Implicit Dynamic Frames with Recursive Predicates

Jenna Wise, Johannes Bader, Jonathan Aldrich, Éric Tanter, Joshua Sunshine

No Institute Given

1 Examples

Fold & Unfold Specification Problem Example

Example 1.

```
1  class List {
2      int head;
3      List tail;
4
5      predicate List(List s) =
6              if s == null then true else
7                  acc(s.head) * acc(s.tail) * List(s.tail)
8
9      List insertList(List l, int val)
10         requires List(l)
11         ensures  List(result)
12     {
13
14         if (l == null) {
15             l = new List;
16             l.head = val;
17             l.tail = null;
18             fold List(l.tail);
19             fold List(l);
20         } else {
21             l = insertListHelper(l, val);
22         }
23
24         return l;
25     }
26
27     List insertListHelper(List l, int val)
28         requires List(l) * l != null
29         ensures  ? * result != null
30     {
31         unfold List(l);
32         List y = l.tail;
33         while (y != null)
34             invariant List(y)
35         {
```

```
36              unfold List(y);
37              y = y.tail;
38          }
39          y = new List;
40          y.head = val;
41          y.tail = null;
42          fold List(y.tail);
43          // fold List(l); ?
44
45          return l;
46      }
```

TODO ▶*Show how an equi-recursive dynamic semantics operates on this example*◀

2 SVL_{RP}

2.1 Abstract Syntax

$x, y, z \in VAR$ *(variables)*
$v \in VAL$ *(values)*
$e \in EXPR$ *(expressions)*
$s \in STMT$ *(statements)*
$o \in LOC$ *(object Ids)*
$f \in FIELDNAME$ *(field names)*
$m \in METHODNAME$ *(method names)*
$C \in CLASSNAME$ *(class names)*
$p \in PREDNAME$ *(predicate names)*

$$
\begin{array}{rll}
\texttt{P} & ::= & \overline{\texttt{cls}}\ \texttt{s} \\
\texttt{cls} & ::= & \texttt{class C \{ } \overline{\texttt{field}}\ \overline{\texttt{pred}}\ \overline{\texttt{method}}\ \texttt{\}} \\
\texttt{field} & ::= & \texttt{T f;} \\
\texttt{pred} & ::= & \texttt{predicate } p(\overline{T\ x}) = \widehat{\phi} \\
\texttt{T} & ::= & \texttt{int} \mid C \mid \top \\
\texttt{method} & ::= & \texttt{T m(}\overline{\texttt{T x}}\texttt{) contract \{s\}} \\
\texttt{contract} & ::= & \texttt{requires } \widehat{\phi}\ \texttt{ensures } \widehat{\phi} \\
\oplus & ::= & + \mid - \mid * \mid \setminus \\
\odot & ::= & \neq \mid = \mid < \mid > \mid \leq \mid \geq \\
s & ::= & \texttt{skip} \mid s_1\texttt{;}\ s_2 \mid \texttt{T x} \mid x := e \mid \texttt{if (e) then } \{\texttt{s}_1\} \texttt{ else } \{\texttt{s}_2\} \\
 & & \mid \texttt{while (e) inv } \widehat{\phi}\ \texttt{\{ s \}} \mid \texttt{x.f := y} \mid \texttt{x := new C} \mid y := z.m(\overline{x}) \\
 & & \mid \texttt{assert } \phi \mid \texttt{fold } p(\overline{e}) \mid \texttt{unfold } p(\overline{e}) \\
e & ::= & v \mid x \mid e \oplus e \mid e \odot e \mid e.f \\
x & ::= & \texttt{result} \mid id \mid \texttt{old}(id) \mid \texttt{this} \\
v & ::= & n \mid o \mid \texttt{null} \mid \texttt{true} \mid \texttt{false} \\
\phi & ::= & e \mid p(\overline{e}) \mid acc(e.f) \mid \phi \wedge \phi \mid \phi * \phi \mid (\texttt{if } e \texttt{ then } \phi \texttt{ else } \phi) \\
 & & \mid (\texttt{unfolding } p(\overline{e}) \texttt{ in } \phi)
\end{array}
$$

Notes:

- (`unfolding` $p(\overline{e})$ `in` ϕ) has the same value as ϕ
  - Notice, we include constructs in SVL_{RP} for abstract recursive predicates in iso-recursive style: fold and unfold statements and the unfolding assertion. These are necessary for static verification, but not for proving soundness or dynamic verification. In fact, we employ an equi-recursive style semantics for proving soundness and later for dynamic verification in GVL_{RP}, because it offers an elegant solution to dropping fold and unfold statements in the gradual setting. Section 2.3 discusses this further.
- $\widehat{\phi}$ is a self-framed formula, in an iso-recursive semantics (cannot determine self-framedness statically using an equi-recursive semantics)
- We purposely choose to use if conditionals in formulas over logical or to improve both static and dynamic checking with respect to performance
  - Logical or on average requires checking at runtime significantly more heap layouts than would be required when using an if conditional in the same fashion, because if conditionals allow us to determine and check the only possible heap layout due to aliasing. Ie. (**acc**(`y.f`) $*$ $x = y \vee$ **acc**(`x.f`) $*$ **acc**(`y.f`)) vs.
    (`if` $x = y$ `then` **acc**(`y.f`) `else` **acc**(`x.f`) $*$ **acc**(`y.f`))
  - Static checking performance with respect to satisfiability can also be similarly improved, because of the if conditional's branches being tied to the boolean expression's truth
- We also introduce a non-separating conjunction $\wedge$, in particular, to simplify WLP calculations
  - This has the benefits of improving the performance of static checking, not reducing the performance of dynamic checking as logical or does, and reducing the intellectual complexity of WLP calculations
  - Verifiers may choose to hide the non-separating conjunction from users by only using it in WLP calculations and taking steps to hide it in dynamic checking, such as by translating the formula with non-separating conjunctions to a formula without non-separating conjunctions before performing dynamic checking

JW ▶*Should we also allow pure recursive functions in assertions to increase expressiveness of the language?*◀

2.2 Static Semantics

As the focus of this work is not on typing, we assume that any program is well-typed, *e.g.*:

- arithmetic operators are only used on numeric expressions
- only boolean expressions appear in formulas (such as in if conditionals in formulas) & conditional statements, such as the if statement or while statement conditional statements
- for the sake of modularity, well-formed programs shall only allow folding/unfolding predicates within the scope of their definition.

– no default value assumed when declaring a variable, or when creating a new object the fields are not assigned a value by default

TODO ▶*add other notes about wellformedness as written by Johannes and me previously*◀ TODO ▶*information about old(id)*◀

2.3 Dynamic Semantics

Program States Program states consist of a heap and a stack, *i.e.* $\textsc{State} = \textsc{Heap} \times \textsc{Stack}$. A heap H is a partial function from heap locations to a value mapping of object fields, *i.e.* $\textsc{Heap} = \textsc{Loc} \rightharpoonup (\textsc{FieldName} \rightharpoonup \textsc{Val})$. Stack frames are extended to also track a dynamic footprint,

$\pi \in \textsc{DynFprint} = \mathcal{P}(\textsc{Loc} \times \textsc{FieldName})$:

$$S \in \textsc{Stack} ::= E \cdot S \mid \mathsf{nil} \qquad \text{where} \qquad E \in \textsc{StackFrame} = \textsc{Env} \times \textsc{DynFprint} \times \textsc{Stmt}$$

The memory accessible at any point of execution can be viewed as a tuple

$\textsc{Mem} = \textsc{Heap} \times \textsc{Env} \times \textsc{DynFprint}$.

Evaluation An expression e is evaluated according to a big-step evaluation relation $H, \rho \vdash e \Downarrow v$, yielding value v using heap H and local variable environment ρ. Variables are looked up in ρ, fields are looked up in H.

Predicate $\cdot \vDash_E \cdot \subseteq \textsc{Mem} \times \textsc{Formula}$ describes the rules for evaluating a formula.

$$\frac{}{H, \rho \vdash v \Downarrow v}\ \textsc{EVal} \qquad \frac{}{H, \rho \vdash x \Downarrow \rho(x)}\ \textsc{EVar}$$

$$\frac{H, \rho \vdash e \Downarrow o \qquad H(o) = \langle C, l \rangle}{H, \rho \vdash e.f \Downarrow l(f)}\ \textsc{EvField} \qquad \frac{H, \rho \vdash e_1 \Downarrow v_1 \qquad H, \rho \vdash e_2 \Downarrow v_2}{H, \rho \vdash e_1 \oplus e_2 \Downarrow v_1 \oplus v_2}\ \textsc{EvOp}$$

$$\frac{H, \rho \vdash e_1 \Downarrow v_1 \qquad H, \rho \vdash e_2 \Downarrow v_2}{H, \rho \vdash e_1 \odot e_2 \Downarrow v_1 \odot v_2}\ \textsc{EvComp}$$

Fig. 1. SVL_{RP}: Expression dynamic semantics

$$\frac{H, \rho \vdash e \Downarrow \texttt{true}}{\langle H, \rho, \pi \rangle \vDash_E e}\ \textsc{EvExpr} \qquad \frac{H, \rho \vdash e \Downarrow o \qquad \langle o, f \rangle \in \pi}{\langle H, \rho, \pi \rangle \vDash_E \textbf{acc}(\texttt{e.f})}\ \textsc{EvAcc}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \texttt{body}(p)(e_1, ..., e_n)}{\langle H, \rho, \pi \rangle \vDash_E p(e_1, ..., e_n)}\ \textsc{EvPred}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \phi_1 \qquad \langle H, \rho, \pi \rangle \vDash_E \phi_2}{\langle H, \rho, \pi \rangle \vDash_E \phi_1 \wedge \phi_2}\ \textsc{EvAndOp}$$

$$\frac{\langle H, \rho, \pi_1 \rangle \vDash_E \phi_1 \qquad \langle H, \rho, \pi_2 \rangle \vDash_E \phi_2}{\langle H, \rho, \pi_1 \uplus \pi_2 \rangle \vDash_E \phi_1 * \phi_2}\ \textsc{EvSepOp}$$

$$\frac{H, \rho \vdash e \Downarrow \texttt{true} \qquad \langle H, \rho, \pi \rangle \vDash_E \phi_T}{\langle H, \rho, \pi \rangle \vDash_E (\texttt{if } e \texttt{ then } \phi_T \texttt{ else } \phi_F)}\ \textsc{EvCondTrue}$$

$$\frac{H, \rho \vdash e \Downarrow \texttt{false} \qquad \langle H, \rho, \pi \rangle \vDash_E \phi_F}{\langle H, \rho, \pi \rangle \vDash_E (\texttt{if } e \texttt{ then } \phi_T \texttt{ else } \phi_F)}\ \textsc{EvCondFalse}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \phi}{\langle H, \rho, \pi \rangle \vDash_E (\texttt{unfolding } p(e_1, ..., e_n) \texttt{ in } \phi)}\ \textsc{EvUnfolding}$$

Fig. 2. SVL_{RP}: Formula dynamic semantics

Footprints and framing

TODO ▶*discuss* PERMISSIONS *minimally*◀

Footprints and framing are defined using the more easily implementable (statically) iso-recursive styled formula evaluation semantics. This differs significantly from the SVL_{RP}'s equi-recursive styled formula dynamic semantics on the following rule:

$$\frac{H, \rho \vdash e_1 \Downarrow v_1 \qquad ... \qquad H, \rho \vdash e_n \Downarrow v_n \qquad \langle p, v_1, ..., v_n \rangle \in \Pi}{\langle H, \rho, \Pi \rangle \vDash_I p(e_1, ..., e_n)}\ \textsc{EvPred}$$

Fig. 3. SVL_{RP}: Iso-recursive styled formula dynamic semantics (select rules)

Otherwise, the two evaluation semantics are defined similarly replacing π with Π.

The *footprint* of a formula ϕ, denoted $\lfloor \phi \rfloor_{H,\rho}$, is the minimum permission mask Π required to satisfy ϕ given a heap H and variable environment:

$$\lfloor \phi \rfloor_{H,\rho} = \min \{\ \Pi \in \textsc{Permissions} \mid \langle H, \rho, \Pi \rangle \vDash_I \phi\ \}$$

The footprint is defined (*i.e.* there exists a unique minimal set of permission Π) for formulas satisfiable under H and ρ. It can be implemented by simply evaluating ϕ using H and ρ, granting and recording precisely the permissions required.

A formula is said to be *framed* by permissions Π if it only mentions fields and unfolds predicates in Π (Fig. 4). Formula ϕ is called *self-framed* (we write $\vdash_{\texttt{frm}} \phi$) if $\langle H, \rho, \Pi\rangle \vDash_I \phi$ implies $\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi$ (for all H, ρ, Π). We define the set of self-framed formulas $\textsc{SFrmFormula} \stackrel{\text{def}}{=} \{\, \phi \in \textsc{Formula} \mid \vdash_{\texttt{frm}} \phi \,\}$. For example, `(x.f = 1)` is not self-framed while **acc**(x.f) `*` `(x.f = 1)` is. We write $\widehat{\phi}$ to denote self-framed formulas. We require method contracts, loop invariants as well as predicates to use self-framed formulas, as indicated in Section 2.1.

$$\frac{}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} v}\ \textsc{FrmVal} \qquad \frac{}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} x}\ \textsc{FrmVar} \qquad \frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_2}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \oplus e_2}\ \textsc{FrmOp}$$

$$\frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_2}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \odot e_2}\ \textsc{FrmComp} \qquad \frac{\langle H, \rho, \Pi\rangle \vDash_I \textbf{acc}(\texttt{e.f}) \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e.f}\ \textsc{FrmField}$$

$$\frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \textbf{acc}(\texttt{e.f})}\ \textsc{FrmAcc} \qquad \frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_1 \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_2}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_1 \wedge \phi_2}\ \textsc{FrmAndOp}$$

$$\frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_1 \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_2}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_1 * \phi_2}\ \textsc{FrmSepOp} \qquad \frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \quad \ldots \quad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_n}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} p(e_1, \ldots, e_n)}\ \textsc{FrmPred}$$

$$\frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e \qquad H, \rho \vdash e \Downarrow \texttt{true} \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_T}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} (\texttt{if } e \texttt{ then } \phi_T \texttt{ else } \phi_F)}\ \textsc{FrmCondTrue}$$

$$\frac{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} e \qquad H, \rho \vdash e \Downarrow \texttt{false} \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} \phi_F}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} (\texttt{if } e \texttt{ then } \phi_T \texttt{ else } \phi_F)}\ \textsc{FrmCondFalse}$$

$$\frac{\begin{array}{c}\langle H, \rho, \Pi\rangle \vDash_I p(e_1, \ldots, e_n) \qquad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_1 \quad \ldots \quad \vdash^{H,\rho,\Pi}_{\texttt{frmI}} e_n \\ \vdash^{H,\rho,\Pi'}_{\texttt{frmI}} \phi \qquad \Pi' = \Pi \cup \lfloor \mathsf{body}(p)(e_1, \ldots, e_n) \rfloor_{H,\rho}\end{array}}{\vdash^{H,\rho,\Pi}_{\texttt{frmI}} (\texttt{unfolding } p(e_1, \ldots, e_n) \texttt{ in } \phi)}\ \textsc{FrmUnfolding}$$

Fig. 4. SVL_{RP}: Framing

2.4 Reduction Rules

Figure 5 shows the reduction relation of SVL_{RP}.

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot nil \rangle \quad \texttt{final}}\ \textsc{SsSkip}$$

$$\frac{\langle H, \langle \rho, \pi, s_1 \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho', \pi', s_1' \rangle \cdot S \rangle}{\langle H, \langle \rho, \pi, s_1 \texttt{;}\ s_2 \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho', \pi', s_1' \texttt{;}\ s_2 \rangle \cdot S \rangle}\ \textsc{SsSeqCong}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{skip;}\ s_2 \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, s_2 \rangle \cdot S \rangle}\ \textsc{SsSeq}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{T x} \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsDeclare}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \phi}{\langle H, \langle \rho, \pi, \texttt{assert}\ \phi \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAssert}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \mathbf{acc}(\texttt{x.f}) \qquad H, \rho \vdash y \Downarrow v \qquad H' = H[o \mapsto [f \mapsto v]]}{\langle H, \langle \rho, \pi, x\texttt{.}f\ \texttt{:=}\ y \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsFAssign}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \mathsf{acc}(e) \qquad H, \rho \vdash e \Downarrow v \qquad \rho' = \rho[x \mapsto v]}{\langle H, \langle \rho, \pi, x\ \texttt{:=}\ e \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho', \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAssign}$$

$$\frac{o \notin \mathsf{dom}(H) \qquad \mathsf{fields}(C) = \overline{T_i\ f_i\texttt{;}} \qquad H' = H[o \mapsto [\overline{f_i \mapsto \mathsf{defaultValue}(T_i)}]]}{\langle H, \langle \rho, \pi, x\ \texttt{:=}\ \texttt{new}\ C \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho[x \mapsto o], \pi \cup \overline{\langle o, f_i \rangle}, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAlloc}$$

$$\frac{\begin{array}{c}\mathsf{method}(m) = T_r\ m(\overline{T\ x'})\ \texttt{requires}\ \widehat{\phi}_p\ \texttt{ensures}\ \widehat{\phi}_q\ \texttt{\{ r \}} \\ H, \rho \vdash z \Downarrow o \qquad \overline{H, \rho \vdash x \Downarrow v} \qquad \rho' = [\texttt{this} \mapsto o, \overline{x' \mapsto v}, \overline{\texttt{old}(x') \mapsto v}] \\ \pi' = \langle\langle\ \lfloor \widehat{\phi}_p \rfloor_{H,\rho'}\ \rangle\rangle_H \qquad \langle H, \rho', \pi' \rangle \vDash_E \widehat{\phi}_p\end{array}}{\langle H, \langle \rho, \pi, y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho', \pi', r \rangle \cdot \langle \rho, \pi \backslash \pi', y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle}\ \textsc{SsCall}$$

$$\frac{\mathsf{post}(m) = \widehat{\phi}_q \qquad \langle H, \rho', \pi' \rangle \vDash_E \widehat{\phi}_q \qquad \rho'' = \rho[y \mapsto \rho'(\texttt{result})]}{\langle H, \langle \rho', \pi', \texttt{skip} \rangle \cdot \langle \rho, \pi, y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho'', \pi \cup \pi', s \rangle \cdot S \rangle}\ \textsc{SsCallFinish}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \widehat{\phi}_i \qquad H, \rho \vdash e \Downarrow \texttt{false}}{\langle H, \langle \rho, \pi, \texttt{while (e) inv}\ \widehat{\phi}_{\texttt{i}}\ \texttt{\{ r \};}\ s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, s \rangle \cdot S \rangle}\ \textsc{SsWhileFalse}$$

$$\frac{\begin{array}{c}\langle H, \rho, \pi \rangle \vDash_E \widehat{\phi}_i \qquad H, \rho \vdash e \Downarrow \texttt{true} \\ \pi' = \langle\langle\ \lfloor \widehat{\phi}_i \rfloor_{H,\rho}\ \rangle\rangle_H\end{array}}{\begin{array}{c}\langle H, \langle \rho, \pi, \texttt{while (e) inv}\ \widehat{\phi}_{\texttt{i}}\ \texttt{\{ r \};}\ s \rangle \cdot S \rangle \\ \longrightarrow \\ \langle H, \langle \rho, \pi', r \rangle \cdot \langle \rho, \pi \backslash \pi', \texttt{while (e) inv}\ \widehat{\phi}_{\texttt{i}}\ \texttt{\{ r \};}\ s \rangle \cdot S \rangle\end{array}}\ \textsc{SsWhileTrue}$$

$$\frac{}{\begin{array}{c}\langle H, \langle \rho', \pi', \texttt{skip} \rangle \cdot \langle \rho, \pi, \texttt{while (e) inv}\ \widehat{\phi}_{\texttt{i}}\ \texttt{\{ r \};}\ s \rangle \cdot S \rangle \\ \longrightarrow \\ \langle H, \langle \rho', \pi \cup \pi', \texttt{while (e) inv}\ \widehat{\phi}_{\texttt{i}}\ \texttt{\{ r \};}\ s \rangle \cdot S \rangle\end{array}}\ \textsc{SsWhileFinish}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{fold}\ p(e_1, \ldots, e_n) \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsFold}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{unfold}\ p(e_1, \ldots, e_n) \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsUnfold}$$

Fig. 5. SVL_{RP}: Small-step semantics

Erasure. $\langle\langle\ \cdot\ \rangle\rangle_H$: PERMISSIONS × HEAP ⇀ DYNFPRINT

$$\langle\langle\ \Pi\ \rangle\rangle_H = \{\langle o, f\rangle \mid \langle o, f\rangle \in \Pi\} \cup \langle\langle\ \Pi'\ \rangle\rangle_H$$
$$\texttt{where}\ \Pi' = \cup_{\langle p, v_1, ..., v_n\rangle \in \Pi} \lfloor \mathsf{body}(p)(v_1, ..., v_n) \rfloor_{H,[]}$$

Note, $\langle\langle\ \Pi\ \rangle\rangle_H$ is a partial function, as it may not be well-defined if a predicate instance held in Π has an infinite unfolding.

2.5 Static Verification

Definition 1 (Denotational Formula Semantics).
Let $[\![\cdot]\!]$: FORMULA → $\mathcal{P}$(HEAP × ENV × PERMISSIONS) *be defined as*
$[\![\phi]\!] \stackrel{\text{def}}{=} \{\ \langle H, \rho, \Pi\rangle \in \text{HEAP} \times \text{ENV} \times \text{PERMISSIONS} \mid \langle H, \rho, \Pi\rangle \vDash_I \phi\ \}$

Definition 2 (Formula Satisfiability). *A formula* ϕ *is satisfiable if and only if* $[\![\phi]\!] \neq \emptyset$.
Let SATFORMULA ⊂ FORMULA *be the set of satisfiable formulas.*

Definition 3 (Formula Implication). $\phi_1 \Rightarrow \phi_2$ *if and only if* $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$

Definitions 1, 2, and 3 all rely on an iso-recursive style formula evaluation semantics, because formula satisfiability and implication must be statically implementable.

WLP Calculus
$\mathsf{acc}(e)$: EXPR → FORMULA and is defined in the following way:

$$\begin{aligned}
\mathsf{acc}(v) &= \texttt{true}\\
\mathsf{acc}(x) &= \texttt{true}\\
\mathsf{acc}(e_1 \odot e_2) &= \mathsf{acc}(e_1) \wedge \mathsf{acc}(e_2)\\
\mathsf{acc}(e_1 \oplus e_2) &= \mathsf{acc}(e_1) \wedge \mathsf{acc}(e_2)\\
\mathsf{acc}(e.f) &= \mathbf{acc}(\texttt{e.f})
\end{aligned}$$

$$
\begin{aligned}
&\mathsf{WLP}(x := e, \widehat{\phi}) && = \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi}[e/x] \quad \wedge \quad \widehat{\phi}' \Rightarrow \mathsf{acc}(e)\ \} \\
&\mathsf{WLP}(\texttt{while (e) inv } \widehat{\phi}_i \texttt{ \{ s \}}, \widehat{\phi}) && = \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi}_i \quad \wedge \quad \widehat{\phi}' * \texttt{(}e \texttt{ = false)} \Rightarrow \widehat{\phi}\ \} \\
&\mathsf{WLP}(\texttt{x.f := y}, \widehat{\phi}) && = \mathbf{acc}(\texttt{x.f}) * \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' * \mathbf{acc}(\texttt{x.f}) * \texttt{(}x.f \texttt{ = } y\texttt{)} \Rightarrow \widehat{\phi} \quad \wedge \\
& && \qquad \widehat{\phi}' * \mathbf{acc}(\texttt{x.f}) \in \textsc{SatFormula}\ \} \\
&\mathsf{WLP}(\texttt{x := new C}, \widehat{\phi}) && = \max_{\Rightarrow} \{\ \widehat{\phi}' \mid x \notin \mathsf{FV}(\widehat{\phi}') \quad \wedge \\
& && \widehat{\phi}' * x \neq \mathit{null} * \overline{x \neq e_i} * \overline{acc(x.f_i)} * \overline{x.f_i = \mathsf{defaultValue}(T_i)} \Rightarrow \widehat{\phi}\ \} \\
& && \texttt{where fields(C)} = \overline{T_i\ f_i} \texttt{ and } x \neq e_i \texttt{ is a conjunctive term in } \widehat{\phi} \\
&\mathsf{WLP}(y := z.m(\overline{x}), \widehat{\phi}) && = \max_{\Rightarrow} \{\ \widehat{\phi}' \mid y \notin \mathsf{FV}(\widehat{\phi}') \quad \wedge \\
& && \widehat{\phi}' \Rightarrow \texttt{(}z \texttt{ } \neq \texttt{ null)} * \mathsf{mpre}(m)[z/\texttt{this}, \overline{x/\mathsf{mparam}(m)}] \quad \wedge \\
& && \widehat{\phi}' * \mathsf{mpost}(m)[z/\texttt{this}, \overline{x/\texttt{old(}\mathsf{mparam}(m)\texttt{)}}, y/\texttt{result}] \Rightarrow \widehat{\phi}\ \} \\
&\mathsf{WLP}(\texttt{assert } \phi_a, \widehat{\phi}) && = \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi} \quad \wedge \quad \widehat{\phi}' \Rightarrow \phi_a\ \} \\
&\mathsf{WLP}(\texttt{fold } p\texttt{(}e_1\texttt{,...,}e_n\texttt{)}, \widehat{\phi}) && = \mathsf{body}(p)(e_1, ..., e_n) * \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' * p\texttt{(}e_1\texttt{,...,}e_n\texttt{)} \Rightarrow \widehat{\phi} \quad \wedge \\
& && \qquad \widehat{\phi}' * p\texttt{(}e_1\texttt{,...,}e_n\texttt{)} \in \textsc{SatFormula}\ \} \\
&\mathsf{WLP}(\texttt{unfold } p\texttt{(}e_1\texttt{,...,}e_n\texttt{)}, \widehat{\phi}) && = p\texttt{(}e_1\texttt{,...,}e_n\texttt{)} * \max_{\Rightarrow} \{\ \widehat{\phi}' \mid \widehat{\phi}' * \mathsf{body}(p)(e_1, ..., e_n) \Rightarrow \widehat{\phi} \quad \wedge \\
& && \qquad \widehat{\phi}' * \mathsf{body}(p)(e_1, ..., e_n) \in \textsc{SatFormula}\ \}
\end{aligned}
$$

Fig. 6. SVL_{RP}: Weakest liberal precondition calculus (selected rules), see Fig. 7 for missing rules

WLP Implementable Calculus

TODO ▶ *in progress...* ◀

$$
\begin{array}{ll}
\mathsf{WLP}(\texttt{skip}, \widehat{\phi}) & = \widehat{\phi} \\
\mathsf{WLP}(s_1\,;\, s_2, \widehat{\phi}) & = \mathsf{WLP}(s_1, \mathsf{WLP}(s_2, \widehat{\phi})) \\
\mathsf{WLP}(\texttt{T x}, \widehat{\phi}) & = \begin{cases} \widehat{\phi} & \texttt{if } x \notin \mathtt{FV}(\widehat{\phi}) \\ \texttt{undefined} & \texttt{otherwise} \end{cases} \\
\mathsf{WLP}(x := e, \widehat{\phi}) & = \overline{\mathbf{acc}(\mathtt{e_i.f_i}) \wedge}\ \widehat{\phi}[e/x] \\
 & \texttt{for } \langle \mathtt{e_i}, \mathtt{f_i} \rangle \in \mathtt{InferFieldAcc(e)} \\
\mathsf{WLP}(\texttt{if (e) then } \{\mathtt{s_1}\} \texttt{ else } \{\mathtt{s_2}\}, \widehat{\phi}) & = (\texttt{if } e \texttt{ then } \mathsf{WLP}(s_1, \widehat{\phi}) \texttt{ else } \mathsf{WLP}(s_2, \widehat{\phi})) \\
\mathsf{WLP}(\texttt{while (e) inv } \widehat{\phi}_i \texttt{ \{ s \}}, \widehat{\phi}) & = \widehat{\phi}_i \\
\mathsf{WLP}(\texttt{x.f := y}, \widehat{\phi}) & = \mathbf{acc}(\mathtt{x.f}) \wedge \widehat{\phi}[y/x.f] \\
\mathsf{WLP}(\texttt{x := new C}, \widehat{\phi}) & = \begin{cases} \mathtt{MinusVar}(\widehat{\phi}, \mathtt{x}) & \texttt{if } \mathtt{MinusVar}(\widehat{\phi}, \mathtt{x}) * \mathtt{x} \neq \mathtt{null} \\ & \quad * \overline{\mathtt{x} \neq \mathtt{e_i}} * \overline{\mathtt{acc(x.f_i)}} \implies \widehat{\phi} \\ \texttt{undefined} & \texttt{otherwise} \end{cases} \\
 & \texttt{where } \mathtt{fields(C)} = \overline{\mathtt{T_i\ f_i}} \texttt{ and } \mathtt{x} \neq \mathtt{e_i} \texttt{ is a conjunctive term in } \widehat{\phi} \\
\mathsf{WLP}(y := z.m(\overline{x}), \widehat{\phi}) & = \\
\mathsf{WLP}(\texttt{assert } \phi_a, \widehat{\phi}) & = \widehat{\phi} \wedge \overline{\mathbf{acc}(\mathtt{e_i.f_i}) \wedge} \\
 & \quad \overline{(\texttt{if } e_j \texttt{ then } \overline{\mathbf{acc}(\mathtt{e_k.f_k}) \wedge} \texttt{ else } \overline{\mathbf{acc}(\mathtt{e_1.f_1}) \wedge})} \wedge \phi_a \\
 & \texttt{for } \langle \mathtt{e_i}, \mathtt{f_i} \rangle \in \mathtt{InferFieldAcc}(\phi_a), \\
 & \langle \mathtt{e_j}, \{\overline{\langle \mathtt{e,f} \rangle}\}_{1_j}, \{\overline{\langle \mathtt{e,f} \rangle}\}_{2_j} \rangle \in \mathtt{InferFieldAcc}(\phi_a), \\
 & \langle \mathtt{e_k}, \mathtt{f_k} \rangle \in \{\overline{\langle \mathtt{e,f} \rangle}\}_{1_j}, \texttt{ and } \langle \mathtt{e_1}, \mathtt{f_1} \rangle \in \{\overline{\langle \mathtt{e,f} \rangle}\}_{2_j} \\
\mathsf{WLP}(\texttt{fold } p(e_1, \ldots, e_n), \widehat{\phi}) & = \\
\mathsf{WLP}(\texttt{unfold } p(e_1, \ldots, e_n), \widehat{\phi}) & =
\end{array}
$$

Fig. 7. $\mathrm{SVL}_{\mathrm{RP}}$: Weakest liberal precondition calculus

Notes:

- accessibility predicates should be ordered properly when removing them from InferFieldAcc computation

Helper Defs for WLP Implementable Calculus

$\texttt{InferFieldAcc} : \text{FORMULA} \to \mathcal{P}((\text{EXPR} \times \text{FIELDNAME}) \cup (\text{EXPR} \times \mathcal{P}(\text{EXPR} \times \text{FIELDNAME}) \times \mathcal{P}(\text{EXPR} \times \text{FIELDNAME})))$

$$
\begin{aligned}
&\texttt{InferFieldAcc}(\texttt{v}) &&= \emptyset \\
&\texttt{InferFieldAcc}(\texttt{x}) &&= \emptyset \\
&\texttt{InferFieldAcc}\big(\texttt{e}_1 \odot \texttt{e}_2\big) &&= \texttt{InferFieldAcc}\big(\texttt{e}_1\big) \cup \texttt{InferFieldAcc}\big(\texttt{e}_2\big) \\
&\texttt{InferFieldAcc}\big(\texttt{e}_1 \oplus \texttt{e}_2\big) &&= \texttt{InferFieldAcc}\big(\texttt{e}_1\big) \cup \texttt{InferFieldAcc}\big(\texttt{e}_2\big) \\
&\texttt{InferFieldAcc}(\texttt{e.f}) &&= \texttt{InferFieldAcc}(\texttt{e}) \cup \{\langle \texttt{e}, \texttt{f} \rangle\} \\
&\texttt{InferFieldAcc}(\textbf{acc}(\texttt{e.f})) &&= \texttt{InferFieldAcc}(\texttt{e}) \\
&\texttt{InferFieldAcc}\big(\phi_1 \wedge \phi_2\big) &&= \texttt{InferFieldAcc}\big(\phi_1\big) \cup \texttt{InferFieldAcc}\big(\phi_2\big) \\
&\texttt{InferFieldAcc}\big(\phi_1 * \phi_2\big) &&= \texttt{InferFieldAcc}\big(\phi_1\big) \cup \texttt{InferFieldAcc}\big(\phi_2\big) \\
&\texttt{InferFieldAcc}\big(\texttt{p}(\texttt{e}_1, ..., \texttt{e}_n)\big) &&= \texttt{InferFieldAcc}\big(\texttt{e}_1\big) \cup ... \cup \texttt{InferFieldAcc}\big(\texttt{e}_n\big) \\
&\texttt{InferFieldAcc}\big(\texttt{(if e then } \phi_1 \texttt{ else } \phi_2\texttt{)}\big) &&= \texttt{InferFieldAcc}(\texttt{e}) \cup \\
&&&\quad \big\{\langle \texttt{e}, \texttt{InferFieldAcc}\big(\phi_1\big), \texttt{InferFieldAcc}\big(\phi_2\big)\rangle\big\} \\
&\texttt{InferFieldAcc}\big(\texttt{(unfolding p}(\texttt{e}_1, ..., \texttt{e}_n) \texttt{ in } \phi\texttt{)}\big) &&= \texttt{InferFieldAcc}\big(\texttt{e}_1\big) \cup ... \cup \texttt{InferFieldAcc}\big(\texttt{e}_n\big) \cup \\
&&&\quad \texttt{InferFieldAcc}(\phi)
\end{aligned}
$$

$\texttt{MinusVar} : \text{FORMULA} \times \text{VAR} \to \text{FORMULA}$

$$
\begin{aligned}
&\texttt{MinusVar}(\texttt{e}, \texttt{x}) &&= \begin{cases} \texttt{true} & \texttt{if x} \in \mathsf{FV}(\texttt{e}) \\ \texttt{e} & \texttt{otherwise} \end{cases} \\
&\texttt{MinusVar}(\textbf{acc}(\texttt{e.f}), \texttt{x}) &&= \begin{cases} \texttt{true} & \texttt{if x} \in \mathsf{FV}(\texttt{e}) \\ \textbf{acc}(\texttt{e.f}) & \texttt{otherwise} \end{cases} \\
&\texttt{MinusVar}\big(\phi_1 \wedge \phi_2, \texttt{x}\big) &&= \texttt{MinusVar}\big(\phi_1, \texttt{x}\big) \wedge \texttt{MinusVar}\big(\phi_2, \texttt{x}\big) \\
&\texttt{MinusVar}\big(\phi_1 * \phi_2, \texttt{x}\big) &&= \texttt{MinusVar}\big(\phi_1, \texttt{x}\big) * \texttt{MinusVar}\big(\phi_2, \texttt{x}\big) \\
&\texttt{MinusVar}\big(\texttt{p}(\texttt{e}_1, ..., \texttt{e}_n), \texttt{x}\big) &&= \begin{cases} \texttt{true} & \texttt{if x} \in \mathsf{FV}(\texttt{e}_1) \vee ... \vee \texttt{x} \in \mathsf{FV}(\texttt{e}_n) \\ \texttt{p}(\texttt{e}_1, ..., \texttt{e}_n) & \texttt{otherwise} \end{cases} \\
&\texttt{MinusVar}\big(\texttt{(if e then } \phi_1 \texttt{ else } \phi_2\texttt{)}, \texttt{x}\big) &&= \\
&&& \begin{cases} \texttt{true} & \texttt{if x} \in \mathsf{FV}(\texttt{e}) \\ \texttt{(if } e \texttt{ then MinusVar}\big(\phi_1, \texttt{x}\big) \texttt{ else MinusVar}\big(\phi_2, \texttt{x}\big)\texttt{)} & \texttt{otherwise} \end{cases} \\
&\texttt{MinusVar}\big(\texttt{(unfolding p}(\texttt{e}_1, ..., \texttt{e}_n) \texttt{ in } \phi\texttt{)}, \texttt{x}\big) = \\
&&& \begin{cases} \texttt{MinusVar}(\phi, \texttt{x}) & \texttt{if x} \in \mathsf{FV}(\texttt{e}_1) \vee ... \vee \texttt{x} \in \mathsf{FV}(\texttt{e}_n) \\ \texttt{(unfolding p}(\texttt{e}_1, ..., \texttt{e}_n) \texttt{ in MinusVar}(\phi, \texttt{x})\texttt{)} & \texttt{otherwise} \end{cases}
\end{aligned}
$$

[TO WRITE:]

- FV function to compute the set of free variables in a formula
- Should define substitution on a formula

Definition 4 (Valid method). *A method with contract* `requires` $\widehat{\phi}_p$ `ensures` $\widehat{\phi}_q$*, parameters* $\overline{x}$*, and body* s *is considered valid if* $\widehat{\phi}_p \Rightarrow \mathsf{WLP}(s, \widehat{\phi}_q)[\overline{x/\texttt{old}(x)}]$ *holds.*

Definition 5 (Valid program). *A program with entry point statement* s *is considered valid if* `true` $\Rightarrow \mathsf{WLP}(s, \texttt{true})$ *holds and all methods are valid.*

2.6 Soundness

Definition 6 (Valid state). *We call the state* $\langle H, \langle \rho_n, \pi_n, s_n\rangle \cdot ... \cdot \langle \rho_1, \pi_1, s_1\rangle \cdot \mathsf{nil}\rangle \in$ STATE valid *if* $\langle H, \rho_i, \pi_i\rangle \vDash_E \mathsf{sWLP}_i(s_n \cdot ... \cdot s_1 \cdot \mathsf{nil}, \texttt{true})$ *for all* $1 \leq i \leq n$.

Proposition 1 ($\mathbf{SVL_{RP}}$ Progress). *If* $\psi \in$ STATE *is a valid state and* $\psi \notin \{\langle H, \langle \rho, \pi, skip\rangle \cdot nil\rangle \mid H \in \text{HEAP}, \rho \in \text{ENV}, \pi \in \text{DYNFPRINT}\}$ *then* $\psi \longrightarrow \psi'$ *for some* $\psi' \in$ STATE.

Proposition 2 ($\mathbf{SVL_{RP}}$ Preservation). *If* ψ *is a valid state and* $\psi \longrightarrow \psi'$ *for some* $\psi' \in$ STATE *then* ψ' *is a valid state.*

We define sWLP for SVL_{RP} in a similar fashion as in SVL, but with the addition of framing rules as shown in Fig. 8. sWLP^m is equivalent to sWLP, except that it weakens the top most formula just enough such that the heap locations it mentions are *disjoint* from those acquired by m. Effectively, $\widehat{\phi}_f$ represents the implicit frame of the executing function, so ownership of all fields mentioned in $\widehat{\phi}_f$ is withdrawn from the call site.

$$\mathsf{sWLP}(s \cdot \mathsf{nil}, \widehat{\phi}) = \mathsf{WLP}(s, \widehat{\phi}) \cdot \mathsf{nil}$$

$$\mathsf{sWLP}(s \cdot (y := z.m(\overline{x})\ ;\ s') \cdot \overline{s}, \widehat{\phi}) = \mathsf{WLP}(s, \mathsf{mpost}(m)) \cdot \mathsf{sWLP}^{\mathsf{mpre}(m)[z/\texttt{this}, \overline{x/\mathsf{mparam}(m)}]}((y := z.m(\overline{x})\ ;\ s') \cdot \overline{s}, \widehat{\phi})$$

$$\mathsf{sWLP}(s \cdot (\texttt{while (e) inv } \widehat{\phi}_{\mathtt{i}} \texttt{ \{ r \}; } s') \cdot \overline{s}, \widehat{\phi}) = \mathsf{WLP}(s, \widehat{\phi}_i) \cdot \mathsf{sWLP}^{\widehat{\phi}_i}((\texttt{while (e) inv } \widehat{\phi}_{\mathtt{i}} \texttt{ \{ r \}; } s') \cdot \overline{s}, \widehat{\phi})$$

where

$$\mathsf{sWLP}^{\widehat{\phi}_f}(\overline{s}, \widehat{\phi}) = \min_{\Rightarrow}\{\ \widehat{\phi}'_n \mid \widehat{\phi}_n \Rightarrow \widehat{\phi}_f * \widehat{\phi}'_n\ \} \cdot \widehat{\phi}_{n-1} \cdot ... \cdot \widehat{\phi}_1 \cdot \mathsf{nil}$$

where $\quad \widehat{\phi}_n \cdot \widehat{\phi}_{n-1} \cdot ... \cdot \widehat{\phi}_1 \cdot \mathsf{nil} = \mathsf{sWLP}(\overline{s}, \widehat{\phi})$

Fig. 8. Heap aware weakest precondition across call boundaries

3 GVL_{RP}

3.1 Abstract Syntax

$$
\begin{array}{rll}
x, y, z & \in VAR & \textit{(variables)} \\
v & \in VAL & \textit{(values)} \\
e & \in EXPR & \textit{(expressions)} \\
s & \in STMT & \textit{(statements)} \\
o & \in LOC & \textit{(object Ids)} \\
f & \in FIELDNAME & \textit{(field names)} \\
m & \in METHODNAME & \textit{(method names)} \\
C & \in CLASSNAME & \textit{(class names)} \\
p & \in PREDNAME & \textit{(predicate names)}
\end{array}
$$

$$
\begin{array}{rll}
\texttt{P} & ::= & \overline{\texttt{cls}}\ \texttt{s} \\
\texttt{cls} & ::= & \texttt{class C \{ } \overline{\texttt{field}}\ \overline{\texttt{pred}}\ \overline{\texttt{method}}\ \texttt{\}} \\
\texttt{field} & ::= & \texttt{T f;} \\
\texttt{pred} & ::= & \texttt{predicate } p(\overline{T\ x}) = \widetilde{\phi} \\
\texttt{T} & ::= & \texttt{int} \mid C \mid \texttt{T} \\
\texttt{method} & ::= & \texttt{T m(}\overline{\texttt{T x}}\texttt{) contract \{s\}} \\
\texttt{contract} & ::= & \texttt{requires } \widetilde{\phi}\ \texttt{ensures } \widetilde{\phi} \\
\oplus & ::= & + \mid - \mid * \mid \backslash \\
\odot & ::= & \neq \mid = \mid < \mid > \mid \leq \mid \geq \\
s & ::= & \texttt{skip} \mid s_1\texttt{;}\ s_2 \mid \texttt{T x} \mid x := e \mid \texttt{if (e) then \{}s_1\texttt{\} else \{}s_2\texttt{\}} \\
 & & \mid \texttt{while (e) inv } \widetilde{\phi}\ \texttt{\{ s \}} \mid \texttt{x.f := y} \mid \texttt{x := new C} \mid y := z.m(\overline{x}) \\
 & & \mid \texttt{assert } \phi \mid \texttt{fold } p(\overline{e}) \mid \texttt{unfold } p(\overline{e}) \\
e & ::= & v \mid x \mid e \oplus e \mid e \odot e \mid e.f \\
x & ::= & \texttt{result} \mid id \mid \texttt{old(}id\texttt{)} \mid \texttt{this} \\
v & ::= & n \mid o \mid \texttt{null} \mid \texttt{true} \mid \texttt{false} \\
\phi & ::= & e \mid p(\overline{e}) \mid acc(e.f) \mid \phi \wedge \phi \mid \phi * \phi \mid \texttt{(if } e \texttt{ then } \phi \texttt{ else } \phi\texttt{)} \\
 & & \mid \texttt{(unfolding } p(\overline{e}) \texttt{ in } \phi\texttt{)} \\
\widetilde{\phi} & ::= & \phi \mid ? * \phi
\end{array}
$$

Note:

- Contracts, predicates, and loop invariants are defined with gradual formulas
- We do not allow conditionals to contain gradual formulas to simplify the formalisms
- It does not make sense to allow imprecise formulas in the *in* part of (**unfolding** $p(\overline{e})$ **in** ϕ), because it addresses framing of ϕ by ... [FINISH]

JW ▶$? * \phi$ *or* $? \wedge \phi$ *? Meaning shouldn't change between them.*◀

4 GVL_{RP}: Static Semantics

4.1 Interpretation of Gradual Formulas

We adjust concretization as defined for SVL to consider self-framed formulas as defined in Section 5.1:

Definition 7 (Concretization of gradual formulas). $\gamma : \widetilde{\text{Formula}} \rightharpoonup \mathcal{P}^{\text{Formula}}$ *is defined as:*

$$\gamma(\widehat{\phi}) = \{\ \widehat{\phi}\ \}$$
$$\gamma(? * \phi) = \{\ \widehat{\phi}' \in \text{SatFormula} \mid \widehat{\phi}' \Rightarrow \phi\ \} \quad \textit{if } \phi \in \text{SatFormula}$$
$$\gamma(? * \phi) \quad \textit{undefined otherwise}$$

Note, however, that we do not require the static part of $? * \phi$ to be self-framed. This is because we want the unknown part of a gradual formula to be able to provide the necessary framing, hence allowing gradual verification regarding heap permissions. This allows programmers to resort to gradual formulas when being uncertain or indifferent about the concrete framing of a heap-dependent formula.

Imprecise predicate bodies. While definition 7 concretizes to syntactically precise formulas, those formulas may still hide imprecision by referencing predicates with imprecise bodies. We have to extend the concept of concretization to also cover predicates in order for γ to be a full bridge between gradual and static verification language.

The fundamental difficulty of SVL_{RP} compared to SVL is the fact that formulas are no longer "local", *i.e.* in order to fully interpret them, one requires some additional context body (looking up predicates from the ambient program, see EvPred in Figure 2). For concretizing a formula ϕ referencing some predicates, it is necessary to *change* this context (*e.g.* replace imprecise predicate bodies with some precise one). While so far, body was the implicit context of all formulas, we will now work with "local formulas" $\langle \phi, body \rangle \in \text{Formula} \times (\text{PredName} \to \text{Expr}^* \to \text{SfrmFormula})$ that explicitly drag along their context. Formulas drawn from the program source (loop invariants, contracts, predicate bodies) are paired with the body corresponding to the predicates as they are declared in the source. Existing rules can easily be adjusted in order to deal with this parametricity, for example:

$$\frac{\langle H, \rho, \pi_1 \rangle \vDash_E \langle \phi_1, body \rangle \qquad \langle H, \rho, \pi_2 \rangle \vDash_E \langle \phi_2, body \rangle}{\langle H, \rho, (\pi_1 \uplus \pi_2) \rangle \vDash_E \langle \phi_1 * \phi_2, body \rangle}\ \text{EvSepOp}$$

$$\frac{body(p)(e_1, ..., e_n) = \phi \qquad \langle H, \rho, \pi \rangle \vDash_E \langle \phi, body \rangle}{\langle H, \rho, \pi \rangle \vDash_E \langle p(e_1, ..., e_n), body \rangle}\ \text{EvPred}$$

As shown in both rules, the context *body* is always forwarded unmodified, simply making explicit what was previously assumed as constant and ambient. EvPred uses *body* to lookup predicates, rather than using the designated body. We can now express concretization that also handles predicates appropriately.

Definition 8 (Concretization of gradual formulas (continued)). *Concretization of a set of predicates* $\gamma : (\textsc{PredName} \to \textsc{Expr}^* \to \widetilde{\textsc{Formula}}) \rightharpoonup \mathcal{P}^{\textsc{PredName}\to\textsc{Expr}^*\to\textsc{SfrmFormula}}$ *is defined as:*

$$\gamma(\widetilde{body}) = \{\ \lambda p \in \mathsf{dom}(\widetilde{body}).\ \lambda \overline{e} \in \textsc{Expr}^*.\ \widehat{\phi}_p[\overline{e}/\overline{\texttt{tmp}}] \mid \langle \widehat{\phi}_{p_1}, \widehat{\phi}_{p_2} ... \rangle \in \gamma(\widetilde{body}(p_1)(\overline{\texttt{tmp}_1})) \times \gamma(\widetilde{body}(p_2)(\overline{\texttt{tmp}_2})) \times ...\ \}$$

where $\mathsf{dom}(\widetilde{body}) = \{\ p_1, p_2, ...\ \} \subseteq \textsc{PredName}$

Given this function, we can concretize a formula and its context, yielding a set of truly (syntactically and semantically) precise formulas:

$$\gamma(\langle \widetilde{\phi}, \widetilde{body} \rangle) = \{\ \langle \widehat{\phi}, body \rangle \mid \widehat{\phi} \in \gamma(\widetilde{\phi}), body \in \gamma(\widetilde{body})\ \}$$

As before, the definition of concretization is a direct formalization of the *plausibility interpretation* of gradual verification: A gradual judgment must hold if it is plausible that all involved imprecise formulas can be replaced by a concrete formula that renders the judgment true. Since formulas may reference imprecise predicates, they must necessarily be treated by γ as well.

4.2 Lifting Predicates

Lemma 1 (Consistent Formula Evaluation).
Let $\cdot \ \widetilde{\models}\ \cdot \subseteq \textsc{Mem} \times \widetilde{\textsc{Formula}}$ *be defined inductively as*

$$\frac{body = \lambda p \in \mathsf{dom}(\widetilde{body}).\ \lambda \overline{e} \in \textsc{Expr}^*.\ \mathsf{static}(\widetilde{body}(p)(\overline{e})) \qquad \langle H, \rho, \pi \rangle \models_E \langle \phi, body \rangle \qquad \vdash^{H,\rho,\pi}_{\texttt{frmE}} \langle \phi, body \rangle}{\langle H, \rho, \pi \rangle \ \widetilde{\models}\ \langle ? * \phi, \widetilde{body} \rangle}\ \widetilde{\textsc{EvalGrad}}$$

$$\frac{body = \lambda p \in \mathsf{dom}(\widetilde{body}).\ \lambda \overline{e} \in \textsc{Expr}^*.\ \mathsf{static}(\widetilde{body}(p)(\overline{e})) \qquad \langle H, \rho, \pi \rangle \models_E \langle \widehat{\phi}, body \rangle \qquad \vdash^{H,\rho,\pi}_{\texttt{frmE}} \langle \widehat{\phi}, body \rangle}{\langle H, \rho, \pi \rangle \ \widetilde{\models}\ \langle \widehat{\phi}, \widetilde{body} \rangle}\ \widetilde{\textsc{EvalStatic}}$$

$\cdot \ \widetilde{\models}\ \cdot$ *is a consistent lifting of* $\cdot \models_E \cdot$.

Note, since SVL_{RP}'s equi-recursive evaluation semantics treats predicate instances as their body, we must carrying around and modify the *body* context when performing equi-recursive evaluation in the gradual setting to evaluate imprecise predicate bodies appropriately. This is not necessary for a consistent lifting of an iso-recursive evaluation semantics. TODO ▶ *Talk about why evalstatic requires framing even though formula self-framing* ◀

Additionally, framing is defined equi-recursively. The equi-recursive style framing judgment is defined similarly (replacing Π with π and iso-recursive formula evaluation with equi-recursive formula evaluation) to its iso-recursive counterpart in SVL_{RP} for all rules except FrmPred and FrmUnfolding (Fig. 9).

We also adjust the equi-recursive framing judgment to pass around and use a *body* context, as described in Sec. 4.1.

$$\frac{\vdash^{H,\rho,\pi}_{\texttt{frmE}} e_1 \quad \ldots \quad \vdash^{H,\rho,\pi}_{\texttt{frmE}} e_n \quad \vdash^{H,\rho,\pi}_{\texttt{frmE}} \mathsf{body}(p)(e_1,\ldots,e_n)}{\vdash^{H,\rho,\pi}_{\texttt{frmE}} p(e_1,\ldots,e_n)}\ \textsc{FrmPred}$$

$$\frac{\vdash^{H,\rho,\pi}_{\texttt{frmE}} e_1 \quad \ldots \quad \vdash^{H,\rho,\pi}_{\texttt{frmE}} e_n \quad \vdash^{H,\rho,\pi}_{\texttt{frmE}} \phi}{\vdash^{H,\rho,\pi}_{\texttt{frmE}} (\texttt{unfolding}\ p(e_1,\ldots,e_n)\ \texttt{in}\ \phi)}\ \textsc{FrmUnfolding}$$

Fig. 9. $\mathrm{GVL}_{\mathrm{RP}}$: Equi-recursive Framing (selected rules)

Lemma 2 (Consistent Formula Implication).
Let $\cdot \mathrel{\widetilde{\Rightarrow}} \cdot \subseteq \widetilde{\textsc{Formula}} \times \widetilde{\textsc{Formula}}$ *be defined inductively as*

$$\frac{\hat{\phi}_1 \Rightarrow \mathsf{static}(\widetilde{\phi}_2)}{\hat{\phi}_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2}\ \widetilde{\textsc{ImplStatic}} \qquad \frac{\hat{\phi} \in \textsc{SatFormula} \quad \hat{\phi} \Rightarrow \phi_1 \quad \hat{\phi} \Rightarrow \mathsf{static}(\widetilde{\phi}_1)}{? * \phi_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2}\ \widetilde{\textsc{ImplGrad}}$$

$\cdot \mathrel{\widetilde{\Rightarrow}} \cdot$ *is a consistent lifting of* $\cdot \Rightarrow \cdot$.

In contrast to the definition of consistent formula evaluation (Lemma 1) and concretization in definition 8, predicate imprecision actually plays no significant role. Instead, predicates are handled *on the fly* rather than requiring any form of upfront concretization.

4.3 Lifting Functions

We must have corresponding abstraction functions for our concretization definitions to consistently lift (partial) functions. TODO ▶*fix definitions to handle argument sets, which contain unsatisfiable formulas - think about wlp def during this*◀

Definition 9 (Precision of formulas). $\widetilde{\phi}_1$ *is more precise than* $\widetilde{\phi}_2$*, written* $\widetilde{\phi}_1 \sqsubseteq \widetilde{\phi}_2$ *if and only if* $\gamma(\widetilde{\phi}_1) \subseteq \gamma(\widetilde{\phi}_2)$

Definition 10 (Abstraction of formulas). *Let* $\alpha : \mathcal{P}^{\textsc{SatFormula}} \rightharpoonup \widetilde{\textsc{Formula}}$ *be defined as* $\alpha(\overline{\phi}) = \min_{\sqsubseteq} \{\ \widetilde{\phi} \in \widetilde{\textsc{Formula}} \mid \overline{\phi} \subseteq \gamma(\widetilde{\phi})\ \}$

Definition 11 (Precision of formulas (continued)). $\langle \widetilde{\phi}_1, \widetilde{body_1} \rangle$ *is more precise than* $\langle \widetilde{\phi}_2, \widetilde{body_2} \rangle$*, written* $\langle \widetilde{\phi}_1, \widetilde{body_1} \rangle \sqsubseteq \langle \widetilde{\phi}_2, \widetilde{body_2} \rangle$ *if and only if* $\gamma(\langle \widetilde{\phi}_1, \widetilde{body_1} \rangle) \subseteq \gamma(\langle \widetilde{\phi}_2, \widetilde{body_2} \rangle)$

Definition 12 (Abstraction of formulas (continued)). *Let* $\alpha : \mathcal{P}^{\textsc{SatFormula} \times (\textsc{PredName} \to \textsc{Expr}^* \to \textsc{SfrmFormula})} \rightharpoonup \widetilde{\textsc{Formula}} \times (\textsc{PredName} \to \textsc{Expr}^* \to \widetilde{\textsc{Formula}})$ *be defined as* $\alpha(\overline{\langle \phi, body \rangle}) = \min_{\sqsubseteq} \{\ \langle \widetilde{\phi}, \widetilde{body} \rangle \mid \overline{\langle \phi, body \rangle} \subseteq \gamma(\langle \widetilde{\phi}, \widetilde{body} \rangle)\ \}$

TODO ▶*make the above fit & flow better in this section*◀

Figure 10 shows how WLP is consistently lifted. Method calls and while loops reference/contain further formulas to consider.

Recall how we adjusted sWLP (Fig. 8) in order for it to be usable for formalizing small step soundness. Figure 11 shows the corresponding definition for GVL_{RP}. Note how full imprecision (in a function precondition or loop invariant) makes it impossible to make assumptions about which access is retained at the call site or across the loop. We conservatively approximate the frame by assuming that no permissions remain: For $\widehat{\phi}'_n$, an empty footprint is enforced, which due to self-framing of $\widehat{\phi}'_n$ also prohibits the *use* of any fields. TODO ▶*note about footprint splitting and it mirroring the dynamic semantics fp splitting*◀

$$\widetilde{\mathsf{WLP}}(s_1\,;\, s_2, \widetilde{\phi}) = \widetilde{\mathsf{WLP}}(s_1, \widetilde{\mathsf{WLP}}(s_2, \widetilde{\phi}))$$

$$\widetilde{\mathsf{WLP}}(\texttt{if (e) then } \{\texttt{s}_1\} \texttt{ else } \{\texttt{s}_2\}, \widetilde{\phi}) = \alpha(\{\ (\texttt{if } e \texttt{ then } \widehat{\phi}_1 \texttt{ else } \widehat{\phi}_2)$$
$$\mid \widehat{\phi}_1 \in \gamma(\widetilde{\mathsf{WLP}}(s_1, \widetilde{\phi})), \widehat{\phi}_2 \in \gamma(\widetilde{\mathsf{WLP}}(s_2, \widetilde{\phi}))\ \})$$

$$\widetilde{\mathsf{WLP}}(y := z.m(\overline{x}), \widetilde{\phi}) = \alpha(\{\ \max_{\Rightarrow}\ \{\ \widehat{\phi}' \mid y \notin \mathsf{FV}(\widehat{\phi}') \quad \wedge$$
$$\widehat{\phi}' \Rightarrow (z \neq \texttt{null}) * \widehat{\phi}_p[z/\texttt{this}, \overline{x/\mathsf{mparam}(m)}] \quad \wedge$$
$$\widehat{\phi}' * \widehat{\phi}_q[z/\texttt{this}, \overline{x/\texttt{old}(\mathsf{mparam}(m))}, y/\texttt{result}] \Rightarrow \widehat{\phi}\ \}$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_p \in \gamma(\mathsf{mpre}(m)),\ \widehat{\phi}_q \in \gamma(\mathsf{mpost}(m))\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{while (e) inv } \widetilde{\phi}_i\ \{\ \texttt{s}\ \}, \widetilde{\phi}) = \alpha(\{\ \max_{\Rightarrow}\ \{\ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi}_i \quad \wedge \quad \widehat{\phi}' * (e \texttt{ = false}) \Rightarrow \widehat{\phi}\ \}$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_i \in \gamma(\widetilde{\phi}_i)\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{fold } p(e_1, \ldots, e_n), \widetilde{\phi}) = \alpha(\{\ \widehat{\phi}_b * \max_{\Rightarrow}\ \{\ \widehat{\phi}' \mid \widehat{\phi}' * p(e_1, \ldots, e_n) \Rightarrow \widehat{\phi} \quad \wedge$$
$$\widehat{\phi}' * p(e_1, \ldots, e_n) \in \textsc{SatFormula}\ \}$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_b \in \gamma(\mathsf{body}(p)(e_1, \ldots, e_n))\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{unfold } p(e_1, \ldots, e_n), \widetilde{\phi}) = \alpha(\{\ p(e_1, \ldots, e_n) * \max_{\Rightarrow}\ \{\ \widehat{\phi}' \mid \widehat{\phi}' * \widehat{\phi}_b \Rightarrow \widehat{\phi} \quad \wedge$$
$$\widehat{\phi}' * \widehat{\phi}_b \in \textsc{SatFormula}\ \}$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_b \in \gamma(\mathsf{body}(p)(e_1, \ldots, e_n))\ \})$$

$$\widetilde{\mathsf{WLP}}(s, \widetilde{\phi}) = \alpha(\{\ \mathsf{WLP}(s, \widehat{\phi}) \mid \widehat{\phi} \in \gamma(\widetilde{\phi})\ \}) \quad \text{otherwise}$$

Fig. 10. GVL_{IDF}: Weakest preconditions.

$$\widetilde{\mathsf{sWLP}}(s \cdot \mathsf{nil}, \widetilde{\phi}) = \widetilde{\mathsf{WLP}}(s, \widetilde{\phi}) \cdot \mathsf{nil}$$

$$\widetilde{\mathsf{sWLP}}(s \cdot (y := z.m(\overline{x})\ ;\ s') \cdot \overline{s}, \widetilde{\phi}) = \widetilde{\mathsf{WLP}}(s, \mathsf{mpost}(m)) \cdot \widetilde{\mathsf{sWLP}}^{\mathsf{mpre}(m)[z/\mathtt{this}, \overline{x/\mathsf{mparam}(m)}]}((y := z.m(\overline{x})\ ;\ s') \cdot \overline{s}, \widetilde{\phi})$$

$$\widetilde{\mathsf{sWLP}}(s \cdot (\mathtt{while}\ (\mathtt{e})\ \mathtt{inv}\ \widetilde{\phi}_i\ \{\ \mathtt{r}\ \}\ ;\ s') \cdot \overline{s}, \widetilde{\phi}) = \widetilde{\mathsf{WLP}}(s, \widetilde{\phi}_i) \cdot \widetilde{\mathsf{sWLP}}^{\widetilde{\phi}_i}((\mathtt{while}\ (\mathtt{e})\ \mathtt{inv}\ \widetilde{\phi}_i\ \{\ \mathtt{r}\ \}\ ;\ s') \cdot \overline{s}, \widetilde{\phi})$$

where

$$\widetilde{\mathsf{sWLP}}^{\widetilde{\phi}_f}(\overline{s}, \widetilde{\phi}) = \begin{cases} \widehat{\phi}_n \cdot \widetilde{\phi}_{n-1} \cdot \ldots \cdot \widetilde{\phi}_1 \cdot \mathsf{nil} & \text{if } \widetilde{\phi}_f \text{ and } \widetilde{\phi}_n \text{ top-level precise} \\ ? * \widehat{\phi}_n \cdot \widetilde{\phi}_{n-1} \cdot \ldots \cdot \widetilde{\phi}_1 \cdot \mathsf{nil} & \text{otherwise} \end{cases}$$

$$\text{where } \widetilde{\phi}_n \cdot \widetilde{\phi}_{n-1} \cdot \ldots \cdot \widetilde{\phi}_1 \cdot \mathsf{nil} = \widetilde{\mathsf{sWLP}}(\overline{s}, \widetilde{\phi})$$

$$\widehat{\phi}_n = \begin{cases} \min_{\Rightarrow}\{\ \widehat{\phi}'_n \mid \mathsf{static}(\widetilde{\phi}_n) \Rightarrow \widehat{\phi}'_n * \widehat{\phi}_f\ \} & \text{if } \widetilde{\phi}_f \text{ completely precise} \\ \min_{\Rightarrow}\{\ \widehat{\phi}'_n \mid \mathsf{static}(\widetilde{\phi}_n) \Rightarrow \widehat{\phi}'_n \wedge \forall H, \rho. \lfloor \widehat{\phi}'_n \rfloor_{H,\rho} = \emptyset\ \} & \text{otherwise} \end{cases}$$

Fig. 11. Stack aware weakest precondition across call boundaries.

JA ▶*if using $\overline{s}$ notation remove use of* nil◀

Definition 13 (Valid method). *A method with contract* `requires` $\widetilde{\phi}_p$ `ensures` $\widetilde{\phi}_q$*, parameters* $\overline{x}$*, and body* s *is considered valid if* $\widetilde{\phi}_p \mathrel{\widetilde{\Rightarrow}} \widetilde{\mathsf{WLP}}(s, \widetilde{\phi}_q)[\overline{x/\mathtt{old}(x)}]$ *holds.*

Definition 14 (Valid program). *A program with entry point statement* s *is considered valid if* `true` $\mathrel{\widetilde{\Rightarrow}} \widetilde{\mathsf{WLP}}(s, \mathtt{true})$ *holds and all methods are valid.*

5 $\mathrm{GVL}_{\mathrm{RP}}$: Dynamic Semantics

5.1 Footprints and Framing

TODO ▶*make sure these definitions are consistent liftings if they need to be*◀

Framing. Unlike GVL, we must redefine framing and self-framing for $\mathrm{GVL}_{\mathrm{RP}}$ to handle potential imprecision in predicate definitions. TODO ▶*provide an example of a formula we would like to be self-framing even with imprecise predicate definitions*◀ We define formula framing in $\mathrm{GVL}_{\mathrm{RP}}$ ($\widetilde{\vdash}^{H,\rho,\Pi}_{\mathtt{frmI}}\ \phi$) in the same way as $\mathrm{SVL}_{\mathrm{RP}}$ except for the FRMUNFOLDING rule, redefined in Figure 12. The difference between the two rules is in the computation Π' for framing ϕ. New footprint definitions (defined in this section) are used in this computation. TODO ▶*explain the difference in more detail*◀ A formula ϕ is called *self-framed* (we write $\widetilde{\vdash}_{\mathtt{frm}}\ \phi$) if $\langle H, \rho, \Pi\rangle \vDash_I \phi$ implies $\widetilde{\vdash}^{H,\rho,\Pi}_{\mathtt{frmI}}\ \phi$ (for all H, ρ, Π). We redefine the set of self-framed formulas: $\textsc{SfrmFormula} \stackrel{\text{def}}{=} \{\ \phi \in \textsc{Formula} \mid \widetilde{\vdash}_{\mathtt{frm}}\ \phi\ \}$. We still write $\widehat{\phi}$ to denote self-framed formulas.

$$\frac{\langle H, \rho, \Pi\rangle \vDash_I p(e_1, ..., e_n) \qquad \widetilde{\vdash}^{H,\rho,\Pi}_{\texttt{frmI}}\ e_1 \qquad ... \qquad \widetilde{\vdash}^{H,\rho,\Pi}_{\texttt{frmI}}\ e_n \qquad \widetilde{\vdash}^{H,\rho,\Pi'}_{\texttt{frmI}}\ \phi \qquad \Pi' = \Pi \cup \lfloor \mathsf{body}(p)(e_1, ..., e_n)\rfloor_{\mathsf{TotalFP}(\phi,H,\rho),H,\rho}}{\widetilde{\vdash}^{H,\rho,\Pi}_{\texttt{frmI}}\ (\texttt{unfolding}\ p(e_1, ..., e_n)\ \texttt{in}\ \phi)}\ \widetilde{\text{FRMUNFOLDING}}$$

Fig. 12. GVL_{RP}: Framing (selected rules)

Footprints. For the new definitions of framing and self-framing, we extend the notion of footprints to gradual formulas in an iso-recursive setting.

$$\begin{aligned} \lfloor \widehat{\phi} \rfloor_{\Pi,H,\rho} &= \lfloor \widehat{\phi} \rfloor_{H,\rho} \\ \lfloor ? * \phi \rfloor_{\Pi,H,\rho} &= \Pi \end{aligned}$$

To split footprints in our dynamic semantics rules, we extend the notion of footprints to gradual formulas in an equi-recursive setting. This definition also relies on a definition of erasure that handles potential imprecision in predicate bodies (presented in this section).

$$\begin{aligned} \lfloor \widehat{\phi} \rfloor_{\pi,H,\rho} &= \langle\langle\ \lfloor \widehat{\phi} \rfloor_{H,\rho}\ \rangle\rangle_{\pi,H} \\ \lfloor ? * \phi \rfloor_{\pi,H,\rho} &= \pi \end{aligned}$$

In presence of full imprecision, we must expect that all available permissions Π or π are represented by the formula and must hence be forwarded.

We also define a function that computes the *total footprint* of a formula, which contains both the explicit and implicit permissions required by the formula.

$\mathsf{TotalFP}(\cdot,\cdot,\cdot) : \text{FORMULA} \times \text{HEAP} \times \text{ENV} \to \text{PERMISSIONS}$

$$
\begin{aligned}
&\mathsf{TotalFP}(v, H, \rho) && = \emptyset \\
&\mathsf{TotalFP}(x, H, \rho) && = \emptyset \\
&\mathsf{TotalFP}(e_1 \odot e_2, H, \rho) && = \mathsf{TotalFP}(e_1, H, \rho) \cup \mathsf{TotalFP}(e_2, H, \rho) \\
&\mathsf{TotalFP}(e_1 \oplus e_2, H, \rho) && = \mathsf{TotalFP}(e_1, H, \rho) \cup \mathsf{TotalFP}(e_2, H, \rho) \\
&\mathsf{TotalFP}(e.f, H, \rho) && = \mathsf{TotalFP}(e, H, \rho) \cup \{\langle o, f\rangle | H, \rho \vdash e \Downarrow o\} \\
&\mathsf{TotalFP}(\mathbf{acc}(\texttt{e.f}), H, \rho) && = \mathsf{TotalFP}(e.f, H, \rho) \\
&\mathsf{TotalFP}(\phi_1 \wedge \phi_2, H, \rho) && = \mathsf{TotalFP}(\phi_1, H, \rho) \cup \mathsf{TotalFP}(\phi_2, H, \rho) \\
&\mathsf{TotalFP}(\phi_1 * \phi_2, H, \rho) && = \mathsf{TotalFP}(\phi_1, H, \rho) \cup \mathsf{TotalFP}(\phi_2, H, \rho) \\
&\mathsf{TotalFP}(p(e_1, ..., e_n), H, \rho) && = \mathsf{TotalFP}(e_1, H, \rho) \cup ... \cup \mathsf{TotalFP}(e_n, H, \rho) \cup \\
& && \quad \{\langle p, v_1, ..., v_n\rangle | H, \rho \vdash e_1 \Downarrow v_1, ..., H, \rho \vdash e_n \Downarrow v_n\} \\
&\mathsf{TotalFP}((\texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2), H, \rho) && = \begin{cases} \mathsf{TotalFP}(\phi_1, H, \rho) & \texttt{if } H, \rho \vdash e \Downarrow \texttt{true} \\ \mathsf{TotalFP}(\phi_2, H, \rho) & \texttt{if } H, \rho \vdash e \Downarrow \texttt{false} \\ \emptyset & \texttt{otherwise} \end{cases} \\
&\mathsf{TotalFP}((\texttt{unfolding } p(e_1, ..., e_n) \texttt{ in } \phi), H, \rho) && = \mathsf{TotalFP}(p(e_1, ..., e_n), H, \rho) \cup \mathsf{TotalFP}(\phi, H, \rho)
\end{aligned}
$$

Permissions to Dynamic Footprint. $\langle\langle\ \cdot\ \rangle\rangle_{\pi,H}$: PERMISSIONS × DYNFPRINT × HEAP ⇀ DYNFPRINT

$$
\begin{aligned}
&\langle\langle\ \Pi\ \rangle\rangle_{\pi,H} = \{\langle o, f\rangle \mid \langle o, f\rangle \in \Pi\} \cup \pi' \\
&\texttt{where } \pi' = \begin{cases} \pi & \texttt{if } \exists \langle p, v_1, ..., v_n\rangle \in \Pi. \exists \phi \in \textsc{Formula}.\mathsf{body}(p)(v_1, ..., v_n) = ? * \phi \\ \langle\langle\ \Pi'\ \rangle\rangle_{\pi,H} & \texttt{otherwise} \\ \quad \texttt{for } \Pi' = \cup_{\langle p, v_1, ..., v_n\rangle \in \Pi} \lfloor \mathsf{body}(p)(v_1, ..., v_n) \rfloor_{H,[]} & \end{cases}
\end{aligned}
$$

Note, $\langle\langle\ \Pi\ \rangle\rangle_{\pi,H}$ is a partial function, as it may not be well-defined if a predicate instance held in Π has an infinite unfolding and no nested imprecise predicate instances.

5.2 Naive Semantics

$\cdot \xrightarrow{\sim} \cdot \subseteq \textsc{State} \times (\textsc{State} \cup \{\textbf{error}\})$

A trivially correct but expensive strategy of adding runtime assertions to each execution step, checking whether the new state would be valid (preservation), right before actually transitioning into that state (progress).

Let $\langle H, \langle \rho_n, \pi_n, s_n \rangle \cdot ... \cdot \langle \rho_1, \pi_1, s_1 \rangle \cdot \mathsf{nil} \rangle, \langle H', \langle \rho'_n, \pi'_n, s'_n \rangle \cdot ... \cdot \langle \rho'_1, \pi'_1, s'_1 \rangle \cdot \mathsf{nil} \rangle \in \textsc{State}$

If $\langle H, \langle \rho_n, \pi_n, s_n \rangle \cdot ... \cdot \langle \rho_1, \pi_1, s_1 \rangle \cdot \mathsf{nil} \rangle \longrightarrow \langle H', \langle \rho'_n, \pi'_n, s'_n \rangle \cdot ... \cdot \langle \rho'_1, \pi'_1, s'_1 \rangle \cdot \mathsf{nil} \rangle$ (Fig. 13) holds, then

$\langle H, \langle \rho_n, \pi_n, s_n \rangle \cdot ... \cdot \langle \rho_1, \pi_1, s_1 \rangle \cdot \mathsf{nil} \rangle \xrightarrow{\sim}$

$$\begin{cases} \langle H', \langle \rho'_n, \pi'_n, s'_n \rangle \cdot ... \cdot \langle \rho'_1, \pi'_1, s'_1 \rangle \cdot \mathsf{nil} \rangle & \text{if } (\langle H', \rho'_n, \pi'_n \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_n, \mathsf{body} \rangle) \wedge ... \wedge (\langle H', \rho'_1, \pi'_1 \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_1, \mathsf{body} \rangle) \\ \quad \text{where } \widetilde{\phi}_n \cdot ... \cdot \widetilde{\phi}_1 \cdot \mathsf{nil} = \widetilde{\mathsf{sWLP}}(s_n \cdot ... \cdot s_1 \cdot \mathsf{nil}, \mathtt{true}) & \\ \textbf{error} & \text{otherwise} \end{cases}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot \mathit{nil} \rangle \quad \texttt{final}}\ \textsc{SsSkip}$$

$$\frac{\langle H, \langle \rho, \pi, s_1 \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho', \pi', s_1' \rangle \cdot S \rangle}{\langle H, \langle \rho, \pi, s_1 \texttt{;}\ s_2 \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho', \pi', s_1' \texttt{;}\ s_2 \rangle \cdot S \rangle}\ \textsc{SsSeqCong}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{skip;}\ s_2 \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, s_2 \rangle \cdot S \rangle}\ \textsc{SsSeq}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{T x} \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsDeclare}$$

$$\frac{\langle H, \rho, \pi \rangle \mathrel{\widetilde{\models}} \langle \texttt{?} * \phi, \mathsf{body} \rangle}{\langle H, \langle \rho, \pi, \texttt{assert}\ \phi \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAssert}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \mathbf{acc}(\texttt{x.f}) \qquad H, \rho \vdash y \Downarrow v \qquad H' = H[o \mapsto [f \mapsto v]]}{\langle H, \langle \rho, \pi, x\texttt{.}f\ \texttt{:=}\ y \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsFAssign}$$

$$\frac{\langle H, \rho, \pi \rangle \vDash_E \mathsf{acc}(e) \qquad H, \rho \vdash e \Downarrow v \qquad \rho' = \rho[x \mapsto v]}{\langle H, \langle \rho, \pi, x\ \texttt{:=}\ e \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho', \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAssign}$$

$$\frac{o \notin \mathsf{dom}(H) \qquad \mathsf{fields}(C) = \overline{T_i\ f_i\texttt{;}} \qquad H' = H[o \mapsto [f_i \mapsto \mathsf{defaultValue}(T_i)]]}{\langle H, \langle \rho, \pi, x\ \texttt{:=}\ \texttt{new}\ C \rangle \cdot S \rangle \longrightarrow \langle H', \langle \rho[x \mapsto o], \pi \cup \overline{\langle o, f_i \rangle}, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsAlloc}$$

$$\frac{\begin{array}{c}\mathsf{method}(m) = T_r\ m(\overline{T\ x'})\ \texttt{requires}\ \widetilde{\phi}_p\ \texttt{ensures}\ \widetilde{\phi}_q\ \texttt{\{}\ r\ \texttt{\}} \\ H, \rho \vdash z \Downarrow o \qquad \overline{H, \rho \vdash x \Downarrow v} \qquad \rho' = [\texttt{this} \mapsto o, \overline{x' \mapsto v}, \overline{\texttt{old}(x') \mapsto v}] \\ \pi' = \lfloor \widetilde{\phi}_p \rfloor_{\pi, H, \rho'} \qquad \langle H, \rho', \pi' \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_p, \mathsf{body} \rangle\end{array}}{\langle H, \langle \rho, \pi, y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho', \pi', r \rangle \cdot \langle \rho, \pi \backslash \pi', y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle}\ \textsc{SsCall}$$

$$\frac{\mathsf{post}(m) = \widetilde{\phi}_q \qquad \langle H, \rho', \pi' \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_q, \mathsf{body} \rangle \qquad \rho'' = \rho[y \mapsto \rho'(\texttt{result})]}{\langle H, \langle \rho', \pi', \texttt{skip} \rangle \cdot \langle \rho, \pi, y\ \texttt{:=}\ z\texttt{.}m(\overline{x})\texttt{;}\ s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho'', \pi \cup \pi', s \rangle \cdot S \rangle}\ \textsc{SsCallFinish}$$

$$\frac{\langle H, \rho, \pi \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_i, \mathsf{body} \rangle \qquad H, \rho \vdash e \Downarrow \texttt{false}}{\langle H, \langle \rho, \pi, \texttt{while (e) inv}\ \widetilde{\phi}_\texttt{i}\ \texttt{\{ r \}; } s \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, s \rangle \cdot S \rangle}\ \textsc{SsWhileFalse}$$

$$\frac{\begin{array}{c}\langle H, \rho, \pi \rangle \mathrel{\widetilde{\models}} \langle \widetilde{\phi}_i, \mathsf{body} \rangle \qquad H, \rho \vdash e \Downarrow \texttt{true} \\ \pi' = \lfloor \widetilde{\phi}_i \rfloor_{\pi, H, \rho}\end{array}}{\begin{array}{c}\langle H, \langle \rho, \pi, \texttt{while (e) inv}\ \widetilde{\phi}_\texttt{i}\ \texttt{\{ r \}; } s \rangle \cdot S \rangle \\ \longrightarrow \\ \langle H, \langle \rho, \pi', r \rangle \cdot \langle \rho, \pi \backslash \pi', \texttt{while (e) inv}\ \widetilde{\phi}_\texttt{i}\ \texttt{\{ r \}; } s \rangle \cdot S \rangle\end{array}}\ \textsc{SsWhileTrue}$$

$$\frac{}{\begin{array}{c}\langle H, \langle \rho', \pi', \texttt{skip} \rangle \cdot \langle \rho, \pi, \texttt{while (e) inv}\ \widetilde{\phi}_\texttt{i}\ \texttt{\{ r \}; } s \rangle \cdot S \rangle \\ \longrightarrow \\ \langle H, \langle \rho', \pi \cup \pi', \texttt{while (e) inv}\ \widetilde{\phi}_\texttt{i}\ \texttt{\{ r \}; } s \rangle \cdot S \rangle\end{array}}\ \textsc{SsWhileFinish}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{fold}\ p(e_1, \ldots, e_n) \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsFold}$$

$$\frac{}{\langle H, \langle \rho, \pi, \texttt{unfold}\ p(e_1, \ldots, e_n) \rangle \cdot S \rangle \longrightarrow \langle H, \langle \rho, \pi, \texttt{skip} \rangle \cdot S \rangle}\ \textsc{SsUnfold}$$

Fig. 13. $\mathrm{GVL}_{\mathrm{RP}}$: Small-step semantics adjusted from Fig. 5 for gradual formulas

5.3 Residual Check Semantics

6 Properties of GVL_{RP}

GVL_{IDF} is a sound, conservative extension of SVL_{RP}

6.1 Soundness

Definition 15 (Valid state). *We call the state* $\langle H, \langle \rho_n, \pi_n, s_n \rangle \cdot ... \cdot \langle \rho_1, \pi_1, s_1 \rangle \cdot \mathsf{nil} \rangle \in$ STATE valid *if* $\langle H, \rho_i, \pi_i \rangle \mathrel{\widetilde{\models}} \widetilde{\mathsf{sWLP}_i}(s_n \cdot ... \cdot s_1 \cdot \mathsf{nil}, \mathtt{true})$ *for all* $1 \leq i \leq n$.

Proposition 3 (SVL_{RP} Progress). *If* $\psi \in$ STATE *is a valid state and* $\psi \notin \{\langle H, \langle \rho, \pi, skip \rangle \cdot nil \rangle \mid H \in \text{HEAP}, \rho \in \text{ENV}, \pi \in \text{DYNFPRINT}\}$ *then* $\psi \mathrel{\widetilde{\longrightarrow}} \psi'$ *for some* $\psi' \in$ STATE *or* $\psi \mathrel{\widetilde{\longrightarrow}} \textbf{error}$.

Proposition 4 (SVL_{RP} Preservation). *If* ψ *is a valid state and* $\psi \mathrel{\widetilde{\longrightarrow}} \psi'$ *for some* $\psi' \in$ STATE *then* ψ' *is a valid state.*

6.2 Gradual Guarantee

It satisfies the gradual guarantee, with slight adjustments to both the static and dynamic gradual guarantees.

Due to potentially imprecise predicate definitions and loop invariants, we must adjust GVL's program precision definition. We say a program p_1 is more precise than program p_2 ($p_1 \sqsubseteq p_2$) if p_1 and p_2 are equivalent except in terms of contracts, loop invariants, and/or predicate definitions and if p_1's contracts, loop invariants, and predicate definitions are more precise than p_2's corresponding contracts, loop invariants, and predicate definitions. A contract `requires` $\widetilde{\phi}^1_p$ `ensures` $\widetilde{\phi}^1_q$ is more precise than contract `requires` $\widetilde{\phi}^2_p$ `ensures` $\widetilde{\phi}^2_q$ if $\widetilde{\phi}^1_p \sqsubseteq \widetilde{\phi}^2_p$ and $\widetilde{\phi}^1_q \sqsubseteq \widetilde{\phi}^2_q$. Similarly, a loop invariant (predicate definition) $\widetilde{\phi}^1_i$ is more precise than loop invariant (predicate definition) $\widetilde{\phi}^2_i$ if $\widetilde{\phi}^1_i \sqsubseteq \widetilde{\phi}^2_i$.

The static gradual guarantee can now be stated (as in GVL) using this new precision definition.

Proposition 5 (Static gradual guarantee of verification).
Let $p_1, p_2 \in$ PROGRAM *such that* $p_1 \sqsubseteq p_2$. *If* p_1 *is valid then* p_2 *is valid.*

The fact that footprint tracking and splitting is influenced by increasing imprecision means that we can no longer talk about *equal* program states ψ_1 and ψ_2 as we did in GVL. We hence first refine state equality to an asymmetric *state precision* relation $\lesssim$:

Definition 16 (State Precision). *Let* $\psi_1, \psi_2 \in$ STATE. *Then* ψ_1 *is more precise than* ψ_2, *written* $\psi_1 \lesssim \psi_2$, *if and only if all of the following applies:*
a) ψ_1 *and* ψ_2 *have stacks of size n and identical heaps.*
b) ψ_1 *and* ψ_2 *have stacks of variable environments and stacks of statements that are identical.*

c) *Let* $\pi^1_{1..n}$ *and* $\pi^2_{1..n}$ *be the stack of footprints of* ψ_1 *and* ψ_2*, respectively. Then the following holds for* $1 \leq m \leq n$*:*

$$\bigcup_{i=m}^{n} \pi^1_i \subseteq \bigcup_{i=m}^{n} \pi^2_i$$

The new dynamic gradual guarantee reflects the fact that imprecision potentially results in larger parts of footprints being passed up the stack, and TODO ▶*confirm this after finishing residual checks*◀ hence residual checks, which are evaluated against the top most stack frame, are more likely to succeed in the less precise program. Therefore, Prop. 6 establishes that successful progress from a state ψ_1 for a program p_1 also implies successful progress for a less precise state ψ_2 and less precise program p_2 (using the new program precision definition).

Proposition 6 (Dynamic gradual guarantee of verification).
Let $p_1, p_2 \in$ PROGRAM *such that* $p_1 \sqsubseteq p_2$*, and* $\psi_1, \psi_2 \in$ STATE *such that* $\psi_1 \lesssim \psi_2$*.*
If $\psi_1 \xrightarrow{\sim}_{p_1} \psi'_1$ *then* $\psi_2 \xrightarrow{\sim}_{p_2} \psi'_2$*, with* $\psi'_1 \lesssim \psi'_2$*.*

Additional Program Imprecision Our formal system allows additional program imprecision in more specific cases. Fold and unfold statements may be removed from programs that do not rely on them to adhere to the static gradual guarantee (ie. if the programs rely on them for validity). Since GVL_{RP}'s equi-recursive styled dynamic semantics essentially ignores fold and unfold statements, the spirit of the dynamic gradual guarantee is preserved in these cases. TODO ▶*maybe state variants on the dynamic and static gradual guarantees, which include these cases of imprecision*◀

References

1. A. J. Summers and S. Drossopoulou. A formal semantics for isorecursive and equirecursive state abstractions. In *European Conference on Object-Oriented Programming*, pages 129–153. Springer, 2013.

A Appendix

$$\widetilde{\mathsf{WLP}}(s_1\,;\ s_2, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \widetilde{\mathsf{WLP}}(s_1, \widetilde{\mathsf{WLP}}(s_2, \langle \widetilde{\phi}, \widetilde{body}\rangle))$$

$$\widetilde{\mathsf{WLP}}(\texttt{if (e) then \{s}_1\texttt{\} else \{s}_2\texttt{\}}, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \langle \texttt{(if } e \texttt{ then } \widehat{\phi}_1 \texttt{ else } \widehat{\phi}_2), body\rangle$$
$$\mid \langle \widehat{\phi}_1, body_1\rangle \in \gamma(\widetilde{\mathsf{WLP}}(s_1, \langle \widetilde{\phi}, \widetilde{body}\rangle)),\ \langle \widehat{\phi}_2, body_2\rangle \in \gamma(\widetilde{\mathsf{WLP}}(s_2, \langle \widetilde{\phi}, \widetilde{body}\rangle)),$$
$$body \in \gamma(\widetilde{body})\ \})$$

$$\widetilde{\mathsf{WLP}}(y := z.m(\overline{x}), \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \langle \max_{\Rightarrow}\{\ \widehat{\phi}' \mid y \notin \mathsf{FV}(\widehat{\phi}') \quad \wedge$$
$$\widehat{\phi}' \Rightarrow \texttt{(}z \neq \texttt{null)} * \widehat{\phi}_p[z/\texttt{this}, \overline{x/\texttt{mparam}(m)}] \quad \wedge$$
$$\widehat{\phi}' * \widehat{\phi}_q[z/\texttt{this}, \overline{x/\texttt{old(mparam}(m)\texttt{)}}, y/\texttt{result}] \Rightarrow \widehat{\phi}\ \}, body\rangle$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_p \in \gamma(\mathsf{mpre}(m)),\ \widehat{\phi}_q \in \gamma(\mathsf{mpost}(m)),\ body \in \gamma(\widetilde{body})\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{while (e) inv } \widetilde{\phi}_i\ \texttt{\{ s \}}, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \langle \max_{\Rightarrow}\{\ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi}_i \quad \wedge \quad \widehat{\phi}' * \texttt{(}e \texttt{ = false)} \Rightarrow \widehat{\phi}\ \}, body\rangle$$
$$\mid \widehat{\phi} \in \gamma(\widetilde{\phi}),\ \widehat{\phi}_i \in \gamma(\widetilde{\phi}_i),\ body \in \gamma(\widetilde{body})\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{fold } p\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)}, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \langle body(p)(e_1, ..., e_n) * \max_{\Rightarrow}\{\ \widehat{\phi}' \mid \widehat{\phi}' * p\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)} \Rightarrow \widehat{\phi} \quad \wedge$$
$$\widehat{\phi}' * p\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)} \in \textsc{SatFormula}\ \}, body\rangle$$
$$\mid \langle \widehat{\phi}, body\rangle \in \gamma(\langle \widetilde{\phi}, \widetilde{body}\rangle)\ \})$$

$$\widetilde{\mathsf{WLP}}(\texttt{unfold } p\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)}, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \langle p\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)} * \max_{\Rightarrow}\{\ \widehat{\phi}' \mid \widehat{\phi}' * body(p)(e_1, ..., e_n) \Rightarrow \widehat{\phi} \quad \wedge$$
$$\widehat{\phi}' * body(p)(e_1, ..., e_n) \in \textsc{SatFormula}\ \}, body\rangle$$
$$\mid \langle \widehat{\phi}, body\rangle \in \gamma(\langle \widetilde{\phi}, \widetilde{body}\rangle)\ \})$$

$$\widetilde{\mathsf{WLP}}(s, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \alpha(\{\ \mathsf{WLP}(s, \langle \widehat{\phi}, body\rangle) \mid \langle \widehat{\phi}, body\rangle \in \gamma(\langle \widetilde{\phi}, \widetilde{body}\rangle)\ \}) \quad \text{otherwise}$$

Fig. 14. $\mathrm{GVL}_{\mathrm{IDF}}$: Weakest preconditions alternative, equivalent to Figure 10 when body provided as an argument.

The $\widetilde{\mathsf{WLP}}$ for *if statements* is defined correctly, since Lemma 3 is provable by induction on the syntax of program statements. Lemma 4 is also provable by induction on the syntax of program statements, leading to an equivalence between definitions in Figures 10 and 14.

Lemma 3 ($\widetilde{\mathsf{WLP}}$ (Fig. 14) preserves $\widetilde{body}$).
Let $s \in \textsc{Stmt}$, $\langle \widetilde{\phi}, \widetilde{body}\rangle \in \widetilde{\textsc{Formula}} \times (\textsc{PredName} \to \textsc{Expr}^* \to \widetilde{\textsc{Formula}})$,

then $\widetilde{\mathsf{WLP}}(s, \langle \widetilde{\phi}, \widetilde{body}\rangle) = \langle \widetilde{\phi}', \widetilde{body}\rangle$ *for some* $\widetilde{\phi}' \in \widetilde{\textsc{Formula}}$.

TODO ▶*proof*◀

Lemma 4 **($\widetilde{\mathsf{WLP}}$ equivalent definitions).**
Let $s \in \textsc{Stmt}$, $\widetilde{\phi} \in \widetilde{\textsc{Formula}}$,

then $\langle \widetilde{\mathsf{WLP}}(s, \widetilde{\phi}), \mathsf{body} \rangle = \widetilde{\mathsf{WLP}}(s, \langle \widetilde{\phi}, \mathsf{body} \rangle)$.

TODO ▶*proof*◀