1 Syntax

$$
\begin{array}{ll}
\mathit{program} & ::= \overline{\mathit{cls}}\ \overline{s} \\
\mathit{cls} & ::= \texttt{class}\ C\ \{\overline{\mathit{field}}\ \overline{\mathit{method}}\} \\
\mathit{field} & ::= T\ f; \\
\mathit{method} & ::= T\ m(T\ x)\ \mathit{contract}\ \{\overline{s}\} \\
\mathit{contract} & ::= \texttt{requires}\ \phi;\ \texttt{ensures}\ \phi; \\
T & ::= \texttt{int} \mid C \\
s & ::= x.f := y; \mid x := e; \mid x := \texttt{new}\ C; \mid x := y.m(z); \\
 & \mid \texttt{return}\ x; \mid \texttt{assert}\ \phi; \mid \texttt{release}\ \phi; \mid T\ x; \\
\phi & ::= \texttt{true} \mid e = e \mid e \neq e \mid \texttt{acc}(e.f) \mid \phi * \phi \\
e & ::= v \mid x \mid e.f \\
x & ::= \texttt{this} \mid \texttt{result} \mid \langle \mathit{other} \rangle \\
v & ::= o \mid n \mid \texttt{null} \\
n & \in\ \mathbb{Z} \\
 & \\
H & \in\ (o \rightharpoonup (C, \overline{(f \rightharpoonup v)})) \\
\rho & \in\ (x \rightharpoonup v) \\
\Gamma & \in\ (x \rightharpoonup T) \\
A_s & ::= \overline{(e, f)} \\
A_d & ::= \overline{(o, f)} \\
S & ::= (\rho, A_d, \overline{s}) \cdot S \mid \mathit{nil}
\end{array}
$$

2 Assumptions

All the rules in the following sections are implicitly parameterized over a *programp* that is well-formed.

2.0.1 Well-formed program ($\mathit{program}\ \texttt{OK}$)

$$\frac{\overline{\mathit{cls}_i\ \texttt{OK}}}{(\overline{\mathit{cls}_i}\ \overline{s})\ \texttt{OK}}\ \textsc{OKProgram}$$

2.0.2 Well-formed class ($\mathit{cls}\ \texttt{OK}$)

$$\frac{\text{unique } \mathit{field}\text{-names} \qquad \text{unique } \mathit{method}\text{-names} \qquad \overline{\mathit{method}_i\ \texttt{OK in}\ C}}{(\texttt{class}\ C\ \{\overline{\mathit{field}_i}\ \overline{\mathit{method}_i}\})\ \texttt{OK}}\ \textsc{OKClass}$$

2.0.3 Well-formed method ($\mathit{method}\ \texttt{OK in}\ C$)

$$\frac{\begin{array}{c} FV(\phi_1) \subseteq \{x, \texttt{this}\} \qquad FV(\phi_2) \subseteq \{x, \texttt{this}, \texttt{result}\} \\ x : T_x, \texttt{this} : C, \texttt{result} : T_m \vdash \{\phi_1\}\overline{s}\{\phi_2\} \qquad \emptyset \vdash_{\texttt{sfrm}} \phi_1 \qquad \emptyset \vdash_{\texttt{sfrm}} \phi_2 \qquad \overline{\neg \mathit{writesTo}(s_i, x)} \end{array}}{(T_m\ m(T_x\ x)\ \texttt{requires}\ \phi_1;\ \texttt{ensures}\ \phi_2;\ \{\overline{s}\})\ \texttt{OK in}\ C}\ \textsc{OKMethod}$$

3 Static semantics

3.1 Expressions ($A_s \vdash_{\texttt{sfrm}} e$)

$$\frac{}{A \vdash_{\texttt{sfrm}} x}\ \textsc{WFVar}$$

$$\frac{}{A \vdash_{\texttt{sfrm}} v} \textsc{WFValue}$$

$$\frac{(e, f) \in A \qquad A \vdash_{\texttt{sfrm}} e}{A \vdash_{\texttt{sfrm}} e.f} \textsc{WFField}$$

3.2 Formulas ($A_s \vdash_{\texttt{sfrm}} \phi$)

$$\frac{}{A \vdash_{\texttt{sfrm}} \texttt{true}} \textsc{WFTrue}$$

$$\frac{A \vdash_{\texttt{sfrm}} e_1 \qquad A \vdash_{\texttt{sfrm}} e_2}{A \vdash_{\texttt{sfrm}} (e_1 = e_2)} \textsc{WFEqual}$$

$$\frac{A \vdash_{\texttt{sfrm}} e_1 \qquad A \vdash_{\texttt{sfrm}} e_2}{A \vdash_{\texttt{sfrm}} (e_1 \neq e_2)} \textsc{WFNEqual}$$

$$\frac{A \vdash_{\texttt{sfrm}} e}{A \vdash_{\texttt{sfrm}} \texttt{acc}(e.f)} \textsc{WFAcc}$$

$$\frac{A_s \vdash_{\texttt{sfrm}} \phi_1 \qquad A_s \cup \lfloor \phi_1 \rfloor \vdash_{\texttt{sfrm}} \phi_2}{A_s \vdash_{\texttt{sfrm}} \phi_1 * \phi_2} \textsc{WFSepOp}$$

3.2.1 Implication ($\phi_1 \stackrel{.}{\Longrightarrow} \phi_2$)

Conservative approx. of $\phi_1 \implies \phi_2$.

3.3 Footprint ($\lfloor \phi \rfloor = A_s$)

$$\begin{aligned}
\lfloor \texttt{true} \rfloor &= \emptyset \\
\lfloor e_1 = e_2 \rfloor &= \emptyset \\
\lfloor e_1 \neq e_2 \rfloor &= \emptyset \\
\lfloor \texttt{acc}(e.f) \rfloor &= \{(e, f)\} \\
\lfloor \phi_1 * \phi_2 \rfloor &= \lfloor \phi_1 \rfloor \cup \lfloor \phi_2 \rfloor
\end{aligned}$$

3.4 Type ($\Gamma \vdash e : T$)

$$\frac{}{\Gamma \vdash n : \texttt{int}} \textsc{STValNum}$$

$$\frac{}{\Gamma \vdash \texttt{null} : T} \textsc{STValNull}$$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \textsc{STVar}$$

$$\frac{\Gamma \vdash e : C \qquad \vdash C.f : T}{\Gamma \vdash e.f : T} \textsc{STField}$$

3.5 Hoare ($\Gamma \vdash \{\phi\}\overline{s}\{\phi\}$)

$$\frac{\phi \implies \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi' \qquad x \notin FV(\phi') \qquad \Gamma \vdash x : C \qquad \mathtt{fields}(C) = \overline{f}}{\Gamma \vdash \{\phi\} x := \mathtt{new}\ C \{\phi' * (x \neq \mathtt{null}) * \overline{\mathtt{acc}(x, f_i)}\}} \text{ HNewObj}$$

$$\frac{\phi \implies \mathtt{acc}(x.f) * (x \neq \mathtt{null}) * \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi' \qquad \Gamma \vdash x : C \qquad \Gamma \vdash y : T \qquad \vdash C.f : T}{\Gamma \vdash \{\phi\} x.f := y \{\phi' * \mathtt{acc}(x.f) * (x \neq \mathtt{null}) * (x.f = y)\}} \text{ HFieldAssign}$$

$$\frac{\phi \implies \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi' \qquad x \notin FV(\phi') \qquad x \notin FV(e) \qquad \Gamma \vdash x : T \qquad \Gamma \vdash e : T \qquad [\![e]\!] \subseteq \phi'}{\Gamma \vdash \{\phi\} x := e \{\phi' * (x = e)\}} \text{ HVarAssign}$$

$$\frac{\phi \implies \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi' \qquad \mathtt{result} \notin FV(\phi') \qquad \Gamma \vdash x : T \qquad \Gamma \vdash \mathtt{result} : T}{\Gamma \vdash \{\phi\} \mathtt{return}\ x \{\phi' * (\mathtt{result} = x)\}} \text{ HReturn}$$

$$\frac{\begin{array}{c}\Gamma \vdash y : C \qquad \mathtt{mmethod}(C, m) = T_r\ m(T_p\ z)\ \mathtt{requires}\ \phi_{pre};\ \mathtt{ensures}\ \phi_{post};\ \{\_\} \\ \Gamma \vdash x : T_r \qquad \Gamma \vdash z' : T_p \qquad \phi \implies (y \neq \mathtt{null}) * \phi_p * \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi' \\ x \notin FV(\phi') \qquad x \neq y \wedge x \neq z' \qquad \phi_p = \phi_{pre}[y, z'/\mathtt{this}, z] \qquad \phi_q = \phi_{post}[y, z', x/\mathtt{this}, z, \mathtt{result}]\end{array}}{\Gamma \vdash \{\phi\} x := y.m(z') \{\phi' * \phi_q\}} \text{ HApp}$$

$$\frac{\phi \implies \phi'}{\Gamma \vdash \{\phi\} \mathtt{assert}\ \phi' \{\phi\}} \text{ HAssert}$$

$$\frac{\phi \implies \phi_r * \phi' \qquad \emptyset \vdash_{\mathtt{sfrm}} \phi'}{\Gamma \vdash \{\phi\} \mathtt{release}\ \phi_r \{\phi'\}} \text{ HRelease}$$

$$\frac{x \notin \mathtt{dom}(\Gamma) \qquad \Gamma, x : T \vdash \{(x = \mathtt{defaultValue}(T)) * \phi\} \overline{s} \{\phi'\}}{\Gamma \vdash \{\phi\} T\ x; \overline{s} \{\phi'\}} \text{ HDeclare}$$

$$\frac{\Gamma \vdash \{\phi_p\} s_1 \{\phi_q\} \qquad \Gamma \vdash \{\phi_q\} s_2 \{\phi_r\}}{\Gamma \vdash \{\phi_p\} s_1; s_2 \{\phi_r\}} \text{ HSec}$$

3.5.1 Notation

$$\frac{\hat{\phi} \implies \hat{\phi}' \qquad x \notin FV(\hat{\phi}') \qquad \Gamma \vdash x : C \qquad \mathtt{fields}(C) = \overline{f}}{\Gamma \vdash \{\hat{\phi}\} x := \mathtt{new}\ C \{\hat{\phi}' \mathbin{\hat{*}} (x \neq \mathtt{null}) \mathbin{\hat{*}} \overline{\mathtt{acc}(x, f_i)}\}} \text{ HNewObj}$$

$$\frac{\hat{\phi} \implies \mathtt{acc}(x.f) * \hat{\phi}' \qquad \Gamma \vdash x : C \qquad \Gamma \vdash y : T \qquad \vdash C.f : T}{\Gamma \vdash \{\hat{\phi}\} x.f := y \{\hat{\phi}' \mathbin{\hat{*}} \mathtt{acc}(x.f) \mathbin{\hat{*}} (x \neq \mathtt{null}) \mathbin{\hat{*}} (x.f = y)\}} \text{ HFieldAssign}$$

$$\frac{\hat{\phi} \implies \hat{\phi}' \qquad x \notin FV(\hat{\phi}') \qquad x \notin FV(e) \qquad \Gamma \vdash x : T \qquad \Gamma \vdash e : T \qquad [\![e]\!] \subseteq \hat{\phi}'}{\Gamma \vdash \{\hat{\phi}\} x := e \{\hat{\phi}' \mathbin{\hat{*}} (x = e)\}} \text{ HVarAssign}$$

$$\frac{\hat{\phi} \implies \hat{\phi}' \qquad \texttt{result} \notin FV(\hat{\phi}') \qquad \Gamma \vdash x : T \qquad \Gamma \vdash \texttt{result} : T}{\Gamma \vdash \{\hat{\phi}\}\texttt{return } x\{\hat{\phi}' \mathbin{\hat{*}} (\texttt{result} = x)\}}\ \textsc{HReturn}$$

$$\frac{\begin{array}{c}\Gamma \vdash y : C \qquad \texttt{mmethod}(C, m) = T_r\ m(T_p\ z)\ \texttt{requires}\ \widehat{\phi_{pre}};\ \texttt{ensures}\ \widehat{\phi_{post}};\ \{\_\} \\ \Gamma \vdash x : T_r \qquad \Gamma \vdash z' : T_p \qquad \hat{\phi} \implies (y \neq \texttt{null}) * \hat{\phi}_p * \hat{\phi}' \\ x \notin FV(\hat{\phi}') \qquad x \neq y \wedge x \neq z' \qquad \hat{\phi}_p = \widehat{\phi_{pre}}[y, z'/\texttt{this}, z] \qquad \hat{\phi}_q = \widehat{\phi_{post}}[y, z', x/\texttt{this}, z, \texttt{result}]\end{array}}{\Gamma \vdash \{\hat{\phi}\}x := y.m(z')\{\hat{\phi}' * \hat{\phi}_q\}}\ \textsc{HApp}$$

$$\frac{\hat{\phi} \implies \phi'}{\Gamma \vdash \{\hat{\phi}\}\texttt{assert } \phi'\{\hat{\phi}\}}\ \textsc{HAssert}$$

$$\frac{\hat{\phi} \implies \phi_r * \hat{\phi}'}{\Gamma \vdash \{\hat{\phi}\}\texttt{release } \phi_r\{\hat{\phi}'\}}\ \textsc{HRelease}$$

$$\frac{x \notin \texttt{dom}(\Gamma) \qquad \Gamma, x : T \vdash \{\hat{\phi} \mathbin{\hat{*}} (x = \texttt{defaultValue}(T))\}\overline{s}\{\hat{\phi}'\}}{\Gamma \vdash \{\hat{\phi}\}T\ x; \overline{s}\{\hat{\phi}'\}}\ \textsc{HDeclare}$$

$$\frac{\Gamma \vdash \{\hat{\phi}_p\}s_1\{\hat{\phi}_q\} \qquad \Gamma \vdash \{\hat{\phi}_q\}s_2\{\hat{\phi}_r\}}{\Gamma \vdash \{\hat{\phi}_p\}s_1; s_2\{\hat{\phi}_r\}}\ \textsc{HSec}$$

3.5.2 Deterministic

$$\frac{\hat{\phi}[\mathbf{w/o}\ x] = \hat{\phi}' \qquad \Gamma \vdash x : C \qquad \texttt{fields}(C) = \overline{f}}{\Gamma \vdash \{\hat{\phi}\}x := \texttt{new } C\{\hat{\phi}' \mathbin{\hat{*}} (x \neq \texttt{null}) \mathbin{\hat{*}} \overline{\texttt{acc}(x, f_i)}\}}\ \textsc{HNewObj}$$

$$\frac{\hat{\phi}[\mathbf{w/o}\ \texttt{acc}(x.f)] = \hat{\phi}' \qquad \hat{\phi} \implies \texttt{acc}(x.f) \qquad \Gamma \vdash x : C \qquad \Gamma \vdash y : T \qquad \vdash C.f : T}{\Gamma \vdash \{\hat{\phi}\}x.f := y\{\hat{\phi}' \mathbin{\hat{*}} \texttt{acc}(x.f) \mathbin{\hat{*}} (x \neq \texttt{null}) \mathbin{\hat{*}} (x.f = y)\}}\ \textsc{HFieldAssign}$$

$$\frac{\hat{\phi}[\mathbf{w/o}\ x] = \hat{\phi}' \qquad x \notin FV(e) \qquad \Gamma \vdash x : T \qquad \Gamma \vdash e : T \qquad \hat{\phi}' \implies [\![e]\!]}{\Gamma \vdash \{\hat{\phi}\}x := e\{\hat{\phi}' \mathbin{\hat{*}} (x = e)\}}\ \textsc{HVarAssign}$$

Have $\mathbin{\hat{*}}$ take care of necessary congruent rewriting of e in order to preserve self-framing!

$$\frac{\hat{\phi}[\mathbf{w/o}\ \texttt{result}] = \hat{\phi}' \qquad \Gamma \vdash x : T \qquad \Gamma \vdash \texttt{result} : T}{\Gamma \vdash \{\hat{\phi}\}\texttt{return } x\{\hat{\phi}' \mathbin{\hat{*}} (\texttt{result} = x)\}}\ \textsc{HReturn}$$

$$\frac{\begin{array}{c}\hat{\phi}[\mathbf{w/o}\ x][\mathbf{w/o}\ \lfloor\hat{\phi}_p\rfloor] = \hat{\phi}' \\ \Gamma \vdash y : C \qquad \texttt{mmethod}(C, m) = T_r\ m(T_p\ z)\ \texttt{requires}\ \widehat{\phi_{pre}};\ \texttt{ensures}\ \widehat{\phi_{post}};\ \{\_\} \qquad \Gamma \vdash x : T_r \qquad \Gamma \vdash z' : T_p \\ \hat{\phi} \implies \hat{\phi}_p \mathbin{\hat{*}} (y \neq \texttt{null}) \qquad x \neq y \wedge x \neq z' \qquad \hat{\phi}_p = \widehat{\phi_{pre}}[y, z'/\texttt{this}, z] \qquad \hat{\phi}_q = \widehat{\phi_{post}}[y, z', x/\texttt{this}, z, \texttt{result}]\end{array}}{\Gamma \vdash \{\hat{\phi}\}x := y.m(z')\{\hat{\phi}' \mathbin{\hat{*}} \hat{\phi}_q\}}\ \textsc{HApp}$$

$$\frac{\hat{\phi} \implies \phi_a}{\Gamma \vdash \{\hat{\phi}\}\texttt{assert}\ \phi_a\{\hat{\phi}\}}\ \textsc{HAssertBAD (grad lifting non-trivial!)}$$

$$\frac{\hat{\phi}[\textbf{w/o}\ \lfloor \phi_a \rfloor] = \hat{\phi}' \qquad \hat{\phi} \implies \phi_a}{\Gamma \vdash \{\hat{\phi}\}\texttt{assert}\ \phi_a\{\hat{\phi}' \mathbin{\hat{*}} \phi_a\}}\ \textsc{HAssert}$$

$$\frac{\hat{\phi}[\textbf{w/o}\ \lfloor \phi_r \rfloor] = \hat{\phi}' \qquad \hat{\phi} \implies \phi_r}{\Gamma \vdash \{\hat{\phi}\}\texttt{release}\ \phi_r\{\hat{\phi}'\}}\ \textsc{HRelease}$$

$$\frac{x \notin \texttt{dom}(\Gamma) \qquad \Gamma, x : T \vdash \{\hat{\phi} \mathbin{\hat{*}} (x = \texttt{defaultValue}(T))\}\overline{s}\{\hat{\phi}'\}}{\Gamma \vdash \{\hat{\phi}\}T\ x; \overline{s}\{\hat{\phi}'\}}\ \textsc{HDeclare}$$

$$\frac{\Gamma \vdash \{\hat{\phi_p}\}s_1\{\hat{\phi_q}\} \qquad \Gamma \vdash \{\hat{\phi_q}\}s_2\{\hat{\phi_r}\}}{\Gamma \vdash \{\hat{\phi_p}\}s_1; s_2\{\hat{\phi_r}\}}\ \textsc{HSec}$$

3.5.3 Gradual

$$\frac{\widetilde{\phi}[\textbf{w/o}\ x] = \widetilde{\phi}' \qquad \Gamma \vdash x : C \qquad \texttt{fields}(C) = \overline{f}}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}x := \texttt{new}\ C\{\widetilde{\phi}' \mathbin{\widetilde{*}} (x \neq \texttt{null}) \mathbin{\widetilde{*}} \overline{\texttt{acc}(x, f_i)}\}}\ \textsc{GHNewObj}$$

$$\frac{\widetilde{\phi}[\textbf{w/o}\ \texttt{acc}(x.f)] = \widetilde{\phi}' \qquad \widetilde{\phi} \mathrel{\widetilde{\implies}} \texttt{acc}(x.f) \qquad \Gamma \vdash x : C \qquad \Gamma \vdash y : T \qquad \vdash C.f : T}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}x.f := y\{\widetilde{\phi}' \mathbin{\widetilde{*}} \texttt{acc}(x.f) \mathbin{\widetilde{*}} (x \neq \texttt{null}) \mathbin{\widetilde{*}} (x.f = y)\}}\ \textsc{GHFieldAssign}$$

$$\frac{\widetilde{\phi}[\textbf{w/o}\ x] = \widetilde{\phi}' \qquad x \notin FV(e) \qquad \Gamma \vdash x : T \qquad \Gamma \vdash e : T \qquad \widetilde{\phi}' \mathrel{\widetilde{\implies}} [\![e]\!]}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}x := e\{\widetilde{\phi}' \mathbin{\widetilde{*}} (x = e)\}}\ \textsc{GHVarAssign}$$

Let $\widetilde{*}$ behave like $\hat{*}$ if first operand is static - otherwise its regular concatenation.

$$\frac{\widetilde{\phi}[\textbf{w/o}\ \texttt{result}] = \widetilde{\phi}' \qquad \Gamma \vdash x : T \qquad \Gamma \vdash \texttt{result} : T}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}\texttt{return}\ x\{\widetilde{\phi}' \mathbin{\widetilde{*}} (\texttt{result} = x)\}}\ \textsc{GHReturn}$$

$$\frac{\begin{array}{c}\widetilde{\phi}[\textbf{w/o}\ x][\textbf{w/o}\ \lfloor \widetilde{\phi_p} \rfloor_{\Gamma, y, z'}] = \widetilde{\phi}' \\ \Gamma \vdash y : C \qquad \texttt{mmethod}(C, m) = T_r\ m(T_p\ z)\ \texttt{requires}\ \widetilde{\phi_{pre}};\ \texttt{ensures}\ \widetilde{\phi_{post}};\ \{\_\} \qquad \Gamma \vdash x : T_r \qquad \Gamma \vdash z' : T_p \\ \widetilde{\phi} \mathrel{\widetilde{\implies}} \widetilde{\phi_p} \mathbin{\widetilde{*}} (y \neq \texttt{null}) \qquad x \neq y \land x \neq z' \qquad \widetilde{\phi_p} = \widetilde{\phi_{pre}}[y, z'/\texttt{this}, z] \qquad \widetilde{\phi_q} = \widetilde{\phi_{post}}[y, z', x/\texttt{this}, z, \texttt{result}]\end{array}}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}x := y.m(z')\{\widetilde{\phi}' \mathbin{\widetilde{*}} \widetilde{\phi_q}\}}\ \textsc{GHApp}$$

$$\frac{\widetilde{\phi}[\textbf{w/o}\ \lfloor \phi_a \rfloor] = \widetilde{\phi}' \qquad \hat{\phi} \mathrel{\widetilde{\implies}} \phi_a}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}\texttt{assert}\ \phi_a\{\widetilde{\phi}' \mathbin{\widetilde{*}} \phi_a\}}\ \textsc{GHAssert}$$

$$\frac{\widetilde{\phi}[\textbf{w/o}\ \lfloor \phi_r \rfloor] = \widetilde{\phi}' \qquad \widetilde{\phi} \mathrel{\widetilde{\implies}} \phi_r}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}\texttt{release}\ \phi_r\{\widetilde{\phi}'\}}\ \textsc{GHRelease}$$

$$\frac{x \notin \texttt{dom}(\Gamma) \qquad \Gamma, x : T \vdash \{\widetilde{\phi} \mathbin{\widetilde{*}} (x = \texttt{defaultValue}(T))\}\overline{s}\{\widetilde{\phi'}\}}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}T\ x; \overline{s}\{\widetilde{\phi'}\}}\ \text{GHDeclare}$$

$$\frac{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_p}\}s_1\{\widetilde{\phi_q}\} \qquad \Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_q}\}s_2\{\widetilde{\phi_r}\}}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_p}\}s_1; s_2\{\widetilde{\phi_r}\}}\ \text{GHSec}$$

4 Dynamic semantics

4.1 Expressions ($H, \rho \vdash e \Downarrow v$)

$$\frac{}{H, \rho \vdash x \Downarrow \rho(x)}\ \text{EEVar}$$

$$\frac{}{H, \rho \vdash v \Downarrow v}\ \text{EEValue}$$

$$\frac{H, \rho \vdash e \Downarrow o}{H, \rho \vdash e.f \Downarrow H(o)(f)}\ \text{EEAcc}$$

4.2 Formulas ($H, \rho, A \vDash \phi$)

$$\frac{}{H, \rho, A \vDash \texttt{true}}\ \text{EATrue}$$

$$\frac{H, \rho \vdash e_1 \Downarrow v_1 \qquad H, \rho \vdash e_2 \Downarrow v_2 \qquad v_1 = v_2}{H, \rho, A \vDash (e_1 = e_2)}\ \text{EAEqual}$$

$$\frac{H, \rho \vdash e_1 \Downarrow v_1 \qquad H, \rho \vdash e_2 \Downarrow v_2 \qquad v_1 \neq v_2}{H, \rho, A \vDash (e_1 \neq e_2)}\ \text{EANEqual}$$

$$\frac{H, \rho \vdash e \Downarrow o \qquad (o, f) \in A}{H, \rho, A \vDash \texttt{acc}(e.f)}\ \text{EAAcc}$$

$$\frac{A_1 = A \backslash A_2 \qquad H, \rho, A_1 \vDash \phi_1 \qquad H, \rho, A_2 \vDash \phi_2}{H, \rho, A \vDash \phi_1 * \phi_2}\ \text{EASepOp}$$

We give a denotational semantics of formulas as $[\![\phi]\!] = \{\ (H, \rho, A) \mid H, \rho, A \vDash \phi\ \}$
Note: ϕ satisfiable $\iff$ $[\![\phi]\!] \neq \emptyset$

4.2.1 Implication ($\phi_1 \implies \phi_2$)

$$\phi_1 \implies \phi_2 \qquad \iff \qquad \forall H, \rho, A : H, \rho, A \vDash \phi_1 \implies H, \rho, A \vDash \phi_2$$

Drawn from def. of entailment in “A Formal Semantics for Isorecursive and Equirecursive State Abstractions”.

4.2.2 Implying inequality

$$
\begin{aligned}
& \phi * (e_1 = e_1) * (e_2 = e_2) \implies (e_1 \neq e_2) \\
= \; & \forall H, \rho, A : \; H, \rho, A \vDash \phi * (e_1 = e_1) * (e_2 = e_2) \implies H, \rho, A \vDash (e_1 \neq e_2) \\
= \; & \forall H, \rho, A : \; (\exists v_1, v_2 : \; H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi) \implies (\exists v_1, v_2 : \; H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land (\\
= \; & \forall H, \rho, A, v_1, v_2 : \; (H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi) \implies (\exists v_1, v_2 : \; H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land (v_1 \\
= \; & \forall H, \rho, A, v_1, v_2 : \; (H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi) \implies (v_1 \neq v2) \\
= \; & \forall H, \rho, A, v_1, v_2 : \; \neg(H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi) \lor (v_1 \neq v2) \\
= \; & \forall H, \rho, A, v_1, v_2 : \; \neg(H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi \land (v_1 = v2)) \\
= \; & \forall H, \rho, A : \; \neg(\exists v_1, v_2 : \; H, \rho \vdash e_1 \Downarrow v_1 \land H, \rho \vdash e_2 \Downarrow v_2 \land H, \rho, A \vDash \phi \land (v_1 = v2)) \\
= \; & \forall H, \rho, A : \; \neg(H, \rho, A \vDash \phi \land H, \rho, A \vDash (e_1 = e_2)) \\
= \; & \forall H, \rho, A : \; \neg H, \rho, A \vDash \phi * (e_1 = e_2) \\
= \; & \neg \text{sat } (\phi * (e_1 = e_2))
\end{aligned}
$$

4.3 Footprint ($\lfloor \phi \rfloor_{H,\rho} = A_d$)

$$
\begin{aligned}
\lfloor \texttt{true} \rfloor_{H,\rho} &= \emptyset \\
\lfloor e_1 = e_2 \rfloor_{H,\rho} &= \emptyset \\
\lfloor e_1 \neq e_2 \rfloor_{H,\rho} &= \emptyset \\
\lfloor \texttt{acc}(x.f) \rfloor_{H,\rho} &= \{(o, f)\} \text{ where } H, \rho \vdash x \Downarrow o \\
\lfloor \phi_1 * \phi_2 \rfloor_{H,\rho} &= \lfloor \phi_1 \rfloor_{H,\rho} \cup \lfloor \phi_2 \rfloor_{H,\rho}
\end{aligned}
$$

4.4 Small-step ($(H, S) \to (H, S)$)

$$
\frac{H, \rho \vdash x \Downarrow o \qquad H, \rho \vdash y \Downarrow v_y \qquad (o, f) \in A \qquad H' = H[o \mapsto [f \mapsto v_y]]}{(H, (\rho, A, x.f := y; \overline{s}) \cdot S) \to (H', (\rho, A, \overline{s}) \cdot S)} \text{ ESFieldAssign}
$$

$$
\frac{H, \rho \vdash e \Downarrow v \qquad \rho' = \rho[x \mapsto v]}{(H, (\rho, A, x := e; \overline{s}) \cdot S) \to (H, (\rho', A, \overline{s}) \cdot S)} \text{ ESVarAssign}
$$

$$
\frac{\texttt{fields}(C) = \overline{T}\ \overline{f} \qquad \rho' = \rho[x \mapsto o] \qquad \begin{array}{c} o \notin \texttt{dom}(H) \\ A' = A * \overline{(o, f_i)} \end{array} \qquad H' = H[o \mapsto [\overline{f \mapsto \texttt{defaultValue}(T)}]]}{(H, (\rho, A, x := \texttt{ new } C; \overline{s}) \cdot S) \to (H', (\rho', A', \overline{s}) \cdot S)} \text{ ESNewObj}
$$

$$
\frac{H, \rho \vdash x \Downarrow v_x \qquad \rho' = \rho[\texttt{result} \mapsto v_x]}{(H, (\rho, A, \texttt{return } x; \overline{s}) \cdot S) \to (H, (\rho', A, \overline{s}) \cdot S)} \text{ ESReturn}
$$

$$
\frac{\begin{array}{c} H, \rho \vdash y \Downarrow o \\ H, \rho \vdash z \Downarrow v \qquad H(o) = (C, \_) \qquad \texttt{mmethod}(C, m) = T_r\ m(T\ w) \texttt{ requires } \phi; \texttt{ ensures } \_; \; \{\overline{r}\} \\ \rho' = [\texttt{result} \mapsto \texttt{defaultValue}(T_r), \texttt{this} \mapsto o, w \mapsto v] \qquad H, \rho', A \vDash \phi \qquad A' = \lfloor \phi \rfloor_{H,\rho'} \end{array}}{(H, (\rho, A, x := y.m(z); \overline{s}) \cdot S) \to (H, (\rho', A', \overline{r}) \cdot (\rho, A \setminus A', x := y.m(z); \overline{s}) \cdot S)} \text{ ESApp}
$$

$$
\frac{\begin{array}{c} H, \rho \vdash y \Downarrow o \\ H(o) = (C, \_) \qquad \texttt{mpost}(C, m) = \phi \qquad H, \rho', A' \vDash \phi \qquad A'' = \lfloor \phi \rfloor_{H,\rho'} \qquad H, \rho' \vdash \texttt{result} \Downarrow v_r \end{array}}{(H, (\rho', A', \emptyset) \cdot (\rho, A, x := y.m(z); \overline{s}) \cdot S) \to (H, (\rho[x \mapsto v_r], A * A'', \overline{s}) \cdot S)} \text{ ESAppFinish}
$$

$$\frac{H, \rho, A \vDash \phi}{(H, (\rho, A, \texttt{assert}\ \phi; \overline{s}) \cdot S) \rightarrow (H, (\rho, A, \overline{s}) \cdot S)}\ \textsc{ESAssert}$$

$$\frac{H, \rho, A \vDash \phi \qquad A' = A \setminus \lfloor \phi \rfloor_{H,\rho}}{(H, (\rho, A, \texttt{release}\ \phi; \overline{s}) \cdot S) \rightarrow (H, (\rho, A', \overline{s}) \cdot S)}\ \textsc{ESRelease}$$

$$\frac{\rho' = \rho[x \mapsto \texttt{defaultValue}(T)]}{(H, (\rho, A, T\ x; \overline{s}) \cdot S) \rightarrow (H, (\rho', A, \overline{s}) \cdot S)}\ \textsc{ESDeclare}$$

5 Gradualization

5.1 Syntax

5.1.1 Gradual formula

$$\widetilde{\phi} \quad ::= \quad \phi \mid ?\ *\ \phi$$

Note: consider ? in other positions as "self-framing delimiter", but with semantically identical meaning.
As long as ? is only legal in the front though: $\phi_1 * \widetilde{\phi_2}$ propagates the ? to the very left in case $\widetilde{\phi_2}$ contains one.

5.1.2 Self-framed and satisfiable formula

$$\hat{\phi} \quad \in \quad \{\ \phi \mid\ \vdash_{\texttt{sfrm}} \phi \wedge \text{sat}\ \phi\ \}$$

5.2 Concretization

$$\begin{aligned}
&\gamma(\hat{\phi}) && = \{\ \hat{\phi}\ \} \\
&\gamma(?\ *\ \phi') && = \{\ \hat{\phi} \mid \hat{\phi} \implies \phi'\ \} \ \text{ if } \phi' \text{ satisfiable} \\
&\gamma(\phi) \text{ undefined otherwise}
\end{aligned}$$

$$\widetilde{\phi_1} \sqsubseteq \widetilde{\phi_2} \quad :\iff \quad \gamma(\widetilde{\phi_1}) \subseteq \gamma(\widetilde{\phi_2})$$

5.3 Abstraction

$$\alpha(\overline{\phi}) \qquad = \min_{\sqsubseteq} \{\ \widetilde{\phi} \mid \overline{\phi} \subseteq \gamma(\widetilde{\phi})\ \}$$

Equivalent to:

$$\begin{aligned}
&\alpha(\{\phi\}) && = \phi \\
&\alpha(\overline{\phi}) && = \dot{\alpha}(\overline{\phi}) := \sup_{\sqsubseteq} \{\ ?\ *\ \phi \mid \phi \in \overline{\phi}\ \}
\end{aligned}$$

Proved:

- partial function
- sound
- optimal
- $\alpha(\gamma(\widetilde{\phi})) = \widetilde{\phi}$
- does this make $\langle \gamma, \alpha \rangle$ a (partial) "galois insertion"?

5.4 Lifting functions

Gradual lifting $\widetilde{f} : \widetilde{\phi} \to \widetilde{\phi}$ of a function $f : \phi \to \phi$:

$$\widetilde{f}(\widetilde{\phi}) := \alpha(\overline{f}(\gamma(\widetilde{\phi})))$$

This formal definition has drawbacks:

- Calculations on infinite set (not implementable)
- Determine supremum of infinite set (not even clear if it exists)

Turns out above definition can be rewritten in an equivalent, computable way.

5.4.1 Dominator Theory

TODO: first tackle singleton case etc.
Theorem:
For every ϕ, there exists a finite set of "dominators" $\mathtt{dom}(\phi)$, such that

$$\gamma(?\ *\ \phi) = \bigcup_{\hat{\phi} \in \mathtt{dom}(f(\phi))} \gamma(?\ *\ \hat{\phi})$$

Consequence:

$$\begin{aligned} ?\ *\ \phi &= \alpha(\gamma(?\ *\ \phi)) \\ &= \dot{\alpha}(\gamma(?\ *\ \phi)) \\ &= \dot{\alpha}(\bigcup_{\hat{\phi} \in \mathtt{dom}(\phi)} \gamma(?\ *\ \hat{\phi})) \\ &= \dot{\alpha}(\bigcup_{\hat{\phi} \in \mathtt{dom}(\phi)} \{\hat{\phi}\}) \\ &= \dot{\alpha}(\mathtt{dom}(\phi)) \\ &= \sup_{\sqsubseteq} \{\ ?\ *\ \phi' \mid \phi' \in \mathtt{dom}(\phi)\ \} \end{aligned}$$

Analogous, for monotonic f:

$$\begin{aligned} &\alpha(\overline{f}(\gamma(?\ *\ \phi))) \\ &= \dot{\alpha}(\overline{f}(\gamma(?\ *\ \phi))) \\ &= \dot{\alpha}(\overline{f}(\bigcup_{\hat{\phi} \in \mathtt{dom}(\phi)} \gamma(?\ *\ \hat{\phi}))) \\ &= \dot{\alpha}(\overline{f}(\bigcup_{\hat{\phi} \in \mathtt{dom}(\phi)} \{\hat{\phi}\})) \\ &= \dot{\alpha}(\overline{f}(\mathtt{dom}(\phi))) \\ &= \sup_{\sqsubseteq} \{\ ?\ *\ f(\phi') \mid \phi' \in \mathtt{dom}(\phi)\ \} \end{aligned}$$

Re-definition of gradual lifting:

$$\widetilde{f}(\phi) := f(\phi)$$
$$\widetilde{f}(?\ *\ \phi) := \alpha(\overline{f}(\gamma(?\ *\ \phi))) = \dot{\alpha}(\overline{f}(\mathtt{dom}(\phi)))$$

In terms of implementation: At least no more infinite sets, need to calculating supremum remains.

Interesting observation:

$$\widetilde{f}(?\ *\ \hat{\phi}) = \dot{\alpha}(\overline{f}(\mathtt{dom}(\hat{\phi}))) = \dot{\alpha}(\overline{f}(\{\hat{\phi}\})) = \dot{\alpha}(\{f(\hat{\phi})\}) = ?\ *\ f(\hat{\phi})$$

This observation raises the question whether it is possible to generalize the equality to work with arbitrary formulas, getting rid of $\dot{\alpha}$ and calculating a supremum entirely.

5.4.2 Generalization: Auto-liftable functions

Goal: Get a definition of $\widetilde{f}$ that is even easier to handle and implement. Therefore we want to investigate whether, or under which circumstances

$$\widetilde{f}(\widetilde{\phi}) = f(\widetilde{\phi}) \qquad \text{(i.e. } f \text{ applied to the static part of } \widetilde{\phi}\text{)}$$

holds.
We call functions f satisfying above equality "auto-liftable".
Counterexamples:

- $f(\phi) = \texttt{acc}(x.f) * \phi$

  $$\widetilde{f}(? * x.f = 3) = ? * \texttt{false} \neq ? * \texttt{acc}(x.f) * (x.f = 3) = f(? * (x.f = 3))$$

  Cause: $\gamma(? * x.f = 3)$ only contains self-framed formulas, so access to $x.f$ is always included. Adding it another time results in duplicate access and therefore unsatisfiable formulas.

- $f(\phi) =$ remove all terms containing x

  $$\widetilde{f}(? * a = 3) = ? \neq ? * (a = 3) = f(? * (a = 3))$$

  Cause: $(a = x) * (x = 3) \in \gamma(? * a = 3)$ and $f((a = x) * (x = 3)) = \texttt{true}$. Abstracting from a (non-singleton) set that contains $\texttt{true}$ yields $?$.

What is necessary to generalize this as $\alpha(\overline{f}(\gamma(? * \phi))) = ? * f(\phi)$?

$$\begin{gathered}
\forall \phi' \in \gamma(? * f(\phi)), \exists \phi'' \in \gamma(? * \phi), \phi' \in \gamma(? * f(\phi'')) \\
\Longrightarrow \\
\forall \phi' \in \gamma(? * f(\phi)), \exists \phi'' \in \gamma(? * \phi), ? * \phi' \sqsubseteq ? * f(\phi'') \\
\Longrightarrow \\
\forall \phi' \in \texttt{dom}(f(\phi)), \exists \phi'' \in \texttt{dom}(\phi), ? * \phi' \sqsubseteq ? * f(\phi'') \\
\Longrightarrow \\
\forall \phi' \in \texttt{dom}(f(\phi)), ? * \phi' \sqsubseteq \sup_{\sqsubseteq} \{ \ ? * f(\phi') \mid \phi' \in \texttt{dom}(\phi) \ \} \\
\Longleftrightarrow \\
\sup_{\sqsubseteq} \{ \ ? * \phi' \mid \phi' \in \texttt{dom}(f(\phi)) \ \} \sqsubseteq \sup_{\sqsubseteq} \{ \ ? * f(\phi') \mid \phi' \in \texttt{dom}(\phi) \ \} \\
\Longleftrightarrow \\
? * f(\phi) \sqsubseteq \alpha(\overline{f}(\gamma(? * \phi)))
\end{gathered}$$

$$\begin{gathered}
? * f(\phi) \sqsubseteq ? * f(\phi) \\
\Longrightarrow \\
\forall \phi' \in \texttt{dom}(\phi), ? * f(\phi') \sqsubseteq ? * f(\phi) \\
\Longleftrightarrow \\
\sup_{\sqsubseteq} \{ \ ? * f(\phi') \mid \phi' \in \texttt{dom}(\phi) \ \} \sqsubseteq ? * f(\phi) \\
\Longleftrightarrow \\
\alpha(\overline{f}(\gamma(? * \phi))) \sqsubseteq ? * f(\phi)
\end{gathered}$$

For a function f to be auto-liftable, the following properties are sufficient:

- Monotonicity
- $\forall \phi' \in \gamma(? * f(\phi)), \exists \phi'' \in \gamma(? * \phi), \phi' \in \gamma(? * f(\phi''))$

5.4.3 Liftable composition

Given liftable functions f and g, is $g \circ f$ liftable? Monotonicity is obviously preserved.
Other condition:

$$
\begin{gathered}
? * g(f(\phi)) \sqsubseteq \alpha(\overline{g}(\gamma(? * f(\phi)))) \wedge ? * f(\phi) \sqsubseteq \alpha(\overline{f}(\gamma(? * \phi))) \\
\Longrightarrow \\
? * g(f(\phi)) \sqsubseteq \alpha(\overline{g}(\gamma(? * f(\phi)))) \wedge \alpha(\gamma(? * f(\phi))) \sqsubseteq \alpha(\overline{f}(\gamma(? * \phi))) \\
\Longrightarrow \\
? * g(f(\phi)) \sqsubseteq \alpha(\overline{g}(\gamma(? * f(\phi)))) \wedge \alpha(\overline{g}(\gamma(? * f(\phi)))) \sqsubseteq \alpha(\overline{g}(\overline{f}(\gamma(? * \phi)))) \\
\Longrightarrow \\
? * g(f(\phi)) \sqsubseteq \alpha(\overline{g}(\overline{f}(\gamma(? * \phi)))) \\
\Longrightarrow \\
? * (g \circ f)(\phi) \sqsubseteq \alpha(\overline{(g \circ f)}(\gamma(? * \phi)))
\end{gathered}
$$

5.5 Gradual Lifting

5.5.1 Self framing

$$\frac{A \vdash_{\texttt{sfrm}} \phi}{A \mathrel{\widetilde{\vdash_{\texttt{sfrm}}}} \phi} \textsc{GSfrmNonGrad}$$

$$\frac{}{A \mathrel{\widetilde{\vdash_{\texttt{sfrm}}}} ? * \phi} \textsc{GSfrmGrad}$$

5.5.2 Implication

$$\frac{\phi_1 \implies \phi_2}{\phi_1 \mathrel{\widetilde{\implies}} \widetilde{\phi_2}} \textsc{GImplNonGrad}$$

$$\frac{\hat{\phi_m} \implies \phi_2 \qquad \hat{\phi_m} \implies \phi_1}{? * \phi_1 \mathrel{\widetilde{\implies}} \widetilde{\phi_2}} \textsc{GImplGrad}$$

$\hat{\phi_m}$ is evidence!

Consistent transitivity
While $\implies$ is transitive, $\widetilde{\implies}$ is generally not.
But maybe not even necessary with smarter hoare rules?

5.5.3 Equality

$$\frac{\phi_1 = \phi_2}{\phi_1 \approx \phi_2} \textsc{GEqStatic}$$

$$\frac{\text{at least one of } \widetilde{\phi_1} \text{ or } \widetilde{\phi_2} \text{ contains } ? \qquad \widetilde{\phi_1} \mathrel{\widetilde{\implies}} \widetilde{\phi_2} \qquad \widetilde{\phi_2} \mathrel{\widetilde{\implies}} \widetilde{\phi_1}}{\widetilde{\phi_1} \approx \widetilde{\phi_2}} \textsc{GEqGradual}$$

5.5.4 Append

by definition:
$\widetilde{\phi} \mathbin{\widetilde{*}} \phi_p = \alpha(\gamma(\widetilde{\phi}) \overline{*} \phi_p)$

equivalent to:

$$\begin{array}{ll} \widetilde{\phi} \mathbin{\widetilde{*}} \phi_p = \widetilde{\phi} * \phi_p & \text{if } \forall \hat{\phi}_1, (\hat{\phi}_1 \implies \phi * \phi_p) \implies \exists \hat{\phi}_2, (\hat{\phi}_2 \implies \phi \wedge \hat{\phi}_1 \implies \hat{\phi}_2 * \phi_p) \\ & \text{if } \forall \hat{\phi}_1 \in \gamma(\widetilde{\phi} * \phi_p), \exists \hat{\phi}_2 \in \gamma(\widetilde{\phi}), \hat{\phi}_1 \implies \hat{\phi}_2 * \phi_p \\ \widetilde{\phi} \mathbin{\widetilde{*}} \phi_p \ \textit{undefined} & \text{otherwise} \end{array}$$

5.6 Gradual Hoare: minimal static rule approach

Example:

$$\frac{\emptyset \vdash_{\texttt{sfrm}} \widetilde{\phi'} \qquad x \notin FV(\widetilde{\phi'}) \qquad x \notin FV(e) \qquad \epsilon \vdash \widetilde{\phi} \vdash x : T \qquad \epsilon \vdash \widetilde{\phi} \vdash e : T \qquad \epsilon \vdash \lfloor \widetilde{\phi'} \rfloor \vdash_{\texttt{sfrm}} e \qquad \overset{\epsilon \vdash \widetilde{\phi} \,\widetilde{\implies}\, \widetilde{\phi'}}{}}{\vdash \{\widetilde{\phi}\} x := e \{\widetilde{\phi'} * (x = e)\}} \textsc{GHVarAssign}$$

Collapsing (hidden) gradual implications into a single one:

$$\frac{\epsilon \vdash \widetilde{\phi} \,\widetilde{\implies}\, (x : T) * [\![e : T]\!]_C * \widetilde{\phi'} \qquad \emptyset \vdash_{\texttt{sfrm}} [\![e : T]\!]_C * \widetilde{\phi'} \qquad x \notin FV(\widetilde{\phi'}) \qquad x \notin FV(e) \qquad [e : T]_C}{\vdash \{\widetilde{\phi}\} x := e \{[\![e : T]\!]_C * \widetilde{\phi'} * (x = e)\}} \textsc{GHVarAssig}$$

When shifting implication responsibility to GHSec:

$$\frac{x \notin FV(\widetilde{\phi'}) \qquad x \notin FV(e) \qquad [e : T]_C}{\vdash \{(x : T) * [\![e : T]\!]_C * \widetilde{\phi'}\} x := e \{[\![e : T]\!]_C * \widetilde{\phi'} * (x = e)\}} \textsc{GHVarAssign}$$

Example derivation:

$\{(x : T) * (y : C) * \texttt{acc}(y.a) * \texttt{acc}(y.a.b) * \texttt{acc}(y.a.b.c) * \widetilde{\phi'}\}$

$\{(x : T) * [\![y.a.b.c : T]\!]_C * \widetilde{\phi'}\}$

$x := y.a.b.c;$ $\qquad x \notin FV(\widetilde{\phi'})$
$\qquad x \notin FV(y.a.b.c)$
$\qquad [y.a.b.c : T]_C = \ \vdash C_y = C \ \wedge \ \vdash C_y.a : C_a \ \wedge \ \vdash C_a.b : C_b \ \wedge \ \vdash C_b.c : T$

$\{[\![y.a.b.c : T]\!]_C * \widetilde{\phi'} * (x = y.a.b.c)\}$

$\{(y : C) * \texttt{acc}(y.a) * \texttt{acc}(y.a.b) * \texttt{acc}(y.a.b.c) * \widetilde{\phi'} * (x = y.a.b.c)\}$

5.6.1 GHFieldAssign

$$\frac{\widetilde{\phi_1} \approx (x : C) * (y : T) * (x \neq \texttt{null}) * \phi * \texttt{acc}(x.f) \qquad \overset{\vdash_{\texttt{sfrm}} \phi \qquad \vdash C.f : T}{\widetilde{\phi_2} \approx (x : C) * \texttt{acc}(x.f) * (x \neq \texttt{null}) * (x.f = y) * \phi}}{\widetilde{\vdash} \{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}} \textsc{GHFieldA}$$

5.6.2 GHSec - sound but obviously not complete!

$$\frac{\widetilde{\vdash} \{\widetilde{\phi_p}\} s_1 \{\widetilde{\phi_{q1}}\} \qquad \phi_{q1} \implies \phi_{q2} \qquad \emptyset \vdash_{\texttt{sfrm}} \phi_{q2} \qquad \widetilde{\vdash} \{\phi_{q2}\} s_2 \{\widetilde{\phi_r}\}}{\widetilde{\vdash} \{\widetilde{\phi_p}\} s_1; s_2 \{\widetilde{\phi_r}\}} \textsc{GHSec}$$

5.7 Gradual Hoare: minimal HSec approach (implications per rule)

$$\frac{\vdash_{\texttt{sfrm}} \phi \qquad \vdash C.f : T \qquad \phi_1 \implies (x : C) * (y : T) * \phi * \texttt{acc}(x.f) \qquad \phi_2 = (x : C) * \texttt{acc}(x.f) * (x.f = y) * \phi}{\vdash \{\phi_1\} x.f := y \{\phi_2\}} \text{ HFieldAssign}$$

$$\frac{\vdash_{\texttt{sfrm}} \phi \qquad \vdash C.f : T \qquad \widetilde{\phi_1} \widetilde{\implies} (x : C) * (y : T) * \phi * \texttt{acc}(x.f) \qquad \widetilde{\phi_2} \approx (x : C) * \texttt{acc}(x.f) * (x.f = y) * \phi}{\widetilde{\vdash} \{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}} \text{ GHFieldAssign}$$

Note: With this alternative rule design $\widetilde{\implies}$ is consistently used with static formulas as second argument. This plays nicely with the fact that $\widetilde{\implies}$ does not care about the gradualness of that argument. Might make sense to define lifting of $\implies$ as lifting on only the first parameter in the first place.

Minimum runtime checks: For $\widetilde{\phi_1} \widetilde{\implies} \widetilde{\phi_2}$ to hold at runtime, practically just ϕ_2 needs to hold. So that would be a valid assertion to check. Yet, we know statically that ϕ_1 holds, so we can remove everything from the runtime check that is implied by ϕ_1. So in a sense, we only need to check $\phi_2 \backslash \phi_1$ at runtime (the operator can be an approximation).

5.8 Gradual Hoare: deterministic approach

5.8.1 HFieldAssign

$$\frac{\vdash C.f : T \qquad \phi_1 \implies (x : C) * (y : T) * \texttt{acc}(x.f) \qquad \phi_2 = (x : C) * \texttt{acc}(x.f) * (x.f = y) * \phi_1[\textbf{w/o } \texttt{acc}(x.f)]}{\vdash \{\phi_1\} x.f := y \{\phi_2\}} \text{ HFieldAssign}$$

Note: $\phi[\textbf{w/o } \texttt{acc}(x.f)]$ removes $\texttt{acc}(x.f)$ and all uses of $x.f$ from ϕ. The result is self-framed given that ϕ is.

Attention: This version is weaker than the other (pairwise equivalent) versions of HFieldAssign!
Explanation: Above operator may remove more information than necessary from ϕ.
Example:

- Given: $\phi_1 = \texttt{acc}(x.f) * (x.f = a) * (x.f = b)$
- Goal: $\phi_2 \implies (a = b)$
- **not provable** with this deterministic version of HFieldAssign
- **provable** with all other versions

Probably it's possible to apply the operator without information loss after expanding formula using equalities (transitive hull).

5.8.2 GHFieldAssign

(= gradual lifting of GHFieldAssign as function)

$$\frac{\widetilde{\phi_2} = \alpha(\{\phi_2 \mid \phi_1 \in \gamma(\widetilde{\phi_1}) \ \wedge \ \vdash \{\phi_1\} x.f := y \{\phi_2\} \ \})}{\widetilde{\vdash} \{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}} \text{ GHFieldAssign}$$

Which should be equivalent to this:

$$\frac{\vdash C.f : T \qquad \phi_1 \implies (x : C) * (y : T) * \texttt{acc}(x.f) \qquad \phi_2 = (x : C) * (y : T) * \texttt{acc}(x.f) * (x.f = y) * \phi_1[\textbf{w/o } \texttt{acc}(x.f)]}{\widetilde{\vdash} \{\phi_1\} x.f := y \{\phi_2\}} \text{ GHFA1}$$

$$\dfrac{\begin{array}{c}\vdash C.f : T \\ ? \, * \, \phi_1 \widetilde{\Longrightarrow}_{\phi_m} (x : C) * \texttt{acc}(x.f) \\ \phi_2 = (x : C) * \texttt{acc}(x.f) * (x.f = y) * \phi_m[\textbf{w/o}\ \texttt{acc}(x.f)]\end{array}}{\widetilde{\vdash}\{? \, * \, \phi_1\} x.f := y \{? \, * \, \phi_2\}} \text{ GHFA2}$$

Which should be summarizable as this:

$$\dfrac{\begin{array}{c}\vdash C.f : T \\ \widetilde{\phi_1} \widetilde{\Longrightarrow}_{\widetilde{\phi_m}} (x : C) * (y : T) * \texttt{acc}(x.f) \\ \widetilde{\phi_2} = (x : C) * \texttt{acc}(x.f) * (x.f = y) * \widetilde{\phi_m}[\textbf{w/o}\ \texttt{acc}(x.f)]\end{array}}{\widetilde{\vdash}\{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}} \text{ GHFA}$$

Which for well-formed programs is equivalent to:

$$\dfrac{\begin{array}{c}\vdash C.f : T \\ \phi_1 \implies (x : C) * (y : T) \qquad \widetilde{\phi_1} \widetilde{\Longrightarrow} \texttt{acc}(x.f) \\ \widetilde{\phi_2} = (x : C) * (y : T) * \texttt{acc}(x.f) * (x.f = y) * \widetilde{\phi_1}[\textbf{w/o}\ \texttt{acc}(x.f)]\end{array}}{\widetilde{\vdash}\{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}} \text{ GHFA}$$

Observations:

- $\widetilde{\phi_m}$ is the interior (first argument) of the implication, effectively the meet of first and second argument.
- for the gradual rules to work, the **w/o**-operator **must** be implemented with minimal information loss

5.9 Theorems

5.9.1 Soundness of α

$$\forall \overline{\phi} : \overline{\phi} \subseteq \gamma(\alpha(\overline{\phi}))$$

5.9.2 Optimality of α

$$\forall \overline{\phi}, \widetilde{\phi} : \overline{\phi} \subseteq \gamma(\widetilde{\phi}) \implies \gamma(\alpha(\overline{\phi})) \subseteq \gamma(\widetilde{\phi})$$

6 Theorems

6.1 Invariant $invariant(H, \rho, A_d, \phi)$

6.1.1 Phi valid

$$\vdash_{\texttt{sfrm}} \phi$$

6.1.2 Phi holds

$$H, \rho, A_d \vDash \phi$$

6.1.3 Types preserved

$$\begin{array}{c}\forall e, T : \phi \vdash e : T \\ \implies H, \rho \vdash e : T\end{array}$$

6.1.4 Heap consistent

$$
\begin{aligned}
\forall o, C, \mu, f, T : & \ H(o) = (C, \mu) \\
& \implies \texttt{fieldType}(C, f) = T \\
& \implies H, \rho \vdash \mu(f) : T
\end{aligned}
$$

6.1.5 Heap not total

$$
\begin{aligned}
& \exists o_{min} : \\
\forall o \geq o_{min} : & \ o \notin \texttt{dom}(H) \\
& \wedge \forall f, (o, f) \notin A
\end{aligned}
$$

6.2 Soundness

6.2.1 Progress

$$
\begin{aligned}
\forall \ldots : & \quad \vdash \{\phi_1\} s' \{\phi_2\} \\
& \implies \mathit{invariant}(H_1, \rho_1, A_1, \phi_1) \\
& \implies \exists H_2, \rho_2, A_2 : (H_1, (\rho_1, A_1, s'; \overline{s}) \cdot S) \to^* (H_2, (\rho_2, A_2, \overline{s}) \cdot S)
\end{aligned}
$$

6.2.2 Preservation

$$
\begin{aligned}
\forall \ldots : & \quad \vdash \{\phi_1\} s' \{\phi_2\} \\
& \implies \mathit{invariant}(H_1, \rho_1, A_1, \phi_1) \\
& \implies (H_1, (\rho_1, A_1, s'; \overline{s}) \cdot S) \to^* (H_2, (\rho_2, A_2, \overline{s}) \cdot S) \\
& \implies \mathit{invariant}(H_2, \rho_2, A_2, \phi_2)
\end{aligned}
$$