1 Syntax

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

2 Assumptions

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

2.0.1 Well-formed program (*program* OK)

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

2.0.2 Well-formed class (*cls* OK)

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

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

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

3 Static semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\phi \implies (x:T)}{\phi \vdash x:T}\ \textsc{STVar}$$

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

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

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

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

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

$$\frac{\phi \implies \phi' \qquad \emptyset \vdash_{\texttt{sfrm}} \phi' \qquad x \notin FV(\phi') \qquad x \notin FVe(e) \qquad \Gamma \vdash x:T \qquad \Gamma \vdash e:T \qquad \lfloor \phi' \rfloor \vdash_{\texttt{sfrm}} e}{\Gamma \vdash \{\phi\}x := e\{\phi' * (x = e)\}}\ \textsc{HVarAssign}$$

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

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

$$\frac{\phi_1 \implies \phi_2}{\vdash \{\phi_1\}\texttt{assert}\ \phi_2\{\phi_1\}}\ \textsc{HAssert}$$

$$\frac{\phi_1 \implies \phi_2 * \phi_r \qquad \emptyset \vdash_{\texttt{sfrm}} \phi_r}{\Gamma \vdash \{\phi_1\}\texttt{release}\ \phi_2\{\phi_r\}}\ \textsc{HRelease}$$

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

3.5.1 Hoare revisited - pre-grad minification

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

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

$$\frac{\Gamma \vdash x.f : T \qquad \Gamma \vdash y : T}{\Gamma \vdash \{\phi * \texttt{acc}(x.f)\}x.f := y\{\texttt{acc}(x.f) * (x.f = y) * \phi\}} \ \textsc{HFieldAssign}$$

$$\frac{x \notin FV(\phi) \qquad x \notin FV(e) \qquad [e : T]_{T'}}{\Gamma \vdash \{(x : T) * [\![e : T]\!]_{T'} * \phi\}x := e\{[\![e : T]\!]_{T'} * \phi * (x = e)\}} \ \textsc{HVarAssign}$$

$$\frac{\texttt{result} \notin FV(\phi)}{\Gamma \vdash \{(x : T) * (\texttt{result} : T) * \phi\}\texttt{return}\ x\{(\texttt{result} : T) * (\texttt{result} = x) * \phi\}} \ \textsc{HReturn}$$

$$\frac{\texttt{mmethod}(C, m) = T_r\ m(T_p\ z)\ \texttt{requires}\ \phi_{pre};\ \texttt{ensures}\ \phi_{post};\ \{\_\} \qquad x \notin FV(\phi_r) \qquad x \neq y \land x \neq z'}{\Gamma \vdash \{(x : T_r) * (y : C) * (z' : T_p) * \phi_r * (y \neq \texttt{null}) * \phi_{pre}[y, z'/\texttt{this}, z]\}x := y.m(z')\{\phi_r * \phi_{post}[y, z', x/\texttt{this}, z, \texttt{result}]\}}$$

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

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

$$\frac{x \notin FV(\phi)}{\Gamma \vdash \{\phi\}T\ x\{(x : T) * (x = \texttt{defaultValue}(T)) * \phi\}} \ \textsc{HDeclare}$$

3.5.2 Hoare revisited - pre-grad minification with simple HSec

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

(other rules not printed here: think of previous subsection, but with self-framed implication on every precondition)

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

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

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

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

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

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

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

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".

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

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

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

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

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

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

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

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

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

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

5 Gradualization

5.1 Syntax

5.1.1 Gradual formula

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

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 Type judgment expansion

Motivation: Materialize and combine the requirements on ϕ.
Designed so that

$$\phi \vdash e : T \quad \wedge \quad \lfloor \phi \rfloor \vdash_{\mathtt{sfrm}} e$$

$$\Longleftrightarrow$$

$$[e : T]_C \quad \wedge \quad \phi \implies [\![e : T]\!]_C$$

Expand into premise: $[e : T]_C$

$$\begin{array}{lcl} [v : T]_C & = & T = C \quad \wedge \quad \vdash v : T \\ [x : T]_C & = & T = C \\ [e.f : T]_C & = & [e : C']_C \quad \wedge \quad \vdash C'.f : T \end{array}$$

Expand into formula: $[\![e : T]\!]_C$

$$\begin{array}{ll} [\![v : T]\!]_C & = \mathtt{true} \\ [\![x : T]\!]_C & = (x : C) \\ [\![e.f : T]\!]_C & = [\![e : T]\!]_C * \mathtt{acc}(e.f) \end{array}$$

5.2 Concretization

Syntax $\hat{\phi}$:= self-framed and satisfiable ϕ

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

5.3 Abstraction

$$\begin{aligned}
&\alpha(\emptyset) \text{ undefined} && \\
&\alpha(\{\ \phi\ \}) && = \phi \\
&\alpha(\overline{\phi} \text{ with maximum element } \phi) && = \ ?\ *\ \phi \\
&\alpha(\overline{\phi}) && = \ ? \quad \text{otherwise}
\end{aligned}$$

5.4 Gradual Lifting

5.4.1 Self framing

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

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

5.4.2 Implication

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

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

$\hat{\phi_m}$ is evidence!

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

5.4.3 Equality

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

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

5.5 Gradual Hoare: minimal static rule approach

Example:

$$\frac{\emptyset \vdash_{\texttt{sfrm}} \widetilde{\phi}' \qquad x \notin FV(\widetilde{\phi}') \qquad x \notin FV(e) \qquad \dfrac{\epsilon \vdash \widetilde{\phi} \mathrel{\widetilde{\Longrightarrow}} \widetilde{\phi}'}{\epsilon \vdash \widetilde{\phi} \vdash x : T} \qquad \epsilon \vdash \widetilde{\phi} \vdash e : T \qquad \epsilon \vdash \lfloor \widetilde{\phi}' \rfloor \vdash_{\texttt{sfrm}} e}{\vdash \{\widetilde{\phi}\} x := e \{\widetilde{\phi}' * (x = e)\}} \text{ GHVarAssign}$$

Collapsing (hidden) gradual implications into a single one:

$$\frac{\epsilon \vdash \widetilde{\phi} \mathrel{\widetilde{\Longrightarrow}} (x : T) * [\![e : T]\!]_C * \widetilde{\phi}' \qquad \emptyset \vdash_{\texttt{sfrm}} [\![e : T]\!]_C * \widetilde{\phi}' \qquad x \notin FV(\widetilde{\phi}') \qquad x \notin FV(e) \qquad [e : T]_C}{\vdash \{\widetilde{\phi}\} x := e \{[\![e : T]\!]_C * \widetilde{\phi}' * (x = e)\}} \text{ GHVarAssign}$$

When shifting implication responsibility to GHSec:

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

Example derivation:

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

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

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

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

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

5.5.1 GHFieldAssign

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

5.5.2 GHSec - sound but obviously not complete!

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

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

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

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

Note: With this alternative rule design $\widetilde{\Longrightarrow}$ is consistently used with static formulas as second argument. This plays nicely with the fact that $\widetilde{\Longrightarrow}$ 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 $\widetilde{\phi_1} \mathrel{\widetilde{\Longrightarrow}} \widetilde{\phi_2}$ 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

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

Note: $\phi[\mathbf{w/o}\ \mathtt{acc}(x.f)]$ 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:

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

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

Which should be summarizable as this:

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

Which for well-formed programs is equivalent to:

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

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

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

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

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

6.2 Soundness

6.2.1 Progress

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

6.2.2 Preservation

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