1 Syntax

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

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 (*program* **OK**)

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

2.0.2 Well-formed class (*cls* **OK**)

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

2.0.3 Well-formed method (*method* **OK in** C)

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

3 Static semantics

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

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

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

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

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

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

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

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

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

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

3.2.1 Implication ($\phi_1 \overset{\cdot}{\implies} \phi_2$)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.5.1 Notation

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

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

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

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

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

$$\dfrac{\hat{\phi} \implies \phi'}{\Gamma \vdash \{\hat{\phi}\}\mathtt{assert}\ \phi'\{\hat{\phi}\}}\ \textsc{HAssert}$$

$$\dfrac{\hat{\phi} \implies \phi_r * \hat{\phi}'}{\Gamma \vdash \{\hat{\phi}\}\mathtt{release}\ \phi_r\{\hat{\phi}'\}}\ \textsc{HRelease}$$

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

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

3.5.2 Deterministic

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

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

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

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

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

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

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

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

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

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

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

3.5.3 Gradual

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

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

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

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

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

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

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

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

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

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

4 Dynamic semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Gradualization

5.1 Syntax

5.1.1 Gradual formula

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

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:

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

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” $\mathrm{dom}(\phi)$, such that

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

Consequence:

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

Analogous, for monotonic f:

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

Re-definition of gradual lifting:

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

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

Interesting observation:

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

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:

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

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

  Cause: $\gamma(\texttt{?} * \texttt{(x.f = 3)})$ 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`

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

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

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

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

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

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

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

5.4.3 Liftable composition

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

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

5.5 Lifting implication

Implication is the only predicate on pairs of ϕ occurring in our Hoare rules.

Looking at HAssert (the version with forwarding pre to post-condition) shows that the gradual lifting is not achieved by merely replacing the implication with gradual implication. The gradual lifting potentially has a stronger postcondition than precondition due to filtering out of all concrete formulas that don't satisfy the implication (i.e. where the partial function HAssert is undefined). TODO in thesis: Actually elaborate this in more detail (with example, etc.) AAAANNND show the alternative definition (that is actually correct but not ideal) that uses removal and readdition of the asserted formula!

GHAssert could be defined using appropriate additional measures (e.g. manually fixing up the postcondition) but the problem can be tackled more elegantly by redesigning implication itself.

In general, whenever implication is used, the set of potential formulas is reduced. Without mirroring this reduction in the static system, evidence is required to make sure the reduction does not contradict assertions down the road.

We will create a partial function based on classical implication and give its gradual lifting. The gradual lifting will by definition mirror reductions caused by the implication, restoring the simplicity of HAssert and especially GHAssert and removing(?) the need of evidence.

5.5.1 Implication as partial function imp : $\phi \to \hat{\phi} \to \hat{\phi}$

$$
\begin{aligned}
&\text{imp}(\phi_a)(\hat{\phi}) = \hat{\phi} && \text{if } \hat{\phi} \implies \phi_a \\
&\text{imp}(\phi_a)(\hat{\phi}) \ \textit{undefined} && \text{otherwise}
\end{aligned}
$$

5.5.2 Gradual lifting $\widetilde{\textbf{imp}}$: $\phi \to \widetilde{\phi} \to \widetilde{\phi}$

$$
\begin{aligned}
\widetilde{\text{imp}}(\phi_a)(\widetilde{\phi}) &= \alpha(\overline{\text{imp}(\phi_a)}(\gamma(\widetilde{\phi}))) \\
&= \alpha(\{ \ \text{imp}(\phi_a)(\hat{\phi}) \mid \hat{\phi} \in \gamma(\widetilde{\phi}) \ \}) \\
&= \alpha(\{ \ \hat{\phi} \mid \hat{\phi} \in \gamma(\widetilde{\phi}) \wedge \hat{\phi} \implies \phi_a \ \}) \\
&= \alpha(\{ \ \hat{\phi} \mid \hat{\phi} \in \gamma(\widetilde{\phi}) \wedge \hat{\phi} \in \gamma(? * \phi_a) \ \}) \\
&= \alpha(\gamma(? * \phi_a) \cap \gamma(\widetilde{\phi}))
\end{aligned}
$$

5.6 Gradual Lifting

5.6.1 Self framing

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

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

5.6.2 Implication

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

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

Minimum runtime checks: For $\widetilde{\phi_1} \mathrel{\widetilde{\implies}} \widetilde{\phi_2}$ 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).
$\hat{\phi_m}$ is evidence!

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

5.6.3 Equality

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

$$\dfrac{\begin{array}{c}\text{at least one of } \widetilde{\phi_1} \text{ or } \widetilde{\phi_2} \text{ contains } ? \\ \widetilde{\phi_1} \mathrel{\widetilde{\implies}} \widetilde{\phi_2} \qquad \widetilde{\phi_2} \mathrel{\widetilde{\implies}} \widetilde{\phi_1}\end{array}}{\widetilde{\phi_1} \approx \widetilde{\phi_2}}\ \textsc{GEqGradual}$$

5.6.4 Append

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

equivalent to:

$$\begin{array}{ll}
\widetilde{\phi} \mathbin{\widetilde{*}} \phi_p = \widetilde{\phi} * \phi_p & \text{if } \forall \hat{\phi_1}, (\hat{\phi_1} \implies \phi * \phi_p) \implies \exists \hat{\phi_2}, (\hat{\phi_2} \implies \phi \land \hat{\phi_1} \implies \hat{\phi_2} * \phi_p) \\
 & \text{if } \forall \hat{\phi_1} \in \gamma(\widetilde{\phi} * \phi_p), \exists \hat{\phi_2} \in \gamma(\widetilde{\phi}), \hat{\phi_1} \implies \hat{\phi_2} * \phi_p \\
\widetilde{\phi} \mathbin{\widetilde{*}} \phi_p \ \textit{undefined} & \text{otherwise}
\end{array}$$

5.7 Theorems

5.7.1 Soundness of α

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

5.7.2 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

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

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

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

6.2.2 Preservation

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