# Concurrent Abstract Predicates

Chung-Kil Hur

21 Jul 2011
@CPL Seminar

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Language

$$(\mathsf{Cmd}) \qquad C ::= \mathbf{skip} \mid c \mid f \mid \langle C \rangle \mid C_1; C_2 \mid C_1 + C_2 \mid C^* \mid C_1 \| C_2 \mid$$
$$\text{let } f_1 = C_1 \ldots f_n = C_n \text{ in } C$$

$$\mathbf{if}(B)\ C_1\ \mathbf{else}\ C_2 \stackrel{\text{def}}{=} (\mathsf{assume}(B); C_1) + (\mathsf{assume}(\neg B); C_2)$$
$$\mathbf{while}(B)\ C \stackrel{\text{def}}{=} (\mathsf{assume}(B); C)^*; \mathsf{assume}(\neg B)$$

- $c \in \mathcal{P}(\mathrm{Heap} \times \mathrm{Heap})$: basic command
- How to model Stack ?
- Allow mutual recursion ?

# Operational Semantics

$$\boxed{\eta \in \mathsf{FEnv} \stackrel{\text{def}}{=} \mathsf{FName} \longrightarrow \mathsf{Cmd}}$$

$$\frac{(C,h) \xrightarrow{\eta[f_1 \mapsto C_1 \ldots f_n \mapsto C_n]} (C',h')}{(\text{let } f_1 = C_1 \ldots f_n = C_n \text{ in } C, h) \xrightarrow{\eta} (\text{let } f_1 = C_1 \ldots f_n = C_n \text{ in } C', h')}$$

$$\frac{}{(\text{let } \ldots \text{ in } \mathbf{skip}, h) \xrightarrow{\eta} (\mathbf{skip}, h)} \qquad \frac{f \in \mathrm{dom}(\eta)}{(f,h) \xrightarrow{\eta} (\eta(f), h)} \qquad \frac{(h,h') \in c \quad h' \neq \mathtt{fault}}{(c,h) \xrightarrow{\eta} (\mathbf{skip}, h')}$$

$$\frac{(C,h) \xrightarrow{\eta} (C_1,h')}{(C;C',h) \xrightarrow{\eta} (C_1;C',h')} \qquad \frac{}{(\mathbf{skip};C,h) \xrightarrow{\eta} (C,h)} \qquad \frac{}{(C^*,h) \xrightarrow{\eta} (\mathbf{skip} + (C;C^*), h)}$$

$$\frac{}{(C_1 + C_2, h) \xrightarrow{\eta} (C_1, h)} \qquad \frac{}{(C_1 + C_2, h) \xrightarrow{\eta} (C_2, h)} \qquad \frac{(C,h) \xrightarrow{\eta}{}^{*} (\mathbf{skip}, h')}{(\langle C \rangle, h) \xrightarrow{\eta} (\mathbf{skip}, h')}$$

$$\frac{(C_1,h) \xrightarrow{\eta} (C_1',h')}{(C_1 \| C_2, h) \xrightarrow{\eta} (C_1' \| C_2, h')} \qquad \frac{(C_2,h) \xrightarrow{\eta} (C_2',h')}{(C_1 \| C_2, h) \xrightarrow{\eta} (C_1 \| C_2', h')} \qquad \frac{}{(\mathbf{skip} \| \mathbf{skip}, h) \xrightarrow{\eta} (\mathbf{skip}, h)}$$

# Operational Semantics

$$\boxed{\eta \in \mathsf{FEnv} \stackrel{\text{def}}{=} \mathsf{FName} \rightarrow \mathsf{Cmd}}$$

$$\frac{(C_1, h) \xrightarrow{\eta} \mathit{fault}}{(C_1; C_2, h) \xrightarrow{\eta} \mathit{fault}} \qquad \frac{(C_1, h) \xrightarrow{\eta} \mathit{fault}}{(C_1 \| C_2, h) \xrightarrow{\eta} \mathit{fault}} \qquad \frac{(C_2, h) \xrightarrow{\eta} \mathit{fault}}{(C_1 \| C_2, h) \xrightarrow{\eta} \mathit{fault}}$$

$$\frac{f \notin \mathrm{dom}(\eta)}{(f, h) \xrightarrow{\eta} \mathit{fault}} \qquad \frac{(h, \mathtt{fault}) \in c}{(c, h) \xrightarrow{\eta} \mathit{fault}} \qquad \frac{(C, h) \xrightarrow{\eta}{}^{*} \mathit{fault}}{(\langle C \rangle, h) \xrightarrow{\eta} \mathit{fault}}$$

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Worlds (Logical Memory)

$$H_L, pr_L \quad \overset{r_1}{\boxed{H_1, pr_1, I_1}} \quad \cdots \quad \overset{r_n}{\boxed{H_n, pr_n, I_n}}$$

$$w \in \mathsf{World} \stackrel{\text{def}}{=} \{(l, s) \in \mathsf{LState} \times \mathsf{SState} \mid \mathsf{wf}(l, s)\}$$

$$l \in \mathsf{LState} \stackrel{\text{def}}{=} \mathsf{Heap} \times \mathsf{Perm}$$

$$s \in \mathsf{SState} \stackrel{\text{def}}{=} \mathsf{RName} \stackrel{\text{fin}}{\rightharpoonup} (\mathsf{LState} \times \mathsf{IAssn})$$

# Worlds (Logical Memory)

$$H_L, pr_L \quad \overset{r_1}{\boxed{H_1, pr_1, I_1}} \quad \cdots \quad \overset{r_n}{\boxed{H_n, pr_n, I_n}}$$

$$pr \in \mathsf{Perm} \stackrel{\text{def}}{=} \mathsf{Token} \rightarrow [0,1]$$

$$t,\ (r, \gamma, \vec{v})\ \in \mathsf{Token} \stackrel{\text{def}}{=} \mathsf{RName} \times \mathsf{AName} \times \mathsf{Val}^*$$

$$I ::= \gamma(\vec{x})\colon \exists \vec{y}.\ (P \rightsquigarrow Q)\ \mid\ I_1, I_2$$

# Worlds (Logical Memory)

$r_1$ $\qquad$ $r_n$

$$H_L, pr_L \quad \boxed{H_1, pr_1, I_1} \;\cdots\; \boxed{H_n, pr_n, I_n}$$

$$w \oplus w' \stackrel{\text{def}}{=} \begin{cases} (w_{\mathrm{L}} \oplus w'_{\mathrm{L}}, w_{\mathrm{S}}) & \text{if } w_{\mathrm{S}} = w'_{\mathrm{S}} \\ \bot & \text{otherwise.} \end{cases}$$

$$H_L \oplus H'_L = H_L \uplus H'_L$$

$$pr_L \oplus pr'_L = \lambda t.\, pr_L(t) +_{\leq 1} pr'_L(t)$$

# Worlds (Logical Memory)

$H_L, pr_L$ $\quad$ $\boxed{r_1}$ $\boxed{H_1, pr_1, I_1}$ $\cdots$ $\boxed{r_n}$ $\boxed{H_n, pr_n, I_n}$

$$\mathsf{wf}(l, s) \overset{\text{def}}{\iff} \lfloor (l, s) \rfloor \text{ is defined } \wedge$$
$$\forall r, \gamma, \vec{v}.\ \lfloor (l, s) \rfloor_{\mathrm{P}}(r, \gamma, \vec{v}) > 0 \implies (\gamma, \vec{v}) \in \mathrm{adom}((s(r))_2)$$

$$\lfloor (l, s) \rfloor \overset{\text{def}}{=} l \oplus \left( \bigoplus_{r \in \mathrm{dom}(s)} (s(r))_1 \right)$$

$$\mathrm{adom}(\gamma(x_1, \ldots, x_n) : \exists \vec{y}.\ (P \rightsquigarrow Q)) \overset{\text{def}}{=} \{(\gamma, (v_1, \ldots, v_n)) \mid v_i \in \mathsf{Val}\}$$
$$\mathrm{adom}(I_1, I_2) \overset{\text{def}}{=} \mathrm{adom}(I_1) \cup \mathrm{adom}(I_2)$$

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Assertions

$$
\begin{aligned}
&(\mathsf{Assn}) \quad P, Q ::= \mathsf{emp} \mid E_1 \mapsto E_2 \mid P * Q \mid P \mathrel{-\!\circledast} Q \mid \mathsf{false} \mid P \Rightarrow Q \mid \exists x.\ P \mid \circledast x.\ P \mid \\
&\qquad\qquad\qquad \mathsf{all}(I, r) \mid [\,\gamma(E_1, \ldots, E_n)\,]^r_\pi \mid \boxed{P}^{\,r}_{I} \mid \alpha(E_1, \ldots, E_n)
\end{aligned}
$$

$$
(\mathsf{BAssn}) \quad p, q ::= \mathsf{emp} \mid E_1 \mapsto E_2 \mid p * q \mid p \mathrel{-\!\circledast} q \mid \mathsf{false} \mid p \Rightarrow q \mid \exists x.\ p \mid \circledast x.\ p
$$

- $x, y, \ldots$ : Free logical variables
- $\alpha, \beta, \ldots$ : Abstract predicates

# Assertion Semantics

$$\begin{aligned}
(\!|E_1 \mapsto E_2|\!)_{\delta,i} &\stackrel{\text{def}}{=} \left\{ (l, s) \,\middle|\, \begin{array}{r} \mathrm{dom}(l_{\mathrm{H}}) = \{[\![E_1]\!]_i\} \wedge l_{\mathrm{H}}([\![E_1]\!]_i) = [\![E_2]\!]_i \\ \wedge\, l_{\mathrm{P}} = \mathbf{0}_{\mathsf{Perm}} \wedge s \in \mathsf{SState} \end{array} \right\} \\
(\!|\mathsf{emp}|\!)_{\delta,i} &\stackrel{\text{def}}{=} \{((\emptyset, \mathbf{0}_{\mathsf{Perm}}), s) \mid s \in \mathsf{SState}\} \\
(\!|P_1 * P_2|\!)_{\delta,i} &\stackrel{\text{def}}{=} \{w_1 \oplus w_2 \mid w_1 \in (\!|P_1|\!)_{\delta,i} \wedge w_2 \in (\!|P_2|\!)_{\delta,i}\} \\
(\!|P_1 -\!\circledast P_2|\!)_{\delta,i} &\stackrel{\text{def}}{=} \{w \mid \exists w_1, w_2.\, w_2 = w \oplus w_1 \wedge w_1 \in (\!|P_1|\!)_{\delta,i} \wedge w_2 \in (\!|P_2|\!)_{\delta,i}\} \\
(\!|\circledast\, x.\, P|\!)_{\delta,i} &\stackrel{\text{def}}{=} \bigcup_W \left\{ \bigoplus_v W(v) \,\middle|\, \forall v.\, W(v) \in (\!|P|\!)_{\delta,i[x\mapsto v]} \right\}
\end{aligned}$$

$$\delta \in \mathsf{PEnv} \stackrel{\text{def}}{=} \mathsf{PName} \times \mathsf{Val}^* \rightarrow \mathcal{P}(\mathsf{World})$$

$$i \in \mathsf{Interp} \stackrel{\text{def}}{=} \mathsf{Var} \rightarrow \mathsf{Val}$$

# Assertion Semantics

$$
\begin{aligned}
(\!|\mathsf{all}(I,r)|\!) &\stackrel{\text{def}}{=} \left\{(\emptyset,\ \bigoplus_{(\gamma,\vec{v})\in \mathrm{adom}(I)}[(r,\gamma,\vec{v})\mapsto 1])\right\} \\
(\!|[\gamma(E_1,\dots E_n)]^r_\pi|\!)_{\delta,i} &\stackrel{\text{def}}{=} \left\{((\emptyset,[([\![r]\!]_i,\gamma,[\![E_1]\!]_i,\dots,[\![E_n]\!]_i)\mapsto [\![\pi]\!]_i]),s) \,\middle|\, \begin{array}{l} s\in \mathsf{SState}\ \wedge \\ [\![\pi]\!]_i \in (0,1] \end{array}\right\} \\
(\!|\boxed{P}^r_I|\!)_{\delta,i} &\stackrel{\text{def}}{=} \{((\emptyset,\mathbf{0}_{\mathsf{Perm}}),s) \mid \exists l.\ (l,s)\in (\!|P|\!)_{\delta,i} \wedge s([\![r]\!]_i) = (l,[\![I]\!]_i)\} \\
(\!|\alpha(E_1,\dots,E_n)|\!)_{\delta,i} &\stackrel{\text{def}}{=} \delta(\alpha,[\![E_1]\!]_i,\dots,[\![E_n]\!]_i)
\end{aligned}
$$

$$
[\![P]\!]_{\delta,i} \stackrel{\text{def}}{=} \{(l,s)\in (\!|P|\!)_{\delta,i} \mid \mathsf{wf}((l,s))\}
$$

$$
\begin{aligned}
\delta \in \mathsf{PEnv} &\stackrel{\text{def}}{=} \mathsf{PName}\times \mathsf{Val}^* \rightarrow \mathcal{P}(\mathsf{World}) \\
i \in \mathsf{Interp} &\stackrel{\text{def}}{=} \mathsf{Var}\rightarrow \mathsf{Val}
\end{aligned}
$$

# Simple Equalities

$$\boxed{P}^{r}_{I} * \boxed{Q}^{r}_{I} \iff \boxed{P \land Q}^{r}_{I}$$

$$\boxed{\boxed{P}^{r}_{I} * Q}^{r'}_{I'} \iff \boxed{P}^{r}_{I} * \boxed{Q}^{r'}_{I'}$$

## <u>Nesting is necessary</u>

$$\mathsf{Locked}(x) \equiv \exists r.\ [\textsc{Unlock}]^{r}_{1} * \boxed{x \mapsto 1}^{r}_{I(r,x)}$$

$$\boxed{\mathsf{Locked}(x)}^{s}_{K} \iff \boxed{\exists r.\ [\textsc{Unlock}]^{r}_{1} * \boxed{x \mapsto 1}^{r}_{I(r,x)}}^{s}_{K}$$

$$\iff \boxed{\exists r.\ [\textsc{Unlock}]^{r}_{1}}^{s}_{K} * \boxed{x \mapsto 1}^{r}_{I(r,x)}$$

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Interference Semantics

$$\boxed{\ I ::= \gamma(\vec{x}) \colon \exists \vec{y}.\ (P \rightsquigarrow Q) \ \mid\ I_1, I_2\ }$$

$$[\![ I_1, I_2 ]\!]_\delta(r, \gamma, \vec{v}) \stackrel{\text{def}}{=} [\![ I_1 ]\!]_\delta(r, \gamma, \vec{v}) \cup [\![ I_2 ]\!]_\delta(r, \gamma, \vec{v})$$

$$[\![ \gamma(\vec{x}) \colon \exists \vec{y}. (P \rightsquigarrow Q) ]\!]_\delta(r, \gamma', \vec{v}) \stackrel{\text{def}}{=} \left\{ (s, s') \;\middle|\; \begin{array}{r} \gamma' = \gamma \wedge (\forall r' \neq r.\, s(r') = s'(r')) \\ \wedge\, \exists l, l', l_0, I.\, s(r) = (l \oplus l_0, I) \wedge \\ s'(r) = (l' \oplus l_0, I) \wedge \\ \exists \vec{v}'.\, (l, s) \in (\!| P |\!)_{\delta, [\vec{x} \mapsto \vec{v}, \vec{y} \mapsto \vec{v}']} \wedge \\ (l', s') \in (\!| Q |\!)_{\delta, [\vec{x} \mapsto \vec{v}, \vec{y} \mapsto \vec{v}']} \end{array} \right\}$$

$$\delta \in \mathsf{PEnv} \stackrel{\text{def}}{=} \mathsf{PName} \times \mathsf{Val}^* \to \mathcal{P}(\mathsf{World})$$

$$(r, \gamma, \vec{v}) \in \mathsf{Token} \stackrel{\text{def}}{=} \mathsf{RName} \times \mathsf{AName} \times \mathsf{Val}^*$$

$$s \in \mathsf{SState} \stackrel{\text{def}}{=} \mathsf{RName} \stackrel{\text{fin}}{\rightharpoonup} (\mathsf{LState} \times \mathsf{IAssn})$$

# Guarantee

$$G^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \left\{ (w, w') \;\middle|\; \begin{array}{r} \exists r, I, \ell_1, \ell_2.\;\; r \notin \mathrm{dom}(w_{\mathrm{S}}) \wedge w'_{\mathrm{S}} = w_{\mathrm{S}}[r \mapsto (\ell_1, I)] \wedge \\ w_{\mathrm{L}} = \ell_1 \oplus \ell_2 \wedge w'_{\mathrm{L}} = \ell_2 \oplus (\!|\mathsf{all}(I, r)|\!) \end{array} \right\}$$

$$G_\delta \stackrel{\mathrm{def}}{=} \left\{ (w, w') \;\middle|\; \begin{array}{l} ((\exists r, \gamma, \vec{v}.\, (w_{\mathrm{S}}, w'_{\mathrm{S}}) \in [\![(w_{\mathrm{S}}(r))_2]\!]_\delta(r, \gamma, \vec{v}) \wedge \\ (w_{\mathrm{L}})_{\mathrm{P}}(r, \gamma, \vec{v}) > 0) \vee w_{\mathrm{S}} = w'_{\mathrm{S}}) \wedge \lfloor w \rfloor_{\mathrm{P}} = \lfloor w' \rfloor_{\mathrm{P}} \end{array} \right\} \cup G^{\mathrm{c}} \cup (G^{\mathrm{c}})^{-1}$$

$$\overline{G}_\delta \stackrel{\mathrm{def}}{=} G_\delta \cap \{(w, w') \mid \lfloor w \rfloor_H = \lfloor w' \rfloor_H\}$$

$$\widehat{G}_\delta \stackrel{\mathrm{def}}{=} (\overline{G}_\delta)^*; G_\delta; (\overline{G}_\delta)^*$$

What goes wrong with this?

$$\widehat{G_\delta} = (G_\delta)^*$$

# Rely

$$R^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \left\{ (w, w') \;\middle|\; \begin{array}{r} \exists r, \ell, I.\, r \notin \mathrm{dom}(w_{\mathrm{S}}) \wedge w'_{\mathrm{L}} = w_{\mathrm{L}} \wedge w'_{\mathrm{S}} = w_{\mathrm{S}}[r \mapsto (\ell, I)] \wedge \\ \lfloor w' \rfloor \text{ defined} \wedge (\forall \gamma, \vec{v}.\, \lfloor w' \rfloor_{\mathrm{P}}(r, \gamma, \vec{v}) = 0) \end{array} \right\}$$

$$R_{\delta} \stackrel{\mathrm{def}}{=} \left\{ (w, w') \;\middle|\; \begin{array}{r} \exists r, \gamma, \vec{v}.\, (w_{\mathrm{S}}, w'_{\mathrm{S}}) \in [\![ (w_{\mathrm{S}}(r))_2 ]\!]_{\delta}(r, \gamma, \vec{v}) \wedge \\ w_{\mathrm{L}} = w'_{\mathrm{L}} \wedge \lfloor w \rfloor_{\mathrm{P}}(r, \gamma, \vec{v}) < 1 \end{array} \right\} \cup R^{\mathrm{c}} \cup (R^{\mathrm{c}})^{-1}$$

# Stability

**Definition 4.2** (Stability). $\mathsf{stable}_\delta(P)$ holds iff for all $w$, $w'$ and $i$, if $w \in [\![P]\!]_{\delta,i}$ and $(w, w') \in R_\delta$, then $w' \in [\![P]\!]_{\delta,i}$.

Similarly, a predicate environment is stable if and only if all the predicates it defines are stable.

**Definition 4.3** (Predicate Environment Stability). $\mathsf{pstable}(\delta)$ holds iff for all $X \in \mathrm{ran}(\delta)$, for all $w$ and $w'$, if $w \in X$ and $(w, w') \in R_\delta$, then $w' \in X$.

# Repartitioning

**Definition 4.1** (Repartitioning). $P \Longrightarrow_{\delta}^{\{p\}\{q\}} Q$ holds if and only if, for every variable interpretation $i$ and world $w_1$ in $[\![P]\!]_{\delta,i}$, there exists a heap $h_1$ in $[\![p]\!]_i$ and a residual heap $h'$ such that

- $h_1 \oplus h' = \lfloor w_1 \rfloor_H$; and such that $h_2 \oplus h'$ defined ?
- for every heap $h_2$ in $[\![q]\!]_i$, there exists a world $w_2$ in $[\![Q]\!]_{\delta,i}$ such that
  - $h_2 \oplus h' = \lfloor w_2 \rfloor_H$; and
  - the update is allowed by the guarantee: that is, $(w_1, w_2) \in \widehat{G}_\delta$.

We write $P \Longrightarrow_\delta Q$ as a shorthand for $P \Longrightarrow_{\delta}^{\{\mathsf{emp}\}\{\mathsf{emp}\}} Q$.

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Judgments

$$\Delta ; \Gamma \vdash \{P\}\, C\, \{Q\}$$

$$\Delta ; \Gamma \models \{P\}\, C\, \{Q\}$$

can't $\forall \vec{x}.\, \alpha(\vec{x}) \equiv P$ be a sugar for $\forall \vec{x}.\, \alpha(\vec{x}) \Longrightarrow P, \forall \vec{x}.\, P \Longrightarrow \alpha(\vec{x})$ ?

$$\Delta ::= \varnothing \mid \Delta, \forall \vec{x}.\, P \implies Q \overbrace{\mid \Delta, \forall \vec{x}.\, \alpha(\vec{x}) \equiv P \quad \text{s.t. } \alpha \notin \Delta}^{\text{Why do we need this?}}$$

$$\Gamma ::= \varnothing \mid \Gamma, \{P\} f \{Q\} \quad \text{s.t. } f \notin \Gamma$$

# Configuration Safety

**Definition 4.4** (Configuration safety). $C, w, \eta, \delta, i, Q$ safe$_0$ always holds; and $C, w, \eta, \delta, i,$ safe$_{n+1}$ iff the following four conditions hold:

1. $\forall w'$, if $(w, w') \in (R_\delta)^*$ then $C, w', \eta, \delta, i, Q$ safe$_n$;

2. $\neg((C, \lfloor w \rfloor_H) \xrightarrow{\eta} \mathit{fault})$;

3. $\forall C', h'$, if $(C, \lfloor w \rfloor_H) \xrightarrow{\eta} (C', h')$, then there $\exists w'$ such that $(w, w') \in \widehat{G}_\delta$, $h' = \lfloor w' \rfloor_H$ and $C', w', \eta, \delta, i, Q$ safe$_n$; and

4. if $C$=**skip**, then $\exists w'$ such that $\lfloor w \rfloor_H = \lfloor w' \rfloor_H$, $(w, w') \in \widehat{G}_\delta$, and $w' \in [\![Q]\!]_{\delta,i}$.

$$\delta \in \mathsf{PEnv} \stackrel{\text{def}}{=} \mathsf{PName} \times \mathsf{Val}^* \longrightarrow \mathcal{P}(\mathsf{World})$$

$$i \in \mathsf{Interp} \stackrel{\text{def}}{=} \mathsf{Var} \longrightarrow \mathsf{Val}$$

$$\eta \in \mathsf{FEnv} \stackrel{\text{def}}{=} \mathsf{FName} \longrightarrow \mathsf{Cmd}$$

# Judgment Semantics

**Definition 4.5** (Judgement Semantics). $\Delta; \Gamma \models \{P\}\, C\, \{Q\}$ holds iff

$$\forall i, n.\ \forall \delta \in [\![\Delta]\!]\ .\ \forall \eta \in [\![\Gamma]\!]_{n,\delta,i}\ .\ \forall w \in [\![P]\!]_{\delta,i}\ .\ C, w, \eta, \delta, i, Q\ \mathsf{safe}_{n+1}$$

Step-indexing:

- Easy for dealing with recursive functions.
- But, might be problematic with memoization.
- There might be a coinductive solution.

$$[\![\varnothing]\!] \stackrel{\text{def}}{=} \{\delta \mid \mathsf{pstable}(\delta)\}$$

$$\cancel{[\![\Delta,\ \forall \vec{x}.\, \alpha(\vec{x}) \equiv P]\!] \stackrel{\text{def}}{=} [\![\Delta]\!] \cap \{\delta \mid \mathsf{pstable}(\delta) \wedge \forall \vec{v}.\ \delta(\alpha, \vec{v}) = [\![P]\!]_{\delta,[\vec{x}\mapsto\vec{v}]}\}}$$

$$[\![\Delta,\ \forall \vec{x}.\, P \Rightarrow Q]\!] \stackrel{\text{def}}{=} [\![\Delta]\!] \cap \{\delta \mid \mathsf{pstable}(\delta) \wedge \forall \vec{v}.\ [\![P]\!]_{\delta,[\vec{x}\mapsto\vec{v}]} \subseteq [\![Q]\!]_{\delta,[\vec{x}\mapsto\vec{v}]}\}$$

$$[\![\Gamma]\!]_{n,\delta,i} \stackrel{\text{def}}{=} \{\eta \mid \forall \{P\} f \{Q\} \in \Gamma.\ \forall w \in [\![P]\!]_{\delta,i}\ .\ \eta(f), w, \eta, \delta, i, Q\ \mathsf{safe}_n\}$$

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Proof Rules

$$\frac{}{\Delta;\Gamma \vdash \{P\}\ \mathbf{skip}\ \{P\}}\ (\textsc{Skip}) \qquad \frac{\Delta;\Gamma \vdash \{P\}\ C_1\ \{P'\} \quad \Delta;\Gamma \vdash \{P\}\ C_2\ \{P'\}}{\Delta;\Gamma \vdash \{P\}\ C_1 + C_2\ \{P'\}}\ (\textsc{Choice})$$

$$\frac{\Delta;\Gamma \vdash \{P\}\ C_1\ \{P''\} \quad \Delta;\Gamma \vdash \{P''\}\ C_2\ \{P'\}}{\Delta;\Gamma \vdash \{P\}\ C_1;\ C_2\ \{P'\}}\ (\textsc{Seq}) \qquad \frac{\Delta;\Gamma \vdash \{P\}\ C\ \{P\}}{\Delta;\Gamma \vdash \{P\}\ C^*\ \{P\}}\ (\textsc{Loop})$$

$$\frac{\Delta;\Gamma \vdash \{P_1\}\ C_1\ \{Q_1\} \quad \Delta;\Gamma \vdash \{P_2\}\ C_2\ \{Q_2\}}{\Delta;\Gamma \vdash \{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (\textsc{Par}) \qquad \frac{\vdash_{\mathsf{SL}} \{p\}\ C\ \{q\}}{\Delta;\Gamma \vdash \{p\}\ C\ \{q\}}\ (\textsc{Prim})$$

$$\frac{\Delta;\Gamma \vdash \{P_1\}\ C\ \{Q\} \quad \Delta;\Gamma \vdash \{P_2\}\ C\ \{Q\}}{\Delta;\Gamma \vdash \{P_1 \vee P_2\}\ C\ \{Q\}}\ (\textsc{Disj}) \qquad \frac{x \notin \mathsf{fv}(\Delta,\Gamma,P,C,Q) \quad \Delta;\Gamma \vdash \{P\}\ C\ \{Q\}}{\Delta;\Gamma \vdash \{\exists x.\, P\}\ C\ \{Q\}}\ (\textsc{Ex})$$

All rules assume that the pre- and post-conditions of their judgments are stable.

# Proof Rules

$$\frac{\vdash_{\mathsf{SL}} \{p\}\ C\ \{q\} \qquad \Delta \vdash P \Longrightarrow^{\{p\}\{q\}} Q}{\Delta; \Gamma \vdash \{P\}\ \langle C \rangle\ \{Q\}}\ (\textsc{Atomic}) \qquad \frac{\{P\}\ f\ \{Q\} \in \Gamma}{\Delta; \Gamma \vdash \{P\}\ f\ \{Q\}}\ (\textsc{Call})$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash \{P\}\ C\ \{Q\} \\ \Delta \vdash \mathsf{stable}(R)\end{array}}{\Delta; \Gamma \vdash \{P * R\}\ C\ \{Q * R\}}\ (\textsc{Frame}) \qquad \frac{\begin{array}{c}\Delta; \Gamma \vdash \{P'\}\ C\ \{Q'\} \\ \Delta \vdash P \Longrightarrow P' \qquad \Delta \vdash Q' \Longrightarrow Q\end{array}}{\Delta; \Gamma \vdash \{P\}\ C\ \{Q\}}\ (\textsc{Conseq})$$

$\alpha$ occurs positively in $R$

$$\frac{\Delta \vdash \Delta' \qquad \Delta'; \Gamma \vdash \{P\}C\{Q\}}{\Delta; \Gamma \vdash \{P\}C\{Q\}}\ (\textsc{Pred-I}) \qquad \frac{\begin{array}{c}\Delta \vdash \mathsf{stable}(R) \qquad \alpha \notin \Delta, \Gamma, P, Q \\ \Delta, (\forall \vec{x}.\ \alpha(\vec{x}) \equiv R); \Gamma \vdash \{P\}C\{Q\}\end{array}}{\Delta; \Gamma \vdash \{P\}C\{Q\}}\ (\textsc{Pred-E})$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash \{P_1\}\ C_1\ \{Q_1\} \quad \ldots \quad \Delta; \Gamma \vdash \{P_n\}\ C_n\ \{Q_n\} \\ \Delta; \{P_1\}\ f_1\ \{Q_1\}, \ldots, \{P_n\}\ f_n\ \{Q_n\}, \Gamma \vdash \{P\}\ C\ \{Q\}\end{array}}{\Delta; \Gamma \vdash \{P\}\ \mathrm{let}\ f_1 = C_1 \ldots f_n = C_n\ \mathrm{in}\ C\ \{Q\}}\ (\textsc{Let})$$

Recursive Let is derivable.
But we need a rule
for eliminating unused $\Gamma$.

$$\Delta \vdash P \Longrightarrow^{\{p\}\{q\}} Q \text{ means } \forall \delta \in [\![\Delta]\!]\,.\, P \Longrightarrow_{\delta}^{\{p\}\{q\}} Q$$

$$\Delta \vdash \mathsf{stable}(P) \text{ means } \forall \delta \in [\![\Delta]\!]\,.\, \mathsf{stable}_{\delta}(P)$$

$$\Delta \vdash \Delta' \text{ means } [\![\Delta]\!] \subseteq [\![\Delta']\!].$$

All rules assume that the pre- and post-conditions of their judgments are stable.

# Frame Rule Bug ?

$$\frac{\Delta;\Gamma \vdash \{P\}\ C\ \{Q\} \qquad \Delta \vdash \mathsf{stable}(R)}{\Delta;\Gamma \vdash \{P * R\}\ C\ \{Q * R\}}\ \textsc{(Frame)}$$

region name conflict ?

$$\vdash \{x \mapsto 0\}\ \mathrm{skip}\ \left\{\boxed{x \mapsto 0}^{r}_{\emptyset}\right\}$$

$$\vdash \mathrm{stable}\left(\boxed{y \mapsto 1}^{r}_{\emptyset}\right)$$

---

$$\left\{\mathrm{x} \mapsto 0 * \boxed{y \mapsto 1}^{r}_{\emptyset}\right\}\ \mathrm{skip}\ \left\{\boxed{x \mapsto 0}^{r}_{\emptyset} * \boxed{y \mapsto 1}^{r}_{\emptyset}\right\}$$

My Thought (Maybe Wrong):

- Clients do not know which region names will be used by modules.
- So, if clients use some shared regions, how do they know the region names are not used by other abstract modules?
- In particular, when you frame in abstract predicates using Frame Rule, there is no guarantee that there are no region name conflicts.

# Derived Rule for Module

$$\dfrac{\Delta \vdash \{P_1\}C_1\{Q_1\} \qquad \cdots \qquad \dfrac{\Delta \vdash \Delta' \qquad \Delta'; \{P_1\}f_1\{Q_1\}, \ldots \vdash \{P\}C\{Q\}}{\Delta; \{P_1\}f_1\{Q_1\}, \ldots \vdash \{P\}C\{Q\}}\ \textsc{Pred-I}}{\dfrac{\Delta \vdash \{P\}\ \text{let}\ f_1 = C_1 \ldots f_n = C_n\ \text{in}\ C\ \{Q\}}{\Delta' \vdash \{P\}\ \text{let}\ f_1 = C_1 \ldots f_n = C_n\ \text{in}\ C\ \{Q\}}\ \textsc{Pred-E}}\ \textsc{Let}$$

# Soundness

**Theorem 4.6** (Soundness). *If* $\Delta; \Gamma \vdash \{P\}\, C\, \{Q\}$, *then* $\Delta; \Gamma \models \{P\}\, C\, \{Q\}$.

**Lemma 4.7** (Abstract state locality). *If* $(C, \lfloor w_1 \oplus w_2 \rfloor_H) \xrightarrow{\eta} (C', h)$ *and* $C, w_1, \eta, \delta, i, Q$ *safe*$_n$, *then* $\exists w'_1, w'_2$ *such that* $(C, \lfloor w_1 \rfloor_H) \xrightarrow{\eta} (C', \lfloor w'_1 \rfloor_H)$, $h = \lfloor w'_1 \oplus w'_2 \rfloor_H$, $(w_1, w'_1) \in \widehat{G}_\delta$, *and* $(w_2, w'_2) \in (R_\delta)^*$.

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Lock Specification for Client

$\Gamma$ :

$$\{\mathsf{isLock}(\mathtt{x})\} \quad \mathtt{lock}(\mathtt{x}) \quad \{\mathsf{isLock}(\mathtt{x}) * \mathsf{Locked}(\mathtt{x})\}$$

$$\{\mathsf{Locked}(\mathtt{x})\} \quad \mathtt{unlock}(\mathtt{x}) \quad \{\mathsf{emp}\}$$

$$\{\mathsf{emp}\} \quad \mathtt{makelock}(\mathtt{n}) \quad \left\{ \begin{array}{c} \exists x.\, \mathrm{ret} = x \wedge \mathsf{isLock}(x) * \mathsf{Locked}(x) \\ * (x+1) \mapsto \_ * \ldots * (x+\mathtt{n}) \mapsto \_ \end{array} \right\}$$

$\Delta$ :

$$\mathsf{isLock}(x) \iff \mathsf{isLock}(x) * \mathsf{isLock}(x)$$

$$\mathsf{Locked}(x) * \mathsf{Locked}(x) \iff \mathsf{false}$$

# Lock Specification for Module

Additional Axioms $\Delta'$ :

$$
\begin{aligned}
\mathsf{isLock}(x) &\equiv \exists r.\, \exists \pi.\, [\textsc{Lock}]^r_\pi * \boxed{(x \mapsto 0 * [\textsc{Unlock}]^r_1) \vee x \mapsto 1}^{\,r}_{I(r,x)} \\
\mathsf{Locked}(x) &\equiv \exists r.\, [\textsc{Unlock}]^r_1 * \boxed{x \mapsto 1}^{\,r}_{I(r,x)}
\end{aligned}
$$

$$
I(r,x) \stackrel{\text{def}}{=} \left( \begin{array}{rccc} \textsc{Lock}: & x \mapsto 0 * [\textsc{Unlock}]^r_1 & \rightsquigarrow & x \mapsto 1, \\ \textsc{Unlock}: & x \mapsto 1 & \rightsquigarrow & x \mapsto 0 * [\textsc{Unlock}]^r_1 \end{array} \right)
$$

# Lock Verification

$\left\{\mathsf{isLock}(\mathrm{x})\right\}$

```
lock(x) {
```

$\left\{\exists r.\, \pi.\, [\textsc{Lock}]^r_\pi * \boxed{(\mathrm{x} \mapsto 0 * [\textsc{Unlock}]^r_1) \vee \mathrm{x} \mapsto 1}^r_{I(r,\mathrm{x})}\right\}$

```
  local b;
  do
```

$\left\{\exists r.\, \pi.\, [\textsc{Lock}]^r_\pi * \boxed{(\mathrm{x} \mapsto 0 * [\textsc{Unlock}]^r_1) \vee \mathrm{x} \mapsto 1}^r_{I(r,\mathrm{x})}\right\}$

$\langle$`b := `$\neg$`CAS(&x,0,1)`$\rangle$;

$\left\{\begin{array}{l}\exists r.\, \pi.\, \left(\boxed{\mathrm{x} \mapsto 1}^r_{I(r,\mathrm{x})} * [\textsc{Lock}]^r_\pi * [\textsc{Unlock}]^r_1 * \mathsf{b} = \mathsf{false}\right) \vee \\ \left(\boxed{(\mathrm{x} \mapsto 0 * [\textsc{Unlock}]^r_1) \vee \mathrm{x} \mapsto 1}^r_{I(r,\mathrm{x})} * [\textsc{Lock}]^r_\pi * \mathsf{b} = \mathsf{true}\right)\end{array}\right\}$

```
  while(b)
```

$\left\{\exists r.\, \boxed{\mathrm{x} \mapsto 1}^r_{I(r,\mathrm{x})} * [\textsc{Lock}]^r_\pi * [\textsc{Unlock}]^r_1 * \mathsf{b} = \mathsf{false}\right\}$

```
}
```

$\left\{\mathsf{isLock}(\mathrm{x}) * \mathsf{Locked}(\mathrm{x})\right\}$

# Lock Verification

$\left\{\mathsf{Locked}(\mathrm{x})\right\}$

```
unlock(x) {
```

$\left\{\exists r.\, [\textsc{Unlock}]_1^r * \boxed{\mathrm{x} \mapsto 1}_{I(r,\mathrm{x})}^r\right\}$

$\langle$`[x] := 0`$\rangle$;

$\left\{\exists r.\, \boxed{\mathrm{x} \mapsto 0 * [\textsc{Unlock}]_1^r}_{I(r,\mathrm{x})}^r\right\}$

*// Stabilise the assertion.*

$\left\{\exists r.\, \boxed{(\mathrm{x} \mapsto 0 * [\textsc{Unlock}]_1^r) \vee \mathrm{x} \mapsto 1}_{I(r,\mathrm{x})}^r\right\}$

```
}
```

$\left\{\mathsf{emp}\right\}$

# Lock Verification

$\{\mathsf{emp}\}$

```
makelock(n) {
  local x := alloc(n + 1);
```

$\{\mathrm{x} \mapsto \_ * (\mathrm{x}+1) \mapsto \_ * \ldots * (\mathrm{x}+\mathrm{n}) \mapsto \_\}$

```
  [x] := 1;
```

$\{\mathrm{x} \mapsto 1 * (\mathrm{x}+1) \mapsto \_ * \ldots * (\mathrm{x}+\mathrm{n}) \mapsto \_\}$

```
  // Create shared lock region.
```

$\left\{\exists r. \boxed{\mathrm{x} \mapsto 1}^{r}_{I(r,\mathrm{x})} * [\textsc{Lock}]^{r}_{1} * [\textsc{Unlock}]^{r}_{1} * (\mathrm{x}+1) \mapsto \_ * \ldots * (\mathrm{x}+\mathrm{n}) \mapsto \_\right\}$

```
  return x;
}
```

$\{\exists x. \mathrm{ret} = x \wedge \mathsf{isLock}(x) * \mathsf{Locked}(x) * (x+1) \mapsto \_ * \ldots * (x+\mathrm{n}) \mapsto \_\}$

# Set Specification for Client

$\Gamma$ :

$$\begin{array}{ccc}
\{\mathsf{in}(\mathtt{h},\mathtt{v})\} & \mathtt{contains}(\mathtt{h},\mathtt{v}) & \{\mathsf{in}(\mathtt{h},\mathtt{v}) * \mathrm{ret} = \mathsf{true}\} \\
\{\mathsf{out}(\mathtt{h},\mathtt{v})\} & \mathtt{contains}(\mathtt{h},\mathtt{v}) & \{\mathsf{out}(\mathtt{h},\mathtt{v}) * \mathrm{ret} = \mathsf{false}\} \\
\{\mathsf{own}(\mathtt{h},\mathtt{v})\} & \mathtt{add}(\mathtt{h},\mathtt{v}) & \{\mathsf{in}(\mathtt{h},\mathtt{v})\} \\
\{\mathsf{own}(\mathtt{h},\mathtt{v})\} & \mathtt{remove}(\mathtt{h},\mathtt{v}) & \{\mathsf{out}(\mathtt{h},\mathtt{v})\}
\end{array}$$

$\Delta$ :

$$\mathsf{own}(h,v) \; * \; \mathsf{own}(h,v) \implies \mathsf{false}$$

where $\mathrm{own}(h,v) \coloneqq \mathrm{in}(h,v) \vee \mathrm{out}(h,v)$

$$\{\mathrm{emp}\} \quad \mathrm{mkemp}(\,) \quad \{\circledast_v.\, \mathrm{out}(\mathrm{ret},v)\}$$

is needed to be a concurrent set

# External Modules for Set

## Lock Module

$$\{\mathsf{isLock}(\mathrm{x})\} \quad \mathtt{lock}(\mathrm{x}) \quad \{\mathsf{isLock}(\mathrm{x}) * \mathsf{Locked}(\mathrm{x})\}$$

$$\{\mathsf{Locked}(\mathrm{x})\} \quad \mathtt{unlock}(\mathrm{x}) \quad \{\mathsf{emp}\}$$

$$\{\mathsf{emp}\} \quad \mathtt{makelock}(\mathtt{n}) \quad \left\{ \begin{array}{l} \exists x.\, \mathrm{ret} = x \wedge \mathsf{isLock}(x) * \mathsf{Locked}(x) \\ \quad * (x+1) \mapsto \_ * \ldots * (x+\mathtt{n}) \mapsto \_ \end{array} \right\}$$

$$\mathsf{isLock}(x) \iff \mathsf{isLock}(x) * \mathsf{isLock}(x)$$

$$\mathsf{Locked}(x) * \mathsf{Locked}(x) \iff \mathsf{false}$$

## Sequential Set Module

$$\{\mathsf{Set}(\mathrm{h}, vs)\} \quad \mathtt{scontains}(\mathrm{h}, \mathrm{v}) \quad \{\mathsf{Set}(\mathrm{h}, vs) * \mathrm{ret} = (\mathrm{v} \in vs)\}$$

$$\{\mathsf{Set}(\mathrm{h}, vs)\} \quad \mathtt{sadd}(\mathrm{h}, \mathrm{v}) \quad \{\mathsf{Set}(\mathrm{h}, \{\mathrm{v}\} \cup vs)\}$$

$$\{\mathsf{Set}(\mathrm{h}, vs)\} \quad \mathtt{sremove}(\mathrm{h}, \mathrm{v}) \quad \{\mathsf{Set}(\mathrm{h}, vs \setminus \{\mathrm{v}\})\}$$

$$\{\mathrm{emp}\} \quad \mathrm{mkemp}(\,) \quad \{\mathrm{Set}(\mathrm{ret}, \emptyset)\}$$

# Set Specification for Module

Additional Axioms $\Delta'$ :

$$\mathsf{in}(h,v) \equiv \exists s.\ \mathsf{isLock}(h.\mathrm{lock}) * [\mathrm{SC{\small HANGE}}(v)]_1^s * \boxed{P_\in(h,v,s)}_{C(s,h)}^s$$

$$\mathsf{out}(h,v) \equiv \exists s.\ \mathsf{isLock}(h.\mathrm{lock}) * [\mathrm{SC{\small HANGE}}(v)]_1^s * \boxed{P_\notin(h,v,s)}_{C(s,h)}^s$$

$$C(s,h) \stackrel{\text{def}}{=} \left( \begin{array}{l} \mathrm{SC{\small HANGE}}(v): \begin{pmatrix} \exists vs, ws.\ \mathsf{Set}(h.\mathrm{set}, vs) \\ * [\mathrm{SG{\small AP}}(ws)]_1^s \wedge \\ vs \setminus \{v\} = ws \setminus \{v\} \end{pmatrix} \rightsquigarrow \mathsf{Locked}(h.\mathrm{lock}), \\ \quad \mathrm{SG{\small AP}}(ws):\ \mathsf{Locked}(h.\mathrm{lock}) \rightsquigarrow \mathsf{Set}(h.\mathrm{set}, ws) * [\mathrm{SG{\small AP}}(ws)]_1^s \end{array} \right)$$

$$\mathsf{allgaps}(s) \equiv \circledast ws.\ [\mathrm{SG{\small AP}}(ws)]_1^s$$

$$P_\triangleleft(h,v,s) \equiv \exists vs.\ v \triangleleft vs \wedge \begin{pmatrix} (\mathsf{allgaps}(s) * \mathsf{Set}(h.\mathrm{set}, vs)) \\ \vee\ \mathsf{Locked}(h.\mathrm{lock}) * ([\mathrm{SG{\small AP}}(vs)]_1^s \mathrel{-\!\!\circledast} \mathsf{allgaps}(s)) \end{pmatrix}$$

where $\triangleleft = \in$ or $\triangleleft = \notin$

# Set Verification

$$\left\{\mathsf{out}(\mathtt{h}, \mathtt{v})\right\}$$

```
add(h,v)
```

$$\left\{\exists s.\, \mathsf{isLock}(\mathtt{h.lock}) * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \boxed{P_{\notin}(\mathtt{h}, \mathtt{v}, s)}^{s}_{C(s,\mathtt{h})}\right\}$$

```
lock(h.lock);
```

$$\left\{\exists s.\, \mathsf{isLock}(\mathtt{h.lock}) * \mathsf{Locked}(\mathtt{h.lock}) * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \boxed{P_{\notin}(\mathtt{h}, \mathtt{v}, s)}^{s}_{C(s,\mathtt{h})}\right\}$$

*// use* SChange *to extract* Set *predicate and* SGap *permission.*

$$\left\{\begin{array}{r}\exists s, vs.\, \mathsf{isLock}(\mathtt{h.lock}) * [\mathrm{SG{\scriptsize AP}}(vs \cup \{\mathtt{v}\})]_1^s * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \mathsf{Set}(\mathtt{h.set}, vs) \\ * \boxed{\mathsf{Locked}(\mathtt{h.lock}) * ([\mathrm{SG{\scriptsize AP}}(vs \cup \{\mathtt{v}\})]_1^s \mathrel{-\!\!\circledast} \mathsf{allgaps}(s))}^{s}_{C(s,\mathtt{h})}\end{array}\right\}$$

```
sadd(h.set,v);
```

$$\left\{\begin{array}{r}\exists s, vs.\, \mathsf{isLock}(\mathtt{h.lock}) * [\mathrm{SG{\scriptsize AP}}(vs \cup \{\mathtt{v}\})]_1^s * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \mathsf{Set}(\mathtt{h.set}, vs \cup \{\mathtt{v}\}) \\ * \boxed{\mathsf{Locked}(\mathtt{h.lock}) * ([\mathrm{SG{\scriptsize AP}}(vs \cup \{\mathtt{v}\})]_1^s \mathrel{-\!\!\circledast} \mathsf{allgaps}(s))}^{s}_{C(s,\mathtt{h})}\end{array}\right\}$$

*// use* SGap *permission to put back* Set *and* SGap *permission.*

$$\left\{\exists s.\, \mathsf{isLock}(\mathtt{h.lock}) * \mathsf{Locked}(\mathtt{h.lock}) * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \boxed{P_{\in}(\mathtt{h}, \mathtt{v}, s)}^{s}_{C(s,\mathtt{h})}\right\}$$

```
unlock(h.lock);
```

$$\left\{\exists s.\, \mathsf{isLock}(\mathtt{h.lock}) * [\mathrm{SC{\scriptsize HANGE}}(\mathtt{v})]_1^s * \boxed{P_{\in}(\mathtt{h}, \mathtt{v}, s)}^{s}_{C(s,\mathtt{h})}\right\}$$

```
}
```

$$\left\{\mathsf{in}(\mathtt{h}, \mathtt{v})\right\}$$

# Outline

- Language and Operational Semantics
- Worlds (Logical Memory)
- Assertions
- Interferences
- Judgments
- Proof Rules and Soundness
- Examples
- Conclusions and Related Work

# Conclusions and Related Work

- Abstract Predicate
- Deny-Guarantee
- Context Logic
- B. Jacobs and F. Piessens. Modular full functional specification and verification of lock-free data structures.
- Alternative Approach: Linearizablility