Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, Derek Dreyer

Max Planck Institute for Software Systems (MPI-SWS), Saarland University, Aarhus University, Mozilla Research

Jan 17th
POPL 2015, Mumbai

Iris

A new separation logic that

- can verify complex, lock-free concurrent datastructures
- permits modular (thread-local) reasoning

Tons of prior program logics

Tons of prior program logics

- CSL [O'H07]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]
- iCAP [SB14]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]
- iCAP [SB14]
- FCSL [Nan+14]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]
- iCAP [SB14]
- FCSL [Nan+14]
- TaDA [dDYG14]

Tons of prior program logics

- CSL [O'H07]
- RGSep [VP07]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- FCSL [Nan+14]
- TaDA [dDYG14]

Do we really need yet another concurrency logic?

Yet another concurrency logic

Iris addresses two problems

- Simplifying the foundations of concurrent reasoning
- Supporting a notion of *logical atomicity*

Problem 1: Protocols

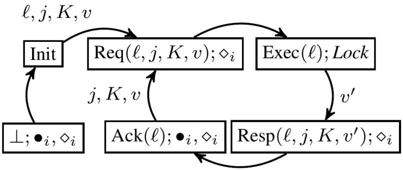
- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]
- iCAP [SB14]
- FCSL [Nan+14]
- TaDA [dDYG14]

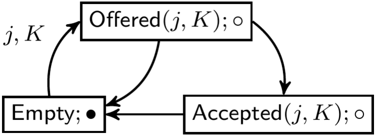
Problem 1: Protocols

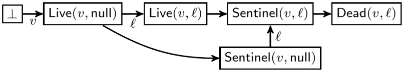
- CSL [O'H07]
- RGSep [VP07]
- SAGL [FFS07]
- LRG [Fen09]
- CAP [DY+10]
- HLRG [Fu+10]
- CaReSL [TDB13]
- SCSL [LWN13]
- HoCAP [SBP13]
- iCAP [SB14]
- FCSL [Nan+14]
- TaDA [dDYG14]

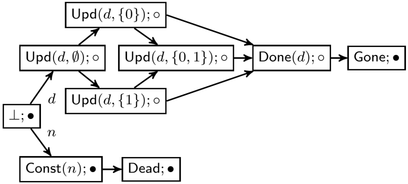
Common approach: "protocol" to deal with interference

STSs in CaReSL









Complex rules built-in as primitives

CaReSL: $$\frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_\pi} b_0.\ (\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!|x.\ \exists b' \overset{\text{guar}}{\sqsupseteq_\pi} b.\ \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\}\ i \Rrightarrow a\ \left\{ x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q \right\}} \textsc{UpdIsl}$$

Complex rules built-in as primitives

CaReSL:

$$\frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_\pi} b_0.\ (\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!|x.\ \exists b' \overset{\text{guar}}{\sqsupseteq_\pi} b.\ \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\}\ i \Rrightarrow a\ \left\{ x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q \right\}}\ \textsc{UpdIsl}$$

iCAP:

$$\frac{\begin{array}{c}\Gamma, \Delta \mid \Phi \vdash \mathsf{stable}(\mathsf{P}) \qquad \Gamma, \Delta \mid \Phi \vdash \forall y.\ \mathsf{stable}(\mathsf{Q}(y)) \\ \Gamma, \Delta \mid \Phi \vdash n \in C \qquad \Gamma, \Delta \mid \Phi \vdash \forall x \in X.\ (x, f(x)) \in \overline{T(A)} \vee f(x) = x \\ \Gamma \mid \Phi \vdash \forall x \in X.\ (\Delta).\langle \mathsf{P} * \circledast_{\alpha \in A}[\alpha]^{\mathsf{n}}_{g(\alpha)} * \triangleright I(x) \rangle\ c\ \langle \mathsf{Q}(x) * \triangleright I(f(x)) \rangle^{C \setminus \{n\}}\end{array}}{\begin{array}{c}\Gamma \mid \Phi \vdash (\Delta).\ \langle \mathsf{P} * \circledast_{\alpha \in A}[\alpha]^{\mathsf{n}}_{g(\alpha)} * \mathsf{region}(X, T, I, n) \rangle \\ c \\ \langle \exists x.\ \mathsf{Q}(x) * \mathsf{region}(\{f(x)\}, T, I, n) \rangle^{C}\end{array}}\ \textsc{Atomic}$$

TaDA:

Use atomic rule

$$\frac{\begin{array}{c} a \notin \mathcal{A} \quad \forall x \in X.\ (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^{*} \\ \lambda; \mathcal{A} \vdash \mathbb{W} x \in X.\ \langle p_p \mid I(\mathbf{t}^{\lambda}_{a}(x)) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\ \langle q_p(x, y) \mid I(\mathbf{t}^{\lambda}_{a}(f(x))) * q(x, y) \rangle \end{array}}{\lambda + 1; \mathcal{A} \vdash \mathbb{W} x \in X.\ \langle p_p \mid \mathbf{t}^{\lambda}_{a}(x) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\ \langle q_p(x, y) \mid \mathbf{t}^{\lambda}_{a}(f(x)) * q(x, y) \rangle}$$

Complex rules built-in as primitives

CaReSL: $$\dfrac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_\pi} b_0.\ \langle\!| \pi[\![b]\!] * P |\!\rangle\ i \Rrightarrow_1 a\ \langle\!| x.\ \exists b' \overset{\text{guar}}{\sqsupseteq_\pi} b.\ \pi[\![b']\!] * Q |\!\rangle}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\}\ i \Rrightarrow a\ \left\{ x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q \right\}}\ \textsc{UpdIsl}$$

All you need are two simple primitives:

- *Monoids* to *express* protocols.
- *Invariants* to *enforce* protocols.

TaDA: Use atomic rule

$$\dfrac{a \notin \mathcal{A} \quad \forall x \in X.\, (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^* \qquad \lambda; \mathcal{A} \vdash \mathbb{W} x \in X.\, \langle p_p \mid I(\mathbf{t}^{\lambda}_{a}(x)) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\, \langle q_p(x,y) \mid I(\mathbf{t}^{\lambda}_{a}(f(x))) * q(x,y) \rangle}{\lambda + 1; \mathcal{A} \vdash \mathbb{W} x \in X.\, \langle p_p \mid \mathbf{t}^{\lambda}_{a}(x) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\, \langle q_p(x,y) \mid \mathbf{t}^{\lambda}_{a}(f(x)) * q(x,y) \rangle}$$

Problem 2: Specifying Atomicity

Complex implementation

fn *push_fancy*(s, x) {
  let h_n = **ref** (next $\mapsto$ **null**, value $\mapsto$ x) **in**
  let h_o = $!s$.head **in**
  h_n.next := h_o;
  let b = **cas**(s.head, h_o, h_n) **in**
  if b **then** () **else**
  let o = **ref** (state $\mapsto$ 0, value $\mapsto$ x) **in**
  s.offer := o;
  s.offer := **null**;
  let b = **cas**(o.state, 0, 2) **in**
  if b **then** *push_fancy*(s, x) **else skip**
}

Problem 2: Specifying Atomicity

Complex implementation

$$
\begin{array}{l}
\textbf{fn } \mathit{push\_fancy}(s, x)\ \{ \\
\quad \textbf{let } h_n = \textbf{ref}\ (\mathsf{next} \mapsto \textbf{null}, \mathsf{value} \mapsto x)\ \textbf{in} \\
\quad \textbf{let } h_o = !s.\mathsf{head}\ \textbf{in} \\
\quad h_n.\mathsf{next} := h_o; \\
\quad \textbf{let } b = \textbf{cas}(s.\mathsf{head}, h_o, h_n)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } ()\ \textbf{else} \\
\quad \textbf{let } o = \textbf{ref}\ (\mathsf{state} \mapsto 0, \mathsf{value} \mapsto x)\ \textbf{in} \\
\quad s.\mathsf{offer} := o; \\
\quad s.\mathsf{offer} := \textbf{null}; \\
\quad \textbf{let } b = \textbf{cas}(o.\mathsf{state}, 0, 2)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } \mathit{push\_fancy}(s, x)\ \textbf{else skip} \\
\}
\end{array}
$$

simple implementation

$$
\begin{array}{l}
\textbf{fn } \mathit{push\_spec}(s, x)\ \{ \\
\quad \textbf{atomic}\ \{ \\
\quad\quad s := (!s) :: x \\
\quad \} \\
\}
\end{array}
$$

Problem 2: Specifying Atomicity

Complex implementation | refines | simple implementation

$$
\begin{array}{l}
\textbf{fn } \mathit{push\_fancy}(s, x) \ \{ \\
\quad \textbf{let } h_n = \textbf{ref } (\text{next} \mapsto \textbf{null}, \text{value} \mapsto x) \textbf{ in} \\
\quad \textbf{let } h_o = !s.\text{head} \textbf{ in} \\
\quad h_n.\text{next} := h_o; \\
\quad \textbf{let } b = \textbf{cas}(s.\text{head}, h_o, h_n) \textbf{ in} \\
\quad \textbf{if } b \textbf{ then } () \textbf{ else} \\
\quad \textbf{let } o = \textbf{ref } (\text{state} \mapsto 0, \text{value} \mapsto x) \textbf{ in} \\
\quad s.\text{offer} := o; \\
\quad s.\text{offer} := \textbf{null}; \\
\quad \textbf{let } b = \textbf{cas}(o.\text{state}, 0, 2) \textbf{ in} \\
\quad \textbf{if } b \textbf{ then } \mathit{push\_fancy}(s, x) \textbf{ else skip} \\
\}
\end{array}
\quad \leq \quad
\begin{array}{l}
\textbf{fn } \mathit{push\_spec}(s, x) \ \{ \\
\quad \textbf{atomic } \{ \\
\quad\quad s := (!s) :: x \\
\quad \} \\
\}
\end{array}
$$

Contextual refinement / Linearizability

Problem 2: Specifying Atomicity

Complex implementation | refines | simple implementation

```
fn push_fancy(s, x) {
  let h_n = ref (next ↦ null, value ↦ x) in
  let h_o = !s.head in
  h_n.next := h_o;
  let b = cas(s.head, h_o, h_n) in
  if b then () else
  let o = ref (state ↦ 0, value ↦ x) in
  s.offer := o;
  s.offer := null;
  let b = cas(o.state, 0, 2) in
  if b then push_fancy(s, x) else skip
}
```

$\leq$

```
fn push_spec(s, x) {
  atomic {
    s := (!s) :: x
  }
}
```

↑

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation | refines | simple implementation

$$\frac{push\_fancy \leq push\_spec}{\text{Behaviors}(C[push\_fancy]) \subseteq \text{Behaviors}(C[push\_spec])}$$

if b **then** $push\_fancy(s, x)$ **else skip**

}

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation refines simple implementation

$$\frac{push\_fancy \leq push\_spec \qquad \{P\}\ C[push\_spec]\ \{Q\}}{\{P\}\ C[push\_fancy]\ \{Q\}}$$

if b **then** $push\_fancy(s, x)$ **else skip**

}

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation refines simple implementation

$$\frac{push\_fancy \leq push\_spec \qquad \{P\}\ C[push\_spec]\ \{Q\}}{\{P\}\ C[push\_fancy]\ \{Q\}}$$

if b **then** $push\_fancy(s, x)$ **else skip**

}

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation — refines — simple implementation

$$
\begin{array}{l}
\textbf{fn } \mathit{push\_fancy}(s, x)\ \{ \\
\quad \textbf{let } h_n = \textbf{ref}\ (\mathsf{next} \mapsto \textbf{null}, \mathsf{value} \mapsto x)\ \textbf{in} \\
\quad \textbf{let } h_o = !s.\mathsf{head}\ \textbf{in} \\
\quad h_n.\mathsf{next} := h_o; \\
\quad \textbf{let } b = \textbf{cas}(s.\mathsf{head}, h_o, h_n)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } ()\ \textbf{else} \\
\quad \textbf{let } o = \textbf{ref}\ (\mathsf{state} \mapsto 0, \mathsf{value} \mapsto x)\ \textbf{in} \\
\quad s.\mathsf{offer} := o; \\
\quad s.\mathsf{offer} := \textbf{null}; \\
\quad \textbf{let } b = \textbf{cas}(o.\mathsf{state}, 0, 2)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } \mathit{push\_fancy}(s, x)\ \textbf{else skip} \\
\}
\end{array}
\quad \leq \quad
\begin{array}{l}
\textbf{fn } \mathit{push\_spec}(s, x)\ \{ \\
\quad \textbf{atomic}\ \{ \\
\quad\quad s := (!s) :: x \\
\quad \} \\
\}
\end{array}
$$

↑

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation — satisfies — logically atomic specification

$$
\begin{array}{l}
\textbf{fn } \textit{push\_fancy}(s, x)\ \{ \\
\quad \textbf{let } h_n = \textbf{ref}\ (\text{next} \mapsto \textbf{null}, \text{value} \mapsto x)\ \textbf{in} \\
\quad \textbf{let } h_o = !s.\text{head}\ \textbf{in} \\
\quad h_n.\text{next} := h_o; \\
\quad \textbf{let } b = \textbf{cas}(s.\text{head}, h_o, h_n)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } ()\ \textbf{else} \\
\quad \textbf{let } o = \textbf{ref}\ (\text{state} \mapsto 0, \text{value} \mapsto x)\ \textbf{in} \\
\quad s.\text{offer} := o; \\
\quad s.\text{offer} := \textbf{null}; \\
\quad \textbf{let } b = \textbf{cas}(o.\text{state}, 0, 2)\ \textbf{in} \\
\quad \textbf{if } b \textbf{ then } \textit{push\_fancy}(s, x)\ \textbf{else skip} \\
\}
\end{array}
$$

$$\models$$

$$
\begin{array}{l}
\langle \text{Stack}(s, l) \rangle \\
\quad \textit{push\_fancy}(s, x) \\
\langle \text{Stack}(s, x :: l) \rangle
\end{array}
$$

↑

Client $C[]$

Problem 2: Specifying Atomicity

Complex implementation | satisfies | logically atomic specification

$$\dfrac{\langle \mathsf{Stack}(s, l) \rangle \; \mathit{push\_fancy}(s, x) \; \langle \mathsf{Stack}(s, x :: l) \rangle \quad \cdots \vdash \{P\} \; C[\mathit{push\_fancy}] \; \{Q\}}{\{P\} \; C[\mathit{push\_fancy}] \; \{Q\}}$$

}

Client $C[]$

Iris Contributions

1. Encoding protocols with invariants and PCMs

Iris Contributions

1. Encoding protocols with invariants and PCMs
2. Supporting logically atomic specifications
    - Defined as derived notion

Iris Contributions

1. Encoding protocols with invariants and PCMs

Do a lot with little:
Derive rules that were built-in for previous logics

Protocols = Monoids + Invariants

A simple example

$$
\begin{array}{l}
\textbf{fn } \textit{inc2}(x)\ \{ \\
\quad \textbf{do}\ \{ \\
\qquad v = !x; \\
\qquad b = \textbf{cas}(x, v, v+2); \\
\quad \}\ \textbf{while}\ (\textbf{not}\ b); \\
\}
\end{array}
$$

A simple example

fn *inc2*(x) {
  do {
    $v = !x$;
    $b = \textbf{cas}(x, v, v + 2)$;
  } **while** (**not** b);
}

Set x from v to $v + 2$

A simple example

fn *inc2*(x) {
    do {
        $v = !x$;
        $b =$ **cas**$(x, v, v + 2)$;
    } **while** (**not** b);
}

Repeat until **cas** succeeds

Invariants

"x points to an even number"

$$
\begin{array}{l}
\textbf{fn } \mathit{inc2}(x)\ \{\\
\quad \textbf{do}\ \{\\
\qquad v = !x;\\
\qquad b = \textbf{cas}(x, v, v+2);\\
\quad \}\ \textbf{while}\ (\textbf{not}\ b);\\
\}
\end{array}
$$

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathrm{ev}(i)$

$$
\begin{array}{l}
\textbf{fn}\ \mathit{inc2}(x)\ \{ \\
\quad \textbf{do}\ \{ \\
\qquad v = !x; \\
\qquad b = \textbf{cas}(x, v, v + 2); \\
\quad \}\ \textbf{while}\ (\textbf{not}\ b); \\
\}
\end{array}
$$

Invariants

"x points to an even number" $R \triangleq \exists i.\, x \mapsto i * \mathrm{ev}(i)$

$\textbf{fn}\ \mathit{inc2}(x)\ \{$
$\quad \textbf{do}\ \{$
$\quad\quad v = !x;$
$\quad\quad b = \textbf{cas}(x, v, v + 2);$
$\quad \}\ \textbf{while}\ (\textbf{not}\ b);$
$\}$

Before **cas**:
Obtain invariant

Invariants

"x points to an even number" $R \triangleq \exists i.\; x \mapsto i * \mathrm{ev}(i)$

fn *inc2*(x) {
  do {
    $v = !x;$
    $b =$ **cas**$(x, v, v + 2);$
  } **while** (**not** b);
}

Before **cas**:
Obtain invariant

After **cas**:
Re-establish invariant

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathsf{ev}(i)$

```
fn inc2(x) {
  do {
    v = !x;
    b = cas(x, v, v + 2);
  } while (not b);
}
```

$$\frac{\{R * P\}\ e\ \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

cf. CSL [O'H07]

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathrm{ev}(i)$

$\mathbf{fn}\ \mathit{inc2}(x)\ \{$
$\quad \mathbf{do}\ \{$
$\quad\quad v = !x;$
$\quad\quad b = \mathbf{cas}(x, v, v + 2);$
$\quad \}\ \mathbf{while}\ (\mathbf{not}\ b);$
$\}$

$$\frac{\{R * P\}\ e\ \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

Invariant

cf. CSL [O'H07]

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathrm{ev}(i)$

fn *inc2*(x) {
  do {
    $v = !x;$
    $b = \mathbf{cas}(x, v, v + 2);$
  } **while** (**not** b);
}

$$\frac{\{R * P\}\ e\ \{R * Q\} \quad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

Resources carried in

cf. CSL [O'H07]

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathrm{ev}(i)$

$$
\begin{array}{l}
\textbf{fn } \mathit{inc2}(x)\ \{ \\
\quad \textbf{do}\ \{ \\
\qquad v = !x; \\
\qquad b = \textbf{cas}(x, v, v + 2); \\
\quad \}\ \textbf{while}\ (\textbf{not}\ b); \\
\}
\end{array}
$$

$$\frac{\{R * P\}\ e\ \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

cf. CSL [O'H07]

Resources carried out

Invariants

"x points to an even number" $R \triangleq \exists i.\ x \mapsto i * \mathrm{ev}(i)$

```
fn inc2(x) {
  do {
    v = !x;
    b = cas(x, v, v + 2);
  } while (not b);
}
```

$$\frac{\{R * P\}\ e\ \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

Avoid interference

cf. CSL [O'H07]

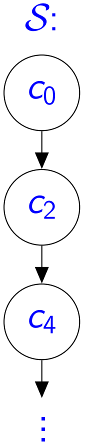
Invariants are not enough

"x points to a *monotonically increasing* even number"

$$
\begin{array}{l}
\textbf{fn}\ \mathit{inc2}(x)\ \{ \\
\quad \textbf{do}\ \{ \\
\qquad v = !x; \\
\qquad b = \textbf{cas}(x, v, v+2); \\
\quad \}\ \textbf{while}\ (\textbf{not}\ b); \\
\}
\end{array}
$$

STS Example

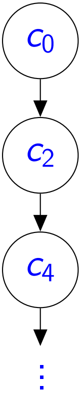
fn *inc2*(x) $\{$
    do $\{$
        $v = !x;$
        $b =$ **cas**$(x, v, v + 2);$
    $\}$ **while** (**not** b);
$\}$



STS Example

$\varphi(c_i) \triangleq x \mapsto i$

$\mathcal{S}$:




```
fn inc2(x) {
    do {
        v = !x;
        b = cas(x, v, v + 2);
    } while (not b);
}
```

STS Example

$\{\geq c_i\}$

fn *inc2*(x) {

  do {

    $v = !x;$

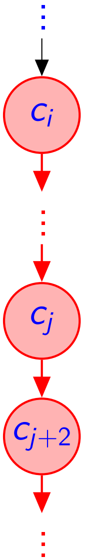
    $b =$ **cas**$(x, v, v+2);$

  } **while** (**not** b);

}

$\{\geq c_{i+2}\}$

$\varphi(c_i) \triangleq x \mapsto i$



STS Example

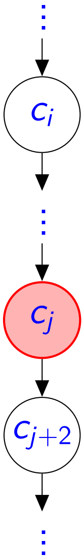
$\{\ge c_i\}$

fn *inc2*(x) {
    do {
        $v = !x;$
        $b =$ **cas**$(x, v, v + 2);$
    } **while** (**not** b);
}

$\{\ge c_{i+2}\}$

$\varphi(c_i) \triangleq x \mapsto i$

Current state: c_j



STS Example

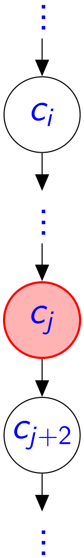
$\{\geq c_i\}$

fn *inc2*(x) {
  do {
    $v = !x;$
    $b =$ **cas**$(x, v, v + 2);$
  } **while** (**not** b);
}

$\{\geq c_{i+2}\}$

$\varphi(c_i) \triangleq x \mapsto i$

Current state: c_j,
so $x \mapsto j$



STS Example

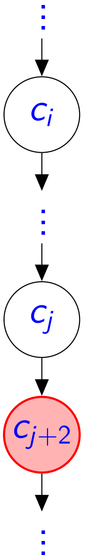
$\{\boxed{\geq c_i}\}$

$\varphi(c_i) \triangleq x \mapsto i$

fn *inc2*(x) {
  do {
    $v = !x;$
    $b =$ **cas**$(x, v, v+2);$
  } **while** (**not** b);
}

$\{\boxed{\geq c_{i+2}}\}$

Update state to c_{j+2}



STS Example

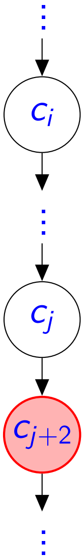
$\{\boxed{\geq c_i}\}$

$\varphi(c_i) \triangleq x \mapsto i$

```
fn inc2(x) {
    do {
        v = !x;
        b = cas(x, v, v + 2);
    } while (not b);
}
```

$\{\boxed{\geq c_{i+2}}\}$

Update state to c_{j+2}, show: $x \mapsto j + 2$



STS Example

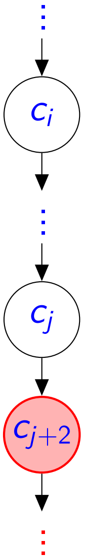
$\{\geq c_i\}$

```
fn inc2(x) {
    do {
        v = !x;
        b = cas(x, v, v + 2);
    } while (not b);
}
```

$\{\geq c_{i+2}\}$

$\varphi(c_i) \triangleq x \mapsto i$

Obtain $\geq c_{j+2}$



STS Example

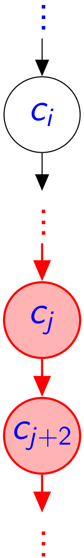
$\{\boxed{\geq c_i}\}$

$\varphi(c_i) \triangleq x \mapsto i$

fn *inc2*(x) {
    do {
        $v = !x;$
        $b = \textbf{cas}(x, v, v+2);$
    } **while** (**not** b);
}

Obtain $\boxed{\geq c_{j+2}}$

$\{\boxed{\geq c_{i+2}}\}$



STS Example

$$\frac{\forall c.\ \{\hat{c} \rightarrow^* c * \varphi(c) * P\}\ e\ \{v.\ \exists c'.\ c \rightarrow^* c' * \varphi(c') * Q\}}{\mathrm{STS}(\mathcal{S}, \varphi) \vdash \{\boxed{\geq \hat{c}} * P\}\ e\ \{v.\ \exists c'.\ \boxed{\geq c'} * Q\}}$$

$\{\boxed{\geq c_i}\}$

$\varphi(c_i) \triangleq x \mapsto i$

fn *inc2*(x) {

  do {

    $v = !x;$

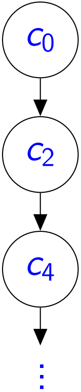
    $b = \mathbf{cas}(x, v, v + 2);$

  } **while** (**not** b);

}

$\{\boxed{\geq c_{i+2}}\}$

$\mathcal{S}$:



Complex rules built-in as primitives

CaReSL: $$\frac{\mathcal{C} \vdash \forall b \sqsupseteq^{\text{rely}}_{\pi} b_0.\ (\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_1 a\ (\!|x.\ \exists b' \sqsupseteq^{\text{guar}}_{\pi} b.\ \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{\boxed{b_0}^{n}_{\pi} * \triangleright P\right\}\ i \Rrightarrow a\ \left\{x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q\right\}}\ \textsc{UpdIsl}$$

All you need are two simple primitives:

- Invariants
- Partial commutative monoids

TaDA: $$\frac{a \notin \mathcal{A} \quad \forall x \in X.\ (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^* \qquad \lambda; \mathcal{A} \vdash \mathbb{V}x \in X.\ \langle p_p \mid I(\mathbf{t}^{\lambda}_a(x)) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\ \langle q_p(x,y) \mid I(\mathbf{t}^{\lambda}_a(f(x))) * q(x,y) \rangle}{\lambda + 1; \mathcal{A} \vdash \mathbb{V}x \in X.\ \langle p_p \mid \mathbf{t}^{\lambda}_a(x) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \exists\!\!\exists y \in Y.\ \langle q_p(x,y) \mid \mathbf{t}^{\lambda}_a(f(x)) * q(x,y) \rangle}\ \text{Use atomic rule}$$

Logical ("ghost") resources

Partial commutative monoid (PCM)

- Set M (carrier)
- An operation $\cdot$ on M (associative, commutative)
- A unit ϵ ("empty")
- A zero $\bot$ ("bottom", "undefined")

Logical ("ghost") resources

Partial commutative monoid (PCM)

- Set M (carrier)
- An operation $\cdot$ on M (associative, commutative)
- A unit ϵ ("empty")
- A zero $\bot$ ("bottom", "undefined")

Resource $a \in M$: Logical assertion $\boxed{a}$ ("own a")

Logical ("ghost") resources

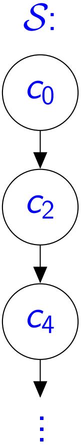
Partial commutative monoid (PCM)

- Set M (carrier)
- An operation $\cdot$ on M (associative, commutative)
- A unit ϵ ("empty")
- A zero $\bot$ ("bottom", "undefined")

Resource $a \in M$: Logical assertion $\boxed{a}$ ("own a")

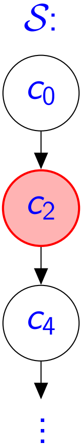
$$\frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}} \qquad \boxed{\bot} \Rightarrow \text{False}$$

STS monoid: Intuition



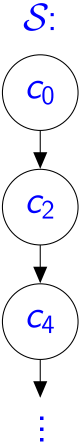
STS monoid: Intuition

Curr c_2



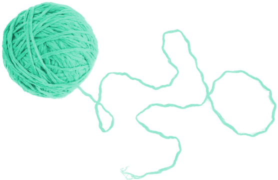
STS monoid: Intuition

Curr c_2



Poss $\{c_j \in \mathcal{S} \mid j \geq 2\}$

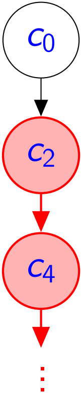
Poss $\mathcal{S}$



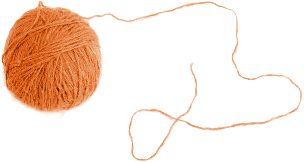
STS monoid: Intuition

Curr c_2

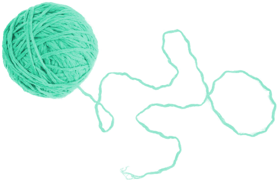
$\mathcal{S}$:



Poss $\{c_j \in \mathcal{S} \mid j \geq 2\}$

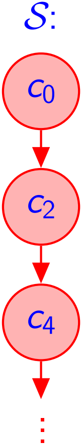


Poss $\mathcal{S}$



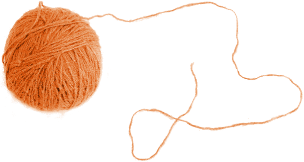
STS monoid: Intuition

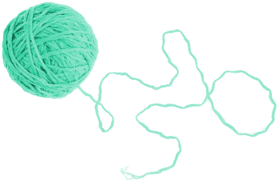
Curr c_2



Poss $\{c_j \in \mathcal{S} \mid j \geq 2\}$

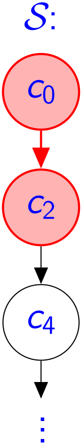
Poss $\mathcal{S}$





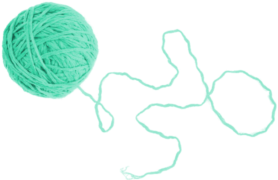
STS monoid: Intuition

Curr c_2



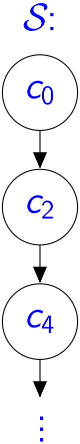
Poss $\{c_j \in \mathcal{S} \mid j \geq 2\}$

Poss $\{c_0, c_2\}$



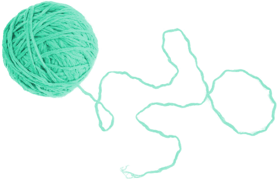
STS monoid: Intuition

Curr c_2



Poss $\{c_j \in \mathcal{S} \mid j \geq 2\}$

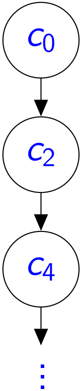
~~Poss $\{c_0, c_2\}$~~



STS monoid: Formal definition

$$M \triangleq \{ \quad \} \cup \left\{ \quad \right\}$$

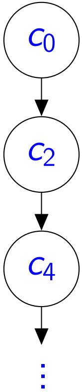
$\mathcal{S}$:



STS monoid: Formal definition

$$M \triangleq \{\mathsf{Curr}\ c \mid c \in \mathcal{S}\} \cup \left\{ \qquad \right\}$$

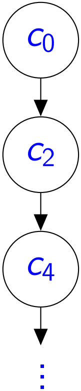
$\mathcal{S}$:



STS monoid: Formal definition

$$M \triangleq \{\mathsf{Curr}\ c \mid c \in \mathcal{S}\} \cup \left\{ \mathsf{Poss}\ B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \wedge B \neq \emptyset \wedge \\ \\ \end{array} \right\}$$

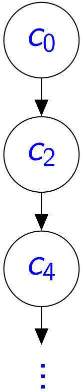
$\mathcal{S}$:



STS monoid: Formal definition

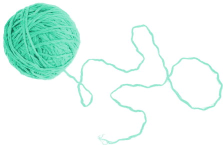
$$M \triangleq \{\text{Curr } c \mid c \in \mathcal{S}\} \cup \left\{ \text{Poss } B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \land B \neq \emptyset \land \\ B \text{ closed under } \to \end{array} \right\}$$

$\mathcal{S}$:

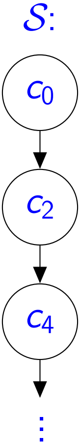


STS monoid: Formal definition

$$M \triangleq \{\text{Curr } c \mid c \in \mathcal{S}\} \cup \left\{ \text{Poss } B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \wedge B \neq \emptyset \wedge \\ B \text{ closed under } \rightarrow \end{array} \right\}$$

 $\parallel$  $\triangleq$ 

$$\text{Poss } B_1 \cdot \text{Poss } B_2 \triangleq \text{Poss } (B_1 \cap B_2)$$



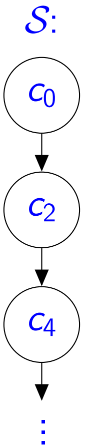
STS monoid: Formal definition

$$M \triangleq \{\text{Curr } c \mid c \in \mathcal{S}\} \cup \left\{ \text{Poss } B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \wedge B \neq \emptyset \wedge \\ B \text{ closed under } \rightarrow \end{array} \right\}$$

 $\|$  $\triangleq$ 

$$\text{Poss } B_1 \cdot \text{Poss } B_2 \triangleq \text{Poss } (B_1 \cap B_2)$$

$$\text{Poss } B \cdot \text{Curr } c \triangleq \text{Curr } c \quad \text{if } c \in B$$



STS monoid: Formal definition

$$M \triangleq \{\mathrm{Curr}\ c \mid c \in \mathcal{S}\} \cup \left\{\mathrm{Poss}\ B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \land B \neq \emptyset \land \\ B \text{ closed under } \to \end{array}\right\}$$

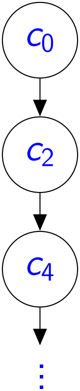
$\|$

$$\mathrm{Poss}\ B_1 \cdot \mathrm{Poss}\ B_2 \triangleq \mathrm{Poss}\ (B_1 \cap B_2)$$

$$\mathrm{Poss}\ B \cdot \mathrm{Curr}\ c \triangleq \mathrm{Curr}\ c \quad \text{if } c \in B$$

$$\mathrm{Curr}\ c_1 \cdot \mathrm{Curr}\ c_2 \triangleq \bot$$

$\mathcal{S}$:



STS invariant

STS invariant

Interaction of monoids and invariants

- Monoids serve to *express* protocols.
- Invariants serve to *enforce* protocols on shared state.

STS invariant

$$\text{Invariant } R \triangleq \exists c.\ \boxed{\mathsf{Curr}\ c} * \varphi(c)$$

Current state of STS

Interaction of monoids and invariants

- Monoids serve to *express* protocols.
- Invariants serve to *enforce* protocols on shared state.

STS invariant

Invariant $R \triangleq \exists c.\ \boxed{\text{Curr } c} * \varphi(c)$

STS interpretation

Interaction of monoids and invariants

- Monoids serve to *express* protocols.
- Invariants serve to *enforce* protocols on shared state.

Show $\{\boxed{\geq c_i}\}\ inc2(x)\ \{\boxed{\geq c_{i+2}}\}$

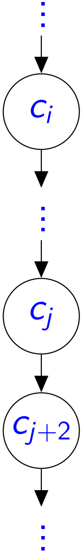
$$\frac{\forall c.\ \{\hat{c} \rightarrow^* c * \varphi(c) * P\}\ e\ \{v.\ \exists c'.\ c \rightarrow^* c' * \varphi(c') * Q\}}{\mathrm{STS}(\mathcal{S}, \varphi) \vdash \left\{\boxed{\geq \hat{c}} * P\right\}\ e\ \left\{v.\ \exists c'.\ \boxed{\geq c'} * Q\right\}}$$

Show $\{\boxed{\geq c_i}\}\ \mathit{inc2}(x)\ \{\boxed{\geq c_{i+2}}\}$

$$\frac{\forall c.\ \{\hat{c} \rightarrow^* c * \varphi(c) * P\}\ e\ \{v.\ \exists c'.\ c \rightarrow^* c' * \varphi(c') * Q\}}{\mathsf{STS}(\mathcal{S}, \varphi) \vdash \left\{\boxed{\geq \hat{c}} * P\right\}\ e\ \left\{v.\ \exists c'.\ \boxed{\geq c'} * Q\right\}}$$

STS reasoning

$R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$



$\{\boxed{\geq c_i}\}$

fn *inc2*(x) {

  do {

    $v = !x;$

    $b = \textbf{cas}(x, v, v + 2);$

  } **while** (**not** b);

}

$\{\boxed{\geq c_{i+2}}\}$

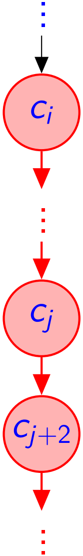
STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {
    do {
        $v = !x$;
        $b =$ **cas**($x, v, v + 2$);
    } **while** (**not** b);
}

$\{\boxed{\text{Poss } B_{i+2}}\}$



STS reasoning

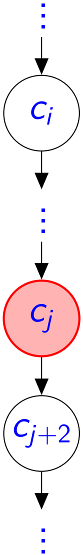
$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

```
fn inc2(x) {
  do {
    v = !x;
    b = cas(x, v, v + 2);
  } while (not b);
}
```

$\{\boxed{\text{Poss } B_{i+2}}\}$

Obtain R:
$\boxed{\text{Poss } B_i} * \boxed{\text{Curr } c_j} * x \mapsto j$



STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {

  do {

    $v = !x;$

    $b = \textbf{cas}(x, v, v + 2);$

  } **while** (**not** b);

}

$\{\boxed{\text{Poss } B_{i+2}}\}$

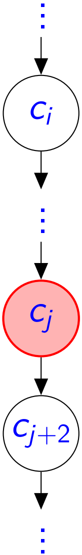
Obtain R:

$\boxed{\text{Poss } B_i} * \boxed{\text{Curr } c_j} * x \mapsto j$

Remember

$\text{Poss } B_i \cdot \text{Curr } c_j \triangleq \text{Curr } c_j$

if $c_j \in B_i$



STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

```
fn inc2(x) {
  do {
    v = !x;
    b = cas(x, v, v + 2);
  } while (not b);
}
```

$\{\boxed{\text{Poss } B_{i+2}}\}$

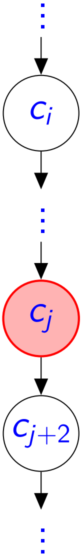
So we have:

$c_j \in B_i * \boxed{\text{Curr } c_j} * x \mapsto j$

Remember

$\text{Poss } B_i \cdot \text{Curr } c_j \triangleq \text{Curr } c_j$

if $c_j \in B_i$



STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {
  do {
    $v = !x$;
    $b =$ **cas**$(x, v, v + 2)$;
  } **while** (**not** b);
}

$\{\boxed{\text{Poss } B_{i+2}}\}$

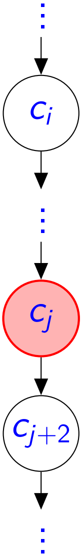
So we have:

$j \geq i * \boxed{\text{Curr } c_j} * x \mapsto j$

Remember

$\text{Poss } B_i \cdot \text{Curr } c_j \triangleq \text{Curr } c_j$

if $c_j \in B_i$



STS reasoning

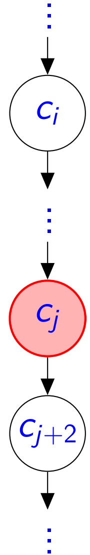
$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {
    do {
        $v = !x;$
        $b =$ **cas**$(x, v, v + 2);$
    } **while** (**not** b);
}

$\{\boxed{\text{Poss } B_{i+2}}\}$

We have:
$\boxed{\text{Curr } c_j} * x \mapsto j + 2,$
We want: R



STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {

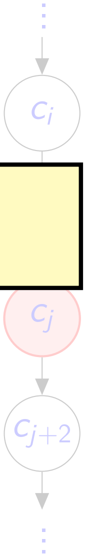
We need a way to update $\boxed{\text{Curr } c_j}$ to $\boxed{\text{Curr } c_{j+2}}$

$b =$ **cas**$(x, v, v+2)$;

} **while** (**not** b);

}

$\{\boxed{\text{Poss } B_{i+2}}\}$



STS reasoning

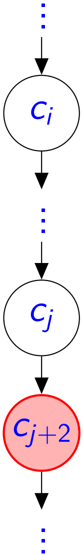
$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn $inc2(x)$ {
  do {
    $v = !x;$
    $b = \textbf{cas}(x, v, v+2);$
  } **while** (**not** b);
}

$\{\boxed{\text{Poss } B_{i+2}}\}$

We have R:
$\boxed{\text{Curr } c_{j+2}} * x \mapsto j + 2$



STS reasoning

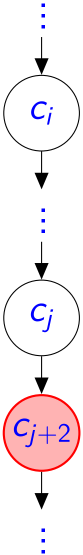
$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad$ $R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {
 do {
  $v = {!}x$;
  $b = \textbf{cas}(x, v, v + 2)$;
 } **while** (**not** b);
}

$\{\boxed{\text{Poss } B_{i+2}}\}$

We have R:
$\boxed{\text{Curr } c_{j+2}} * x \mapsto j + 2$,
We want: $\boxed{\text{Poss } B_{i+2}}$



STS reasoning

$B_i \triangleq \{c_j \in \mathcal{S} \mid j \geq i\}$ $\quad R \triangleq \exists c_i.\ \boxed{\text{Curr } c_i} * x \mapsto i$

$\{\boxed{\text{Poss } B_i}\}$

fn *inc2*(x) {

  do {

    $v = !x;$

    $b = $ **cas**$(x, v, v + 2);$

  } **while** (**not** b);

}

$\{\boxed{\text{Poss } B_{i+2}}\}$

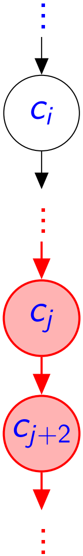
We have R:
$\boxed{\text{Curr } c_{j+2}} * x \mapsto j + 2,$
We want: $\boxed{\text{Poss } B_{i+2}}$

Remember

$\text{Poss } B_{i+2} \cdot \text{Curr } c_{j+2}$

$\triangleq \text{Curr } c_{j+2}$

if $c_{j+2} \in B_{i+2}$



Frame-preserving ghost update

$$\boxed{\text{Curr } c_j} \Rrightarrow \boxed{\text{Curr } c_{j+2}}$$

Frame-preserving ghost update

$$\boxed{a} \Rightarrow \boxed{b}$$

Frame-preserving ghost update

$$\boxed{a} \Rrightarrow \boxed{b}$$

	Us	vs.	the world
We always have	a	$\#$	a_f (for some *frame*)

where $a \# a_f \triangleq a \cdot a_f \neq \bot$

Frame-preserving ghost update

$$\boxed{a} \Rrightarrow \boxed{b}$$

	Us	vs.	the world
We always have	a	$\#$	a_f (for some *frame*)
So if we can show	b	$\#$	a_f

where $a \# a_f \triangleq a \cdot a_f \neq \bot$

Frame-preserving ghost update

$$\boxed{a} \Rrightarrow \boxed{b}$$

	Us	vs.	the world
We always have	a	$\#$	a_f (for some *frame*)
So if we can show	b	$\#$	a_f
We obtain	$\boxed{a} \Rrightarrow \boxed{b}$		

where $a \# a_f \triangleq a \cdot a_f \neq \bot$

Frame-preserving ghost update

$$\frac{\forall a_f.\ a \mathrel{\#} a_f \Rightarrow b \mathrel{\#} a_f}{\boxed{a} \Rrightarrow \boxed{b}}$$

cf. Views [DY+13]

	Us	vs.	the world
We always have	a	$\#$	a_f (for some *frame*)
So if we can show	b	$\#$	a_f
We obtain	$\boxed{a} \Rrightarrow \boxed{b}$		

where $a \mathrel{\#} a_f \triangleq a \cdot a_f \neq \bot$

Frame-preserving ghost update

$$\boxed{\text{Curr } c_j} \Rightarrow \boxed{\text{Curr } c_{j+2}}$$

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\mathsf{Curr}\ c} \Rrightarrow \boxed{\mathsf{Curr}\ c'}}$$

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\mathrm{Curr}\ c} \Rrightarrow \boxed{\mathrm{Curr}\ c'}}$$

	Us	vs.	the world
We have	$\mathrm{Curr}\ c$	$\#$	a_f

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\text{Curr } c} \Rrightarrow \boxed{\text{Curr } c'}}$$

	Us	vs.	the world
We have	$\text{Curr } c$	$\#$	?

Remember

$$\text{Curr } c_1 \cdot \text{Curr } c_2 \triangleq \bot$$

$\parallel$

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\mathsf{Curr}\ c} \Rrightarrow \boxed{\mathsf{Curr}\ c'}}$$

	Us	vs.	the world
We have	$\mathsf{Curr}\ c$	$\#$	$\mathsf{Poss}\ B$

Remember

$$\mathsf{Poss}\ B \cdot \mathsf{Curr}\ c \triangleq \mathsf{Curr}\ c \quad \text{if } c \in B$$

 $\parallel$  $\triangleq$ 

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\mathsf{Curr}\ c} \Rrightarrow \boxed{\mathsf{Curr}\ c'}}$$

Us	vs.	the world
We have $\mathsf{Curr}\ c$	$\#$	$\mathsf{Poss}\ B$, $c \in B$

Remember

$$\mathsf{Poss}\ B \cdot \mathsf{Curr}\ c \triangleq \mathsf{Curr}\ c \quad \text{if } c \in B$$

 $\|$  $\triangleq$ 

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\text{Curr } c} \Rrightarrow \boxed{\text{Curr } c'}}$$

	Us	vs.	the world
We have	$\text{Curr } c$	$\#$	$\text{Poss } B$, $c \in B$
Show	$\text{Curr } c'$	$\#$	$\text{Poss } B$: $c' \in B$

Remember

$$\text{Poss } B \cdot \text{Curr } c \triangleq \text{Curr } c \quad \text{if } c \in B$$

 $\|$  $\triangleq$ 

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\text{Curr } c} \Rrightarrow \boxed{\text{Curr } c'}}$$

	Us	vs.	the world
We have	$\text{Curr } c$	$\#$	$\text{Poss } B$, $c \in B$
Show	$\text{Curr } c'$	$\#$	$\text{Poss } B$: $c' \in B$

Remember

$$M \triangleq \{\text{Curr } c \mid c \in \mathcal{S}\} \cup \left\{ \text{Poss } B \;\middle|\; \begin{array}{l} B \subseteq \mathcal{S} \wedge B \neq \emptyset \wedge \\ B \text{ closed under } \to \end{array} \right\}$$

Frame-preserving ghost update

$$\frac{c \to^* c'}{\boxed{\text{Curr } c} \Rrightarrow \boxed{\text{Curr } c'}}$$

	Us	vs.	the world
We have	Curr c	$\#$	Poss B, $c \in B$
Show	Curr c'	$\#$	Poss B: $c' \in B$

$$\boxed{\text{Curr } c} \Rrightarrow \boxed{\text{Curr } c'}$$

Summary: Rules for PCMs and invariants

$$\frac{\{R * P\}\ e\ \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\ e\ \{Q\}}$$

$$\frac{\forall a_f.\ a \# a_f \Rightarrow b \# a_f}{\boxed{a} \Rrightarrow \boxed{b}}$$

$$\frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}}$$

$$\boxed{\bot} \Rightarrow \text{False}$$

Summary: Rules for PCMs and invariants

We can derive the STS update rule

$$\frac{\forall c.\ \{\hat{c} \rightarrow^* c * \varphi(c) * P\}\ e\ \{v.\ \exists c'.\ c \rightarrow^* c' * \varphi(c') * Q\}}{\mathsf{STS}(\mathcal{S}, \varphi) \vdash \left\{\boxed{\geq \hat{c}} * P\right\}\ e\ \left\{v.\ \exists c'.\ \boxed{\geq c'} * Q\right\}}$$

using just monoids and invariants.

What else is in the paper?

What else is in the paper?

- Contribution 2: Logically atomic specifications
  - à la TaDA by da Rocha Pinto, Dinsdale-Young, and Gardner [dDYG14]
  - Defined as derived form
  - Can handle helping

What else is in the paper?

- Contribution 2: Logically atomic specifications
  - à la TaDA by da Rocha Pinto, Dinsdale-Young, and Gardner [dDYG14]
  - Defined as derived form
  - Can handle helping
- More constructions based on monoids and invariants

What else is in the paper?

- Contribution 2: Logically atomic specifications
  - à la TaDA by da Rocha Pinto, Dinsdale-Young, and Gardner [dDYG14]
  - Defined as derived form
  - Can handle helping
- More constructions based on monoids and invariants
- Coq mechanization

Case study: Stack of abstractions

Elimination stack

Shared memory

Message-passing machine

Case study: Stack of abstractions

Elimination stack

You can do a lot with very little.

Message-passing machine

References I

[dDYG14] Pedro da Rocha Pinto, Thomas Dinsdale-Young, and Philippa Gardner. "TaDA: A Logic for Time and Data Abstraction". In: *ECOOP*. 2014.

[DY+10] T. Dinsdale-Young et al. "Concurrent abstract predicates". In: *ECOOP*. 2010.

[DY+13] Thomas Dinsdale-Young et al. "Views: Compositional reasoning for concurrent programs". In: *POPL*. 2013.

[Fen09] Xinyu Feng. "Local rely-guarantee reasoning". In: *POPL*. 2009.

[FFS07] Xinyu Feng, Rodrigo Ferreira, and Zhong Shao. "On the relationship between concurrent separation logic and assume-guarantee reasoning". In: *ESOP*. 2007.

[Fu+10] Ming Fu et al. "Reasoning about optimistic concurrency using a program logic for history". In: *CONCUR*. 2010.

[LWN13] Ruy Ley-Wild and Aleksandar Nanevski. "Subjective Auxiliary State for Coarse-Grained Concurrency". In: *POPL*. 2013.

[Nan+14] Aleksandar Nanevski et al. "Communicating State Transition Systems for Fine-Grained Concurrent Resources". In: *ESOP*. 2014.

[O'H07] P.W. O'Hearn. "Resources, concurrency, and local reasoning". In: *TCS* 375.1 (2007), pp. 271–307.

References II

[SB14] Kasper Svendsen and Lars Birkedal. "Impredicative Concurrent Abstract Predicates". In: *ESOP*. 2014.

[SBP13] Kasper Svendsen, Lars Birkedal, and Matthew J. Parkinson. "Modular Reasoning about Separation of Concurrent Data Structures". In: *ESOP*. 2013, pp. 169–188.

[TDB13] Aaron Turon, Derek Dreyer, and Lars Birkedal. "Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency". In: *ICFP*. 2013.

[VP07] V. Vafeiadis and M. Parkinson. "A marriage of rely/guarantee and separation logic". In: *CONCUR*. 2007.

Refinement and Hoare specs

$$e_1 \triangleq \textbf{let } x := \textbf{ref } 7 \textbf{ in } 42$$
$$e_2 \triangleq 42$$

Clearly, $e_2 \leq e_1$ and

$$\{\mathsf{True}\}\ e_1\ \{\exists x.\ x \mapsto 7\}$$

But this does not hold:

$$\{\mathsf{True}\}\ e_2\ \{\exists x.\ x \mapsto 7\}$$