}(q, l')\rangle$$



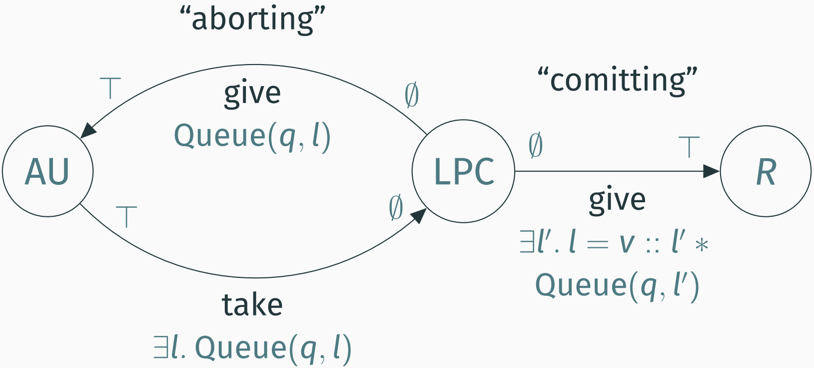
$$\langle l.\ \mathsf{Queue}(q, l)\rangle$$
$$\quad \texttt{blocking\_dequeue}(q)$$
$$\langle v.\ \exists l'.\ l = v :: l' * \mathsf{Queue}(q, l')\rangle$$



$$\mathsf{AU}_R \triangleq \mathsf{True} \;{}^{\top}\!\!\Rrightarrow\!\!\!\ast^{\emptyset}\; \exists l.\, \mathsf{Queue}(q, l) \ast \mathsf{LPC}_{R,l}$$

$$\mathsf{LPC}_{R,l} \triangleq$$

$$\left(\forall v.\, (\exists l'.\, l = v :: l' \ast \mathsf{Queue}(q, l')) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\!\ast^{\top}\; R(v)\right)$$



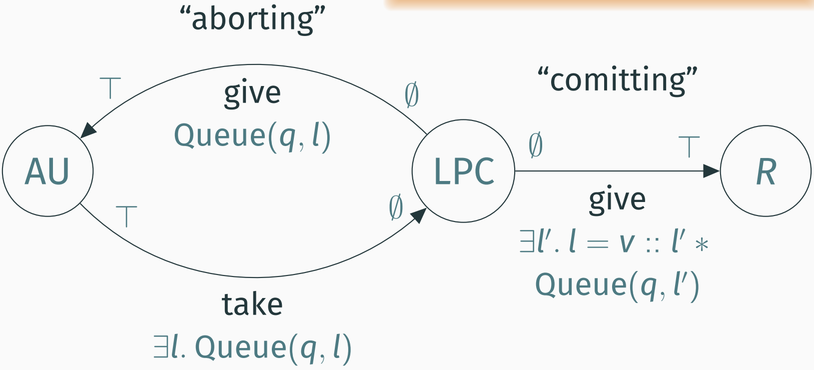
$$\mathrm{AU}_R \triangleq \mathrm{True} \;{}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\; \exists l.\, \mathrm{Queue}(q, l) \ast \mathrm{LPC}_{R,l}$$

$$\mathrm{LPC}_{R,l} \triangleq$$

$$\wedge$$

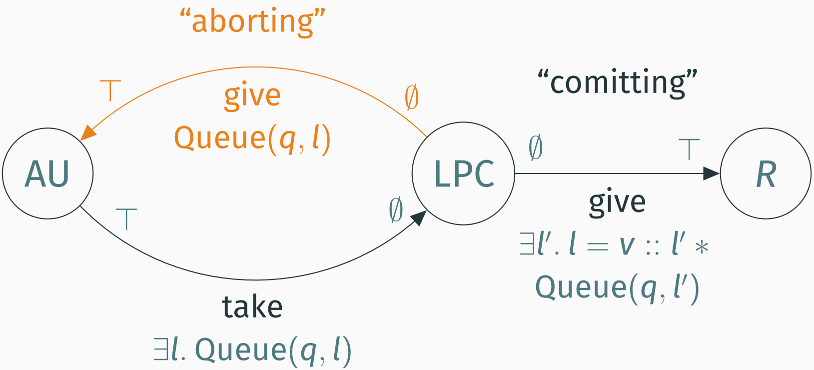
$$\Big(\forall v.\, \big(\exists l'.\, l = v :: l' \ast \mathrm{Queue}(q, l')\big) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\; R(v)\Big)$$

Conjunction ≅ “choice”



$$\mathsf{AU}_R \triangleq \mathsf{True} \;{}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\; \exists l.\, \mathsf{Queue}(q, l) * \mathsf{LPC}_{R,l}$$

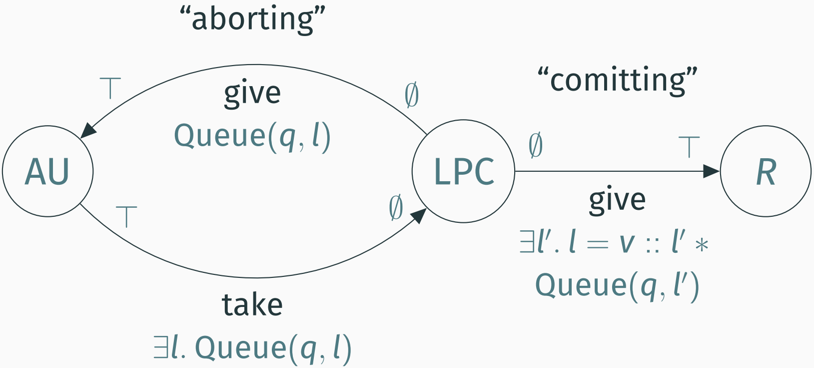
$$\mathsf{LPC}_{R,l} \triangleq \left(\mathsf{Queue}(q, l) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\; \mathsf{AU}_R\right) \wedge \left(\forall v.\, \left(\exists l'.\, l = v :: l' * \mathsf{Queue}(q, l')\right) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\; R(v)\right)$$



$$\mathsf{AU}_R \triangleq \mathsf{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists l.\ \mathsf{Queue}(q, l) * \mathsf{LPC}_{R,l}$$

$$\mathsf{LPC}_{R,l} \triangleq \left(\mathsf{Queue}(q, l)\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ \mathsf{AU}_R\right) \wedge$$

$$\left(\forall v.\ \left(\exists l'.\ l = v :: l' * \mathsf{Queue}(q, l')\right)\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ R(v)\right)$$



$$\mathsf{AU}_R \triangleq \mathsf{True} \;{}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\; \exists l.\, \mathsf{Queue}(q, l) \ast \mathsf{LPC}_{R,l}$$

$$\begin{aligned}\mathsf{LPC}_{R,l} \triangleq\; & \left(\mathsf{Queue}(q, l) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\; \mathsf{AU}_R\right) \wedge \\ & \left(\forall v.\, \left(\exists l'.\, l = v :: l' \ast \mathsf{Queue}(q, l')\right) \;{}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\; R(v)\right)\end{aligned}$$

We can tie the recursive knot using a (greatest) fixed point.

By “aborting” when `dequeue` fails, we can prove the desired specification for `blocking_dequeue`.

$$\langle l.\ \mathrm{Queue}(q, l) \rangle$$
$$\quad \texttt{blocking\_dequeue}(q)$$
$$\langle v.\ \exists l'.\ l = v :: l' * \mathrm{Queue}(q, l') \rangle$$

Helping

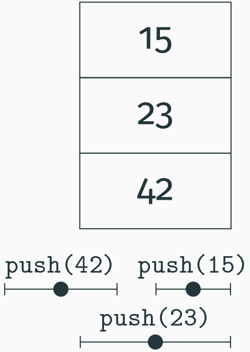
One thread can complete the action of another.

Helping

One thread can complete the action of another.

For example:

Stack content:



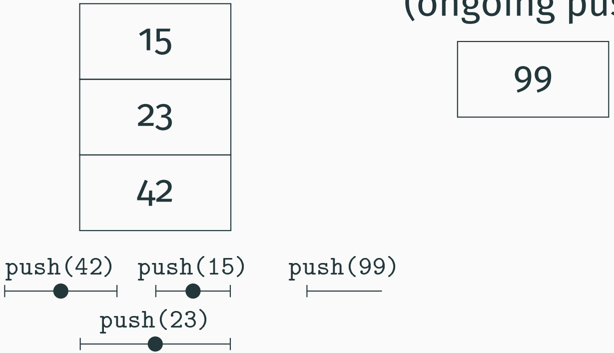
Helping

One thread can complete the action of another.

For example:

Stack content:

"Bypass offer":
(ongoing push)

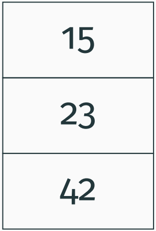


Helping

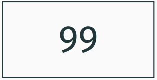
One thread can complete the action of another.

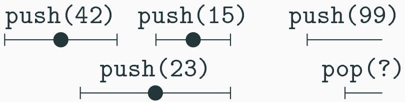
For example:

Stack content:



"Bypass offer":
(ongoing push)

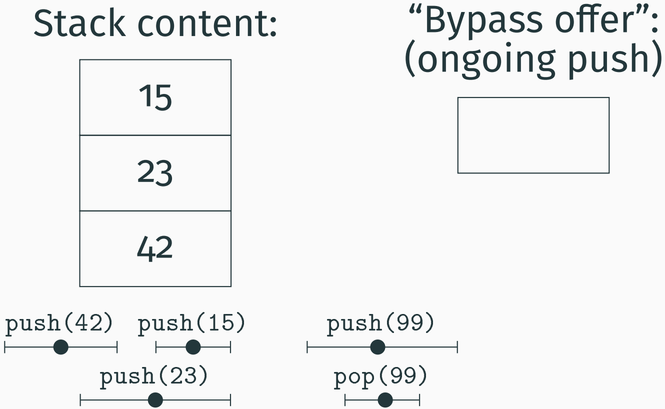




Helping

One thread can complete the action of another.

For example:



Helping

One thread can complete the action of another

AU_R is just a (separation logic) resource!

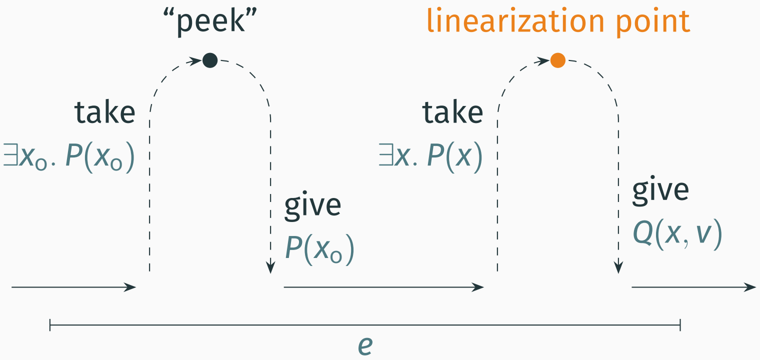
We can send it from one thread to another.

1. Thread A puts their AU_R into invariant
2. Thread B receives AU_R
3. Thread B completes both runs AU_R and its own $AU_{R'}$
4. Thread B puts results R back into invariant
5. Thread A obtains result R and completes

Outline

1. How to specify and use basic logically atomic operations in Iris
2. Advanced logically atomic patterns: aborting, helping
3. Summary and case studies

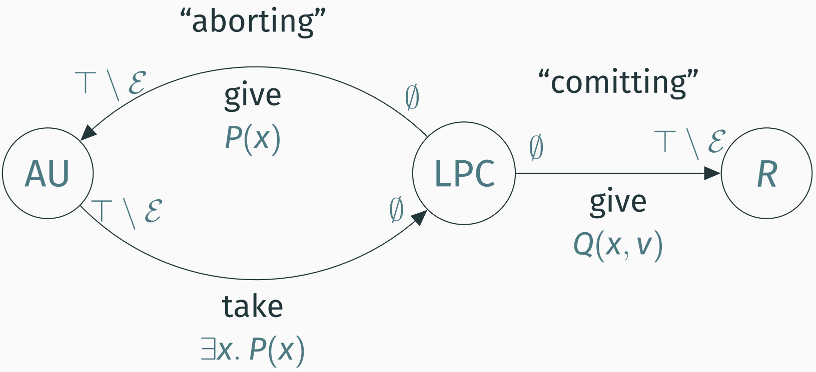
$$\langle x.\, P(x) \rangle\ e\ \langle v.\, Q(x, v) \rangle_{\mathcal{E}}$$



$$\langle x.\, P(x)\rangle\, e\, \langle v.\, Q(x,v)\rangle_{\mathcal{E}} \triangleq \forall R.\, \{\mathrm{AU}_R\}\, e\, \{v.\, R(v)\}$$

$$\mathrm{AU}_R \triangleq \mathrm{True}\ {}^{\top}\!\!\Rrightarrow\!\!\ast^{\emptyset}\ \exists x.\, P(x) * \mathrm{LPC}_{R,x}$$

$$\mathrm{LPC}_{R,x} \triangleq \left(P(x)\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ \mathrm{AU}_R\right) \wedge \left(\forall v.\, Q(x,v)\ {}^{\emptyset}\!\!\Rrightarrow\!\!\ast^{\top}\ R(v)\right)$$



Logically Atomic Case Studies

- Elimination Stack on abstract heap

https://iris-project.org

Logically Atomic Case Studies

- Elimination Stack on abstract heap
- Flat Combiner (by Zhen)

Iris

https://iris-project.org

Logically Atomic Case Studies

- Elimination Stack on abstract heap
- Flat Combiner (by Zhen)
- Atomic snapshot (by Marianna)
- RDCSS (by Marianna, Rodolphe and Gaurav)

Iris

https://iris-project.org

Logically Atomic Case Studies

- Elimination Stack on abstract heap
- Flat Combiner (by Zhen)
- Atomic snapshot (by Marianna)
- RDCSS (by Marianna, Rodolphe and Gaurav)
- Herlihy-Wing-Queue
  (by Rodolphe, Derek, Gaurav)

https://iris-project.org

Logically Atomic Case Studies

- Elimination Stack on abstract heap

Logical atomicity **implies linearizability:**

"Theorems for Free from Separation Logic Specifications"

Birkedal, Dinsdale-Young, Guéneau, Jaber, Svendsen, Tzevelekos; ICFP 2021

https://iris-project.org

Logical Atomicity lets us give

- concise and powerful
- Hoare-style specifications
- to concurrent data structures
- that make use of helping.

https://iris-project.org