1 Syntax

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

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`)

$$\frac{\overline{cls_i\ \mathtt{OK}}}{(\overline{cls_i}\ \overline{s})\ \mathtt{OK}}\ \textsc{OKProgram}$$

2.0.2 Well-formed class (cls `OK`)

$$\frac{\text{unique } field\text{-names} \qquad \text{unique } method\text{-names} \qquad \overline{method_i\ \mathtt{OK}\ \mathtt{in}\ C}}{(\mathtt{class}\ C\ \{\overline{field_i}\ \overline{method_i}\})\ \mathtt{OK}}\ \textsc{OKClass}$$

2.0.3 Well-formed method ($method$ `OK in` C)

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

3 Static semantics

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

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

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

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

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

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

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

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

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

$$\frac{}{A \vdash_{\texttt{sfrm}} (x : T)}\ \textsc{WFType}$$

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

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

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 ($\Gamma \vdash e : T$)

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

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

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

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

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

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

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

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

What is the criterion of a correct gradual lifting of HCall?

4 Dynamic semantics

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

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

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

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

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

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

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

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

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

$$\frac{\rho(x) = v \qquad H \vdash v : T}{H, \rho, A \vDash (x : T)} \text{ EAT}\textsc{ype}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Proved:

- partial function
- sound
- optimal
- $\alpha(\gamma(\widetilde{\phi})) = \widetilde{\phi}$

5.4 Gradual Lifting

5.4.1 Self framing

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

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

5.4.2 Implication

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

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

$\hat{\phi_m}$ is evidence!

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

5.4.3 Equality

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

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

5.4.4 Append

by definition:

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

equivalent to:

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

5.5 Gradual Hoare: minimal static rule approach

Example:

$$\frac{\begin{array}{c}\epsilon \vdash \widetilde{\phi} \mathrel{\widetilde{\Longrightarrow}} \widetilde{\phi}' \\ \emptyset \vdash_{\texttt{sfrm}} \widetilde{\phi}' \qquad x \notin FV(\widetilde{\phi}') \qquad x \notin FV(e) \qquad \epsilon \vdash \widetilde{\phi} \vdash x : T \qquad \epsilon \vdash \widetilde{\phi} \vdash e : T \qquad \epsilon \vdash \lfloor \widetilde{\phi}' \rfloor \vdash_{\texttt{sfrm}} e\end{array}}{\vdash \{\widetilde{\phi}\} x := e \{\widetilde{\phi}' * (x = e)\}} \; \textsc{GHVarAssign}$$

Collapsing (hidden) gradual implications into a single one:

$$\frac{\epsilon \vdash \widetilde{\phi} \mathrel{\widetilde{\Longrightarrow}} (x : T) * \llbracket e : T \rrbracket_C * \widetilde{\phi}' \qquad \emptyset \vdash_{\texttt{sfrm}} \llbracket e : T \rrbracket_C * \widetilde{\phi}' \qquad x \notin FV(\widetilde{\phi}') \qquad x \notin FV(e) \qquad [e : T]_C}{\vdash \{\widetilde{\phi}\} x := e \{\llbracket e : T \rrbracket_C * \widetilde{\phi}' * (x = e)\}} \; \textsc{GHVarAssig}$$

When shifting implication responsibility to GHSec:

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

Example derivation:

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

$\{(x : T) * \llbracket y.a.b.c : T \rrbracket_C * \widetilde{\phi}'\}$

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

$\{\llbracket y.a.b.c : T \rrbracket_C * \widetilde{\phi}' * (x = y.a.b.c)\}$

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

5.5.1 GHFieldAssign

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

5.5.2 GHSec - sound but obviously not complete!

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

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

$$\frac{\begin{array}{c}\vdash_{\mathtt{sfrm}} \phi \qquad \vdash C.f : T \\ \phi_1 \implies (x : C) * (y : T) * \phi * \mathtt{acc}(x.f) \qquad \phi_2 = (x : C) * \mathtt{acc}(x.f) * (x.f = y) * \phi\end{array}}{\vdash \{\phi_1\} x.f := y \{\phi_2\}}\ \textsc{HFieldAssign}$$

$$\frac{\begin{array}{c}\vdash_{\mathtt{sfrm}} \phi \qquad \vdash C.f : T \\ \widetilde{\phi_1} \mathrel{\widetilde{\implies}} (x : C) * (y : T) * \phi * \mathtt{acc}(x.f) \qquad \widetilde{\phi_2} \approx (x : C) * \mathtt{acc}(x.f) * (x.f = y) * \phi\end{array}}{\widetilde{\vdash} \{\widetilde{\phi_1}\} x.f := y \{\widetilde{\phi_2}\}}\ \textsc{GHFieldAssign}$$

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 $\widetilde{\phi_1} \mathrel{\widetilde{\implies}} \widetilde{\phi_2}$ 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).

5.7 Gradual Hoare: deterministic approach

5.7.1 HFieldAssign

$$\frac{\begin{array}{c}\vdash C.f : T \\ \phi_1 \implies (x : C) * (y : T) * \mathtt{acc}(x.f) \qquad \phi_2 = (x : C) * \mathtt{acc}(x.f) * (x.f = y) * \phi_1[\mathbf{\mathtt{w/o}}\ \mathtt{acc}(x.f)]\end{array}}{\vdash \{\phi_1\} x.f := y \{\phi_2\}}\ \textsc{HFieldAssign}$$

Note: $\phi[\mathbf{\mathtt{w/o}}\ \mathtt{acc}(x.f)]$ removes $\mathtt{acc}(x.f)$ 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: $\phi_1 = \mathtt{acc}(x.f) * (x.f = a) * (x.f = b)$
- 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).

5.7.2 GHFieldAssign

(= gradual lifting of GHFieldAssign as function)

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

Which should be equivalent to this:

$$\frac{\begin{array}{c}\vdash C.f : T \\ \phi_1 \implies (x : C) * (y : T) * \mathtt{acc}(x.f) \\ \phi_2 = (x : C) * (y : T) * \mathtt{acc}(x.f) * (x.f = y) * \phi_1[\mathbf{\mathtt{w/o}}\ \mathtt{acc}(x.f)]\end{array}}{\widetilde{\vdash} \{\phi_1\} x.f := y \{\phi_2\}}\ \text{GHFA1}$$

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

Which should be summarizable as this:

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

Which for well-formed programs is equivalent to:

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

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

5.8 Theorems

5.8.1 Soundness of α

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

5.8.2 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

$$\vdash_{\mathtt{sfrm}} \phi$$

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

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

6.2.2 Preservation

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