1 Syntax

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

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)

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

3 Static semantics

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

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

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

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

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

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

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

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

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

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

3.2.1 Implication ($\phi_1 \dot{\Longrightarrow} \phi_2$)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.5.1 Notation

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

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

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

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

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

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

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

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

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

3.5.2 Deterministic

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

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

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

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

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

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

$$\frac{\hat{\phi} \implies \phi_a}{\Gamma \vdash \{\hat{\phi}\}\texttt{assert}\ \phi_a\{\hat{\phi}\}}\ \textsc{HAssertIrregular (grad lifting not piecewise)}$$

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

$$\frac{\text{imp}(\phi_a)(\hat{\phi}) = \hat{\phi}'}{\Gamma \vdash \{\hat{\phi}\}\texttt{assert}\ \phi_a\{\hat{\phi}'\}}\ \textsc{HAssert}$$

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

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

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

3.5.3 Gradual

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

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

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

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

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

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

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

$$\frac{\widetilde{\mathrm{imp}}(\phi_a)(\widetilde{\phi}) = \widetilde{\phi'}}{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi}\}\texttt{assert}\ \phi_a\{\widetilde{\phi'}\}}\ \textsc{GHAssert}$$

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

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

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

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

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

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

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

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

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

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

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

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 ($(H, S) \to (H, S)$)

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

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

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

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

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

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

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

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

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

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

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

5.2 Concretization

$\gamma(\hat{\phi}) = \{\ \hat{\phi}\ \}$
$\gamma(\texttt{?} * \phi') = \{\ \hat{\phi} \mid \hat{\phi} \implies \phi'\ \}$ if ϕ' satisfiable
$\gamma(\phi)$ undefined otherwise

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

5.3 Abstraction

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

Equivalent to:

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

Proved:

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

5.4 Temp

†:

$$(\phi_1 \implies \phi_2) \implies f(\phi_1) \text{ defined} \implies f(\phi_2) \text{ defined}$$

- Concat: the case (total in usage!)
- remove variable/access: total
- substitution: total
- implication: NOT the case

The gradual Hoare rules (except GHSec) themselves are total apart from implication part!

Functions must preserve satisfiability and self-framing!

5.5 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.

5.5.1 Dominator Theory

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

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

Consequence:

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

Analogous, for monotonic f:

$$\begin{aligned}
&\alpha(\overline{f}(\gamma(\texttt{?} * \phi))) \\
&\overset{f \text{ not narrowing to single element (also includes definedness!)}}{=} \dot{\alpha}(\overline{f}(\gamma(\texttt{?} * \phi))) \\
&= \dot{\alpha}(\overline{f}(\bigcup_{\hat{\phi} \in \mathrm{dom}(\phi)} \gamma(\texttt{?} * \hat{\phi}))) \\
&\overset{\dagger}{=} \dot{\alpha}(\overline{f}(\bigcup_{\hat{\phi} \in \mathrm{dom}(\phi)} \{\hat{\phi}\})) \\
&= \dot{\alpha}(\overline{f}(\mathrm{dom}(\phi))) \\
&= \sup_{\sqsubseteq} \{\ \texttt{?} * f(\phi') \mid \phi' \in \mathrm{dom}(\phi)\ \}
\end{aligned}$$

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.

5.5.2 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

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

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 `?`.

Required: $\alpha(\overline{f}(\gamma(\texttt{?} * \phi))) = \texttt{?} * f(\phi)$ Note:

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

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

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

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

5.5.3 Auto-liftable composition

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

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

5.5.4 No liftable composition for HSec - emerging runtime semantics

Motivation: While compositional lifting (involving $\cdot \mathbin{\widetilde{*}} \phi$, $[\mathbf{w/o}\ x]$ etc.) works out great in most static Hoare rules, it does not work in HSec. This is due to a subtlety (†)... TODO
Example: `release acc(x.f); release acc(x.f);`

- not typable with any static precondition
- typable with gradual precondition:

$$\frac{\begin{array}{c}\widetilde{\mathrm{imp}}(\texttt{acc(x.f)})(\texttt{?} * \texttt{(x} \neq \texttt{null)}) = \texttt{?} * \texttt{(x.f = x.f)} \\ (\texttt{?} * \texttt{(x.f = x.f)})[\mathbf{w/o}\ \texttt{acc(x.f)}] = \texttt{?} * \texttt{(x} \neq \texttt{null)}\end{array}}{\Gamma \mathbin{\widetilde{\vdash}} \{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}\texttt{release acc(x.f);}\{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}}\ \textsc{GHRelease}$$

$$\frac{\begin{array}{c}\Gamma \mathbin{\widetilde{\vdash}} \{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}\texttt{release acc(x.f);}\{\texttt{?} * \texttt{(x} \neq \texttt{null)}\} \\ \Gamma \mathbin{\widetilde{\vdash}} \{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}\texttt{release acc(x.f);}\{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}\end{array}}{\Gamma \mathbin{\widetilde{\vdash}} \{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}\texttt{release acc(x.f); release acc(x.f);}\{\texttt{?} * \texttt{(x} \neq \texttt{null)}\}}\ \textsc{GHSec}$$

Trying to "fix" GHSec statically turns out to be impossible without runtime information. That moves our focus towards dynamic semantics.
Discussion, comparison with AGT:

AGT

reduction = composite terms (parts had explicit type annotations!) get simplified to single term.

For type derivations this means that stuff must still be compatible, i.e. the single term must still be typeable. The constructor for composite terms ($\widetilde{\text{T}}$App predicate) is thus not the gradual lifting of TApp - it allows constructing terms which cannot succeed at runtime.

GVER

reduction = pushing an environment through, instruction by instruction (analogous to `ins1(ins2(ins3(env)))` in λ/AGT-setting).

In other words: Instructions are *not* combined with others to form "simpler" instructions. The behavior becomes more apparent if we imagined loops as being part of our language. Although HSec is like a counterpart of TApp in constructing composite terms, the dynamic semantics are quite orthogonal. HSec semantically would more closely correspond to a dedicated B-combinator - which would then also not provide runtime semantics for AGT.

The fundamental difference is that AGT is based typing terms, giving rules to construct these terms, whereas GVER is based on typing environments, giving rules to construct functions *manipulating* these environments.

Let's pretend we had instead given rules of constructing environments (i.e. environments as values, instructions as functions on these values) and see how to derive *direct* runtime semantics from that. In this setting, there would have to be a rule analogous to Tapp:

$$\frac{\Gamma \vdash s : \hat{\phi_{pre}} \to \hat{\phi_{post}} \qquad \vdash env : \hat{\phi} \qquad \hat{\phi} <: \hat{\phi_{pre}}}{\vdash s(env) : \hat{\phi_{post}}}\ \textsc{TappVER}$$

Leveraging our syntax:

$$\frac{\Gamma \vdash \{\hat{\phi_{pre}}\}s\{\hat{\phi_{post}}\} \qquad env \vdash \hat{\phi} \qquad \hat{\phi} \implies \hat{\phi_{pre}}}{s(env) \vdash \hat{\phi_{post}}}\ \textsc{TappVER}$$

Which by definition of implication is exactly the same as:

$$\frac{\Gamma \vdash \{\hat{\phi_{pre}}\}s\{\hat{\phi_{post}}\} \qquad env \vdash \hat{\phi_{pre}}}{s(env) \vdash \hat{\phi_{post}}}\ \textsc{TappVER}$$

Note that this is precisely the definition of soundness, ensuring progress ($s(env)$ being defined) and preservation (ϕ_{post} really holding for the new environment). Gradual attempt:

$$\frac{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_{pre}}\} s \{\widetilde{\phi_{post}}\} \qquad env \mathrel{\widetilde{\vdash}} \widetilde{\phi_{pre}}}{s(env) \vdash \widetilde{\phi_{post}}} \ \widetilde{\text{TappVER}}$$

Remark: $env \mathrel{\widetilde{\vdash}} \widetilde{\phi_{pre}}$ has a very simple implementation.

Just as it is the case with $\widetilde{\text{T}}$app, this rule is imprecise, requiring additional measures to make guarantees about runtime behavior: The gradual Hoare premise might require a concretization of $\widetilde{\phi_{pre}}$ that does not evaluate in env.

Unfortunately, the gradual Hoare predicate is not transparent about its requirements for $\widetilde{\phi_{pre}}$. Let's taking a step back and look at the regular Hoare rules: They are partial functions, being undefined only if s itself is malformed (you could split that part of the static semantics apart from everything involving ϕ!) or if the precondition violates some implication.

Now, the gradual Hoare rule guarantees the well-formedness of s itself since the gradualization only affects formulas. The critical part is the implication, which is getting weakened by the gradualization, subsequently resulting in $\widetilde{\text{T}}$appVER being imprecise as illustrated above.

Let's reiterate TappVER, trying to track the precondition requirements explicitly:

$$\frac{\Gamma \vdash \{\hat{\phi_{pre}}\} s \{\hat{\phi_{post}}\} \qquad \hat{\phi_{pre}} \implies \text{goal}_\Gamma(s) \qquad env \vdash \hat{\phi_{pre}}}{s(env) \vdash \hat{\phi_{post}}} \ \text{TappVER}$$

...where goal(s) is supposed to be implied by ϕ_{pre} according to the Hoare rule for statement s (for Hoare rules that don't have such requirement, we define goal(s) = `true`).

This can be rewritten as:

$$\frac{\Gamma \vdash \{\hat{\phi_{pre}}\} s \{\hat{\phi_{post}}\} \qquad env \vdash \text{goal}_\Gamma(s) \qquad env \vdash \hat{\phi_{pre}}}{s(env) \vdash \hat{\phi_{post}}} \ \text{TappVER}$$

New gradual attempt:

$$\frac{\Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_{pre}}\} s \{\widetilde{\phi_{post}}\} \qquad env \mathrel{\widetilde{\vdash}} \texttt{?} * \text{goal}_\Gamma(s) \qquad env \mathrel{\widetilde{\vdash}} \widetilde{\phi_{pre}}}{s(env) \vdash \widetilde{\phi_{post}}} \ \widetilde{\text{TappVER}}$$

This time, the lifting is actually precise. Proof:

$$
\begin{aligned}
& \widetilde{\mathrm{T}}\mathrm{appVER}_{\Gamma,s,env}(\widetilde{\phi_{pre}}, \widetilde{\phi_{post}}) \\
& \implies \\
& \Gamma \mathrel{\widetilde{\vdash}} \{\widetilde{\phi_{pre}}\} s \{\widetilde{\phi_{post}}\} \ \wedge\ env \mathrel{\widetilde{\vdash}} \mathtt{?} * \mathrm{goal}_\Gamma(s) \ \wedge\ env \mathrel{\widetilde{\vdash}} \widetilde{\phi_{pre}} \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \widetilde{H}_{\Gamma,s}(\widetilde{\phi_{pre}}) = \widetilde{\phi_{post}} \ \wedge\ env \mathrel{\widetilde{\vdash}} \mathtt{?} * \mathrm{goal}_\Gamma(s) \ \wedge\ env \mathrel{\widetilde{\vdash}} \widetilde{\phi_{pre}} \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \widetilde{H}_{\Gamma,s}(\widetilde{\phi_{pre}}) = \widetilde{\phi_{post}} \ \wedge\ env \mathrel{\widetilde{\vdash}} \widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \alpha(\overline{K}_{\Gamma,s}(\gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}})))) = \widetilde{\phi_{post}} \ \wedge\ env \mathrel{\widetilde{\vdash}} \widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \gamma(\alpha(\overline{K}_{\Gamma,s}(\gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}))))) = \gamma(\widetilde{\phi_{post}}) \ \wedge\ env \mathrel{\widetilde{\vdash}} \widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \gamma(\alpha(\overline{K}_{\Gamma,s}(\gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}))))) = \gamma(\widetilde{\phi_{post}}) \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}})) : \ & env \vdash \hat{\phi_{pre}} \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
& \gamma(\alpha(\overline{K}_{\Gamma,s}(\gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}))))) = \gamma(\widetilde{\phi_{post}}) \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}})) : \ & env \vdash \hat{\phi_{pre}} \ \wedge\ K_{\Gamma,s}(\hat{\phi_{pre}}) \in \overline{K}_{\Gamma,s}(\gamma(\mathrm{imp}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}}))) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\mathrm{imp}}(\mathrm{goal}_\Gamma(s))(\widetilde{\phi_{pre}})) : \ & env \vdash \hat{\phi_{pre}} \ \wedge\ K_{\Gamma,s}(\hat{\phi_{pre}}) \in \gamma(\widetilde{\phi_{post}}) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\phi_{pre}}) : \ & \hat{\phi_{pre}} \implies \mathrm{goal}_\Gamma(s) \ \wedge\ env \vdash \hat{\phi_{pre}} \ \wedge\ K_{\Gamma,s}(\hat{\phi_{pre}}) \in \gamma(\widetilde{\phi_{post}}) \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\phi_{pre}}) : \ & H_{\Gamma,s}(\hat{\phi_{pre}}) \in \gamma(\widetilde{\phi_{post}}) \ \wedge\ env \ \vdash\ \hat{\phi_{pre}} \\
& \implies \\
& \Gamma \vdash s \ \wedge \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\phi_{pre}}), \hat{\phi_{post}} \in \gamma(\widetilde{\phi_{post}}) : \ & H_{\Gamma,s}(\hat{\phi_{pre}}) = \hat{\phi_{post}} \ \wedge\ env \ \vdash\ \hat{\phi_{pre}} \\
& \implies \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\phi_{pre}}), \hat{\phi_{post}} \in \gamma(\widetilde{\phi_{post}}) : \ & \Gamma \vdash \{\hat{\phi_{pre}}\} s \{\hat{\phi_{post}}\} \ \wedge\ env \ \vdash\ \hat{\phi_{pre}} \\
& \implies \\
\exists \hat{\phi_{pre}} \in \gamma(\widetilde{\phi_{pre}}), \hat{\phi_{post}} \in \gamma(\widetilde{\phi_{post}}) : \ & \mathrm{TappVER}_{\Gamma,s,env}(\hat{\phi_{pre}}, \hat{\phi_{post}})
\end{aligned}
$$

Note that it implicitly ensures the consistent transitivity of the gradual version of $\hat{\phi} \implies \hat{\phi_{pre}} \implies \mathrm{goal}(s)$ ($\hat{\phi}$ was simplified away earlier). However, in our setting we could simplify away the subtyping/implication-judgments that can be found in AGT: Since checking whether an environment has a certain type is

naturally defined as a conservative approximation (to whichever type describes the environment in most detail), type-judgments can always be combined with subtyping-judgments (just making the approximation a little worse).

Bottom line, instead of introducing evidence we end up with an additional premise $env \vdash \texttt{?} * \texttt{goal}(s)$ which can easily be checked at runtime.

5.6 Curry-Howard Table

content...

5.7 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 re-addition 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 (TODO: evidence is now actually the return value!).

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

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

5.7.2 Gradual lifting $\widetilde{\mathbf{imp}}$: $\phi \to \widetilde{\phi} \to \widetilde{\phi}$

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

This definition turns out to be implementable trivially.

5.8 Gradual Normal Form

TODO: make clear that this is specifically about GRADUAL formulas, i.e. ones containing ?!!!
TODO: declare some appropriate naming conventions to distinguish

We cannot compare gradual formulas (in terms of inclusion or equality) by comparing their static parts:
TODO: example of equal gradual formulas but non-equal static parts

Fortunately, there exists a normal form $\text{norm}(\texttt{?} * \phi) = \texttt{?} * \text{norm}'(\phi)$ for any gradual formula $\texttt{?} * \phi$ that satisfies the following properties:

- $\text{norm}(\texttt{?} * \phi) = \texttt{?} * \phi$ (mandatory for any normal form)
- $\text{norm}'(\phi)$ is well defined modulo equivalence, i.e. the normal form is unique modulo equivalence of the static part
- $\phi \implies \text{norm}'(\phi)$
- $\text{norm}'(\phi)$ contains no access-terms and therefore no elements of linear logic
- $\texttt{?} * \phi_1 \sqsubseteq \texttt{?} * \phi_2 \iff \text{norm}'(\phi_1) \implies \text{norm}'(\phi_2)$

5.8.1 Definition

Recall that gradual formulas $\texttt{?} * \phi_1$ and $\texttt{?} * \phi_2$ are considered equal iff $\gamma(\texttt{?} * \phi_1) = \gamma(\texttt{?} * \phi_1)$. The normal form makes use of the fact that concretizations contain only self-framed formulas.
Lemma: For any formula ϕ mentioning $x.f$:

$$\forall \hat{\phi} \in \gamma(\texttt{?} * \phi), \hat{\phi} \implies \texttt{acc}(x.f)$$

In other words: Merely mentioning a field will make sure that the concretization contains appropriate framing. This is a good starting point to justify removal of access-terms from the static part. Note, however, that just dropping all access from the static part may not result in an equivalent gradual formula for two reasons:

1. Mentioning
Dropping $\texttt{acc}(x.f)$ might result in $x.f$ not being mentioned in the formula anymore, so there would be no more reason for the access to be restored by concretization.

2. Aliasing
In general there are different ways in which access to multiple fields can be restored (this is were linear logic plays in). Example: Dropping all access from `acc(a.f) * acc(b.f) * (a.f = 3) * (b.f = x)` results in `(a.f = 3) * (b.f = x)` which might be re-framed as `acc(a.f) * (a = b) * (a.f = 3) * (a.f = x)`. In other words, the possibility of aliasing may result in a variety of re-framed formulas that are not equivalent with the original one.

Fortunately, we can prevent both problems from occurring by carefully preparing the static part before dropping all access, resulting in the following two-step approach:

1. Enhancement
Enrich the static part to counteract above problems, i.e. to enforce that access is restored exactly the right way. This is achieved by simply spelling out certain implications of the access-terms:

$\texttt{acc}(x.f) \implies (x.f = x.f)$
Access to a field implicitly guarantees that it actually evaluates to some (arbitrary) value. Note that $(x.f = x.f)$ is not a logical tautology (i.e. it is not implies by `true`), since it indeed makes sure that $x.f$ evaluates, whereas `true` does not. The bottom line is that $x.f$ is being mentioned even after dropping $\texttt{acc}(x.f)$, therefore solving the first problem.

$\texttt{acc}(x.f) * \texttt{acc}(y.f) \implies (x \neq y)$
Having access to the same field of different expressions actively prevents those expressions to ever alias. Spelling out this restriction by adding the corresponding inequality also prevents re-framing in a way that relies on aliasing. The bottom line is that any valid re-framing must restore two distinct access-terms, therefore solving the second problem.

We enhance the non-linear part of our formula by spelling out above implications in every possible way, i.e. accounting for all (pairs of) access-terms. It is worth noting that this enhancement preserves equality of the formula as only terms are added that were implied by the original formula, anyway.

2. Delinearization
All access-terms are dropped.

5.8.2 Properties

...prove properties postulated above.
equality (results directly from last prop.)
Intersection:

$$
\begin{aligned}
\texttt{?} * \phi_1 \sqcap \texttt{?} * \phi_2 \quad & := \alpha(\gamma(\texttt{?} * \phi_1) \cap \gamma(\texttt{?} * \phi_2)) \\
& = \alpha(\gamma(\text{norm}(\texttt{?} * \phi_1)) \cap \gamma(\text{norm}(\texttt{?} * \phi_2))) \\
& = \alpha(\gamma(\texttt{?} * \text{norm}'(\phi_1)) \cap \gamma(\texttt{?} * \text{norm}'(\phi_2))) \\
& = \alpha(\{\ \hat{\phi} \mid \hat{\phi} \implies \text{norm}'(\phi_1)\ \} \cap \{\ \hat{\phi} \mid \hat{\phi} \implies \text{norm}'(\phi_2)\ \}) \\
& = \alpha(\{\ \hat{\phi} \mid \hat{\phi} \implies \text{norm}'(\phi_1) \wedge \hat{\phi} \implies \text{norm}'(\phi_2)\ \}) \\
& = \alpha(\{\ \hat{\phi} \mid \hat{\phi} \implies \text{norm}'(\phi_1) * \text{norm}'(\phi_2)\ \}) \\
& = \alpha(\gamma(\texttt{?} * \text{norm}'(\phi_1) * \text{norm}'(\phi_2))) \\
& = \texttt{?} * \text{norm}'(\phi_1) * \text{norm}'(\phi_2)
\end{aligned}
$$

5.8.3 Redefinition of gradual lifting

$$
\begin{aligned}
& \widetilde{\text{imp}}(\phi_a)(\widetilde{\phi}) && = \alpha(\gamma(\widetilde{\phi}) \cap \gamma(\texttt{?} * \phi_a)) \\
& \text{is equivalent to} \\
& \widetilde{\text{imp}}(\phi_a)(\phi) && = \text{imp}(\phi_a)(\phi) \qquad \text{(by definition)} \\
& \widetilde{\text{imp}}(\phi_a)(\texttt{?} * \phi) && = \texttt{?} * \text{norm}'(\phi) * \text{norm}'(\phi_a)
\end{aligned}
$$

5.9 Gradual Lifting

5.9.1 Self framing

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

$$\frac{}{A \widetilde{\vdash}_{\texttt{sfrm}} \texttt{?} * \phi}\ \textsc{GSfrmGrad}$$

5.9.2 Implication

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

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

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).
$\hat{\phi_m}$ is evidence!

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

5.9.3 Equality

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

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

5.9.4 Append

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

equivalent to:

$\widetilde{\phi} \mathbin{\widetilde{*}} \phi_p = \widetilde{\phi} * \phi_p$	if $\forall \hat{\phi_1}, (\hat{\phi_1} \implies \phi * \phi_p) \implies \exists \hat{\phi_2}, (\hat{\phi_2} \implies \phi \wedge \hat{\phi_1} \implies \hat{\phi_2} * \phi_p)$
	if $\forall \hat{\phi_1} \in \gamma(\widetilde{\phi} * \phi_p), \exists \hat{\phi_2} \in \gamma(\widetilde{\phi}), \hat{\phi_1} \implies \hat{\phi_2} * \phi_p$
$\widetilde{\phi} \mathbin{\widetilde{*}} \phi_p$ *undefined*	otherwise

5.10 Theorems

5.10.1 Soundness of α

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

5.10.2 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

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

6.1.4 Heap consistent

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

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

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

6.2.2 Preservation

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