# SVL with Recursive Predicates

Henry Blanchette

## Contents

# 1 Grammar

$$
\begin{array}{rcl}
x, y, z & \in & \mathit{VAR} \\
v & \in & \mathit{VAL} \\
e & \in & \mathit{EXPR} \\
s & \in & \mathit{STMT} \\
o & \in & \mathit{LOC} \\
f & \in & \mathit{FIELDNAME} \\
m & \in & \mathit{METHODNAME} \\
C, D & \in & \mathit{CLASSNAME} \\
\alpha & \in & \mathit{PREDNAME} \\
P & ::= & \overline{\mathit{cls}}\ s \\
\mathit{cls} & ::= & \texttt{class}\ C\ \texttt{extends}\ D\ \{\overline{\mathit{field}}\ \overline{\mathit{pred}}\ \overline{\mathit{method}}\} \\
\mathit{field} & ::= & T\ f; \\
\mathit{pred} & ::= & \texttt{predicate}\ \alpha_C(\overline{T\ x}) = \widetilde{\phi} \\
T & ::= & \texttt{int} \mid \texttt{bool} \mid C \mid \top \\
\mathit{method} & ::= & T\ m(\overline{T\ x})\ \texttt{dynamically}\ \mathit{contract}\ \texttt{statically}\ \mathit{contract}\ \{s\} \\
\mathit{contract} & ::= & \texttt{requires}\ \widetilde{\phi}\ \texttt{ensures}\ \widetilde{\phi} \\
\oplus & ::= & + \mid - \mid * \mid \backslash \mid \&\& \mid \ || \\
\odot & ::= & \neq \mid = \mid < \mid > \mid \leq \mid \geq \\
s & ::= & \texttt{skip} \mid s_1\ ;\ s_2 \mid T\ x \mid x := e \mid \texttt{if}\ (e)\ \{s_1\}\ \texttt{else}\ \{s_2\} \\
 & & \mid \texttt{while}\ (e)\ \texttt{invariant}\ \widetilde{\phi}\ \{s\} \mid x.f := y \mid x := \texttt{new}\ C \mid y := z.m(\overline{x}) \\
 & & \mid y := z.m_C(\overline{x}) \mid \texttt{assert}\ \phi \mid \texttt{release}\ \phi \mid \texttt{hold}\ \phi\ \{s\} \mid \texttt{fold}\ A \mid \texttt{unfold}\ A \\
e & ::= & v \mid x \mid e \oplus e \mid e \odot e \mid e.f \\
x & ::= & \texttt{result} \mid \mathit{id} \mid \texttt{old}(\mathit{id}) \mid \texttt{this} \\
v & ::= & n \mid o \mid \texttt{null} \mid \texttt{true} \mid \texttt{false} \\
A & ::= & \alpha(\overline{e}) \mid \alpha_C(\overline{e}) \\
\circledast & ::= & \wedge \mid * \\
\phi & ::= & e \mid A \mid \texttt{acc}(e.f) \mid \phi \circledast \phi \mid (\texttt{if}\ e\ \texttt{then}\ \phi\ \texttt{else}\ \phi) \mid (\texttt{unfolding}\ A\ \texttt{in}\ \phi) \\
\widetilde{\phi} & ::= & \phi \mid ? * \phi
\end{array}
$$

## 2 Well-formedness

# 3 Aliasing

## 3.1 Definitions

An **object variable** is one of the following:

- a class instance variable i.e. a variable $v$ such that $v : C$ for some class $C$,
- a class instance field reference i.e. a field reference $e.f$ where $e.f : C$ for some class $C$,
- `null` as a value such that $\texttt{null} : C$ for some class $C$.

Let $\mathcal{O}$ be a set of object variables. An $O \subset \mathcal{O}$ **aliases** if and only if each $o \in O$ refers to the same memory in the heap as each other, written propositionally as

$$\forall o, o' \in O : o = o' \iff \mathsf{aliases}(O)$$

While it is possible to keep track of negated aliasings (of the form $\sim \mathsf{aliases}\,\{o_\alpha\}$), this will not be needed for either aliasing tree construction or self-framing desicions. So, it will not be tracked i.e. $x \neq y$ does not contribute anything to an aliasing context.

## 3.2 Aliasing Context

Let $\phi$ be a formula. The **aliasing context** $\mathcal{A}$ of $\phi$ is a tree of set of aliasing proposition about aliasing of object variables that appear in $\phi$. $\mathcal{A}$ needs to be a tree because the conditional sub-formulas that may appear in $\phi$ allow for branching aliasing contexts not expressible flatly at the top level. Each node in the tree corresponds to a set of aliasing propositions, and each branch refers to a branch of a unique conditional in $\phi$. The parts of the tree are labeled in such a way that modularly allows a specified sub-formula of $\phi$ to be matched to the unique aliasing sub-context that corresponds to it. For example, consider the following formula:

$$\begin{aligned}
\phi := {} & (o_1 = o_2) * \\
& (\texttt{if } (b_1) \\
& \quad \texttt{then } ( \\
& \qquad (o_1 \neq o_3) * \\
& \qquad (\texttt{if } (b_2) \\
& \qquad\quad \texttt{then } (o_1 = o_4) \\
& \qquad\quad \texttt{else } (b_3))) \\
& \quad \texttt{else } (o_1 = o_3)) * \\
& (o_1 = o_4)
\end{aligned}$$

where $b_1, b_2$ are arbitrary boolean expressions that do not assert aliasing propositions. $\phi$ has a formula-structure represented by the tree in figure 3.2. The formula-structure tree for

Figure 1: Formula structure tree for $\phi$.

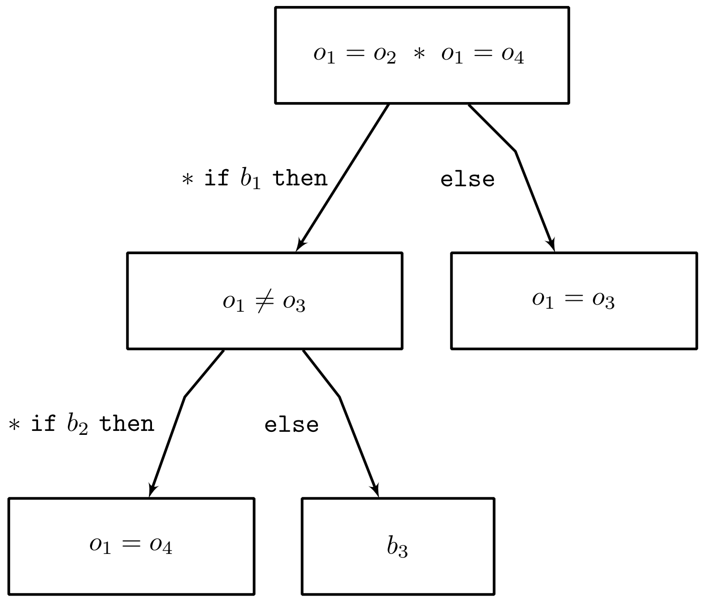
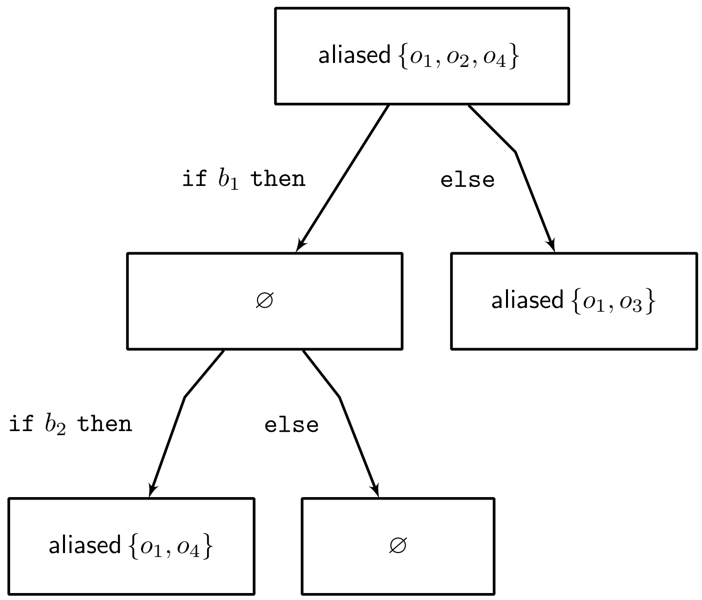



Figure 2: $\mathcal{A}(\phi)$, the aliasing context tree for $\phi$.



$\phi$ corresponds node-for-node and edge-for-edge to the aliasing context tree in figure 3.2.

More generally, for $\phi$ a formula and $\phi'$ a sub-formula of $\phi$, write $\mathcal{A}_\phi(\phi')$ as the **total aliasing context** of $\phi'$ which includes aliasing propositions inherited from its ancestors in the aliasing context tree of $\phi$. These aliasing contexts are combined via $\sqcup$ which will be defined in the next section. For example, the total aliasing context at the sub-formula $(o_1 = o_4)$ of $\phi$ is:

$$\mathcal{A}_\phi(o_1 = o_4) := \{\mathsf{aliased}\,\{o_1, o_2, o_4\}\}$$

along with the fact that it has no child branches. Usually $\mathcal{A}_{\phi_{\mathsf{root}}}(\phi')$ is abbreviated to $\mathcal{A}(\phi')$ when the top level formula $\phi$ is implicit and $\phi'$ is a sub-formula of $\phi_{\mathsf{root}}$.

An aliasing context $\mathcal{A}$ may entail $\mathsf{aliased}(O)$ for some $O \subset \mathcal{O}$. Since $\mathcal{A}$ is efficiently represented as a set of propositions about sets, it may be the case that $\mathsf{aliased}(O) \notin \mathcal{A}$ yet still the previous judgement holds. For example, this is true when $\exists O' \subset \mathcal{O}$ such that $O \subset O'$ and $\mathsf{aliased}(O') \in \mathcal{A}$. So, the explicit definition for making this judgement is as follows:

$$\mathcal{A} \vdash \mathsf{aliased}(O) \iff \exists O' \subset \mathcal{O} : (O \subset O') \ \wedge \ (\mathsf{aliased}(O') \in \mathcal{A})$$

The notations $\mathsf{aliased}(O) \in \mathcal{A}$ is a little misleading because $\mathcal{A}$ is in fact a tree and not just a set. To be explicit, $\mathsf{aliased}(O) \in \mathcal{A}$ is defined to be set membership of the set of aliasing propositions in the total aliasing context at $\mathcal{A}$.

## 3.3 Constructing an Aliasing Context

An aliasing context of a formula $\phi$ is a tree, where nodes represent local aliasing contexts and branches represent the branches of conditional sub-formulas nested in $\phi$. So, an aliasing context is defined structurally as

$$\mathcal{A} \ ::= \langle A, \{e_\alpha : \mathcal{A}_\alpha\}\rangle$$

where $A$ is a set of propositions about aliasing and the $e_\alpha : \mathcal{A}_\alpha$ are the nesting aliasing contexts that correspond to the $\mathsf{then}$ and $\mathsf{else}$ branches of conditionals directly nested in $\phi$, the $e_\alpha$ indicating the condition for branching to $\mathcal{A}_\alpha$. For the purposes of look-up, each $\mathcal{A}$ is labeled by the sub-formula it corresponds to.

Given a root formula $\phi_{\mathsf{root}}$, the aliasing context of $\phi_{\mathsf{root}}$ is written $\mathcal{A}(\phi_{\mathsf{root}})$. With the root invariant, the following recursive algorithm constructs $\mathcal{A}(\phi)$ for any sub-formula of $\phi_{\mathsf{root}}$

(including $\mathcal{A}(\phi_{\mathsf{root}})$).

$$
\begin{array}{lcl}
\mathcal{A}(\phi) & := & \mathsf{match}\ \phi\ \mathsf{with} \\
& & \begin{array}{lcl}
v & \mapsto & \langle \varnothing, \varnothing \rangle \\
x & \mapsto & \langle \varnothing, \varnothing \rangle \\
e_1\ \&\&\ e_2 & \mapsto & \mathcal{A}(e_1) \sqcup \mathcal{A}(e_2) \\
e_1 \mid\mid e_2 & \mapsto & \mathcal{A}(e_1) \sqcap \mathcal{A}(e_2) \\
e_1 \oplus e_2 & \mapsto & \langle \varnothing, \varnothing \rangle \\
o_1 = o_2 & \mapsto & \langle \{\mathsf{aliases}\,\{o_1, o_2\}\}\,, \varnothing \rangle \\
e_1 \odot e_2 & \mapsto & \langle \varnothing, \varnothing \rangle \\
e.f & \mapsto & \langle \varnothing, \varnothing \rangle \\
\mathtt{acc}(e.f) & \mapsto & \langle \varnothing, \varnothing \rangle \\
\phi_1 * \phi_2 & \mapsto & \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2) \\
\phi_1 \wedge \phi_2 & \mapsto & \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2) \\
\alpha_C(e_1, \ldots, e_k) & \mapsto & \langle \varnothing, \varnothing \rangle \\
\mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2 & \mapsto & \langle \varnothing, \{\mathcal{A}(e) \sqcup \mathcal{A}(\phi_1), (\mathcal{A}(\sim e)) \sqcup \mathcal{A}(\phi_2)\} \rangle \\
\mathtt{unfolding}\ \alpha_C(e_1, \ldots, e_k)\ \mathtt{in}\ \phi' & \mapsto & \mathcal{A}(\phi')
\end{array}
\end{array}
$$

Note that the $\mathcal{A}(\sim e)$ result of the rule for $\mathcal{A}(\mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2)$ means to negate the boolean expression of $e$ and then take the aliasing context of that. As examples,

$$
\begin{aligned}
\mathcal{A}(\sim (x = y)) &= \mathcal{A}(x \neq y) = \langle \varnothing, \varnothing \rangle \\
\mathcal{A}(\sim (x \neq y)) &= \mathcal{A}(x = y) = \langle \{\mathsf{aliased}\,\{x, y\}\}\,, \varnothing \rangle
\end{aligned}
$$

Context union, $\sqcup$, and context intersection, $\sqcap$, are operations that combine aliasing contexts and are defined below.

$$
\begin{aligned}
&\langle A_1, \{e_\alpha : \mathcal{A}_\alpha\} \rangle \sqcup \langle A_2, \{e_\beta : \mathcal{A}_\beta\} \rangle := \\
&\qquad \langle \{\mathsf{aliased}\,\{o' \mid \forall o' : (A_1 \vdash \mathsf{aliased}\,\{o, o'\}) \ \vee\ (A_2 \vdash \mathsf{aliased}\,\{o, o'\})\} \mid \forall o\}\,, \\
&\qquad \{e_\alpha : \mathcal{A}_\alpha\} \cup \{e_\beta : \mathcal{A}_\beta\} \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\langle A_1, \{e_\alpha : \mathcal{A}_\alpha\} \rangle \sqcap \langle A_2, \{e_\beta : \mathcal{A}_\beta\} \rangle := \\
&\qquad \langle \{\mathsf{aliased}\,\{o' \mid \forall o' : (A_1 \vdash \mathsf{aliased}\,\{o, o'\}) \ \wedge\ (A_2 \vdash \mathsf{aliased}\,\{o, o'\})\} \mid \forall o\}\,, \\
&\qquad \{e_\alpha : \mathcal{A}_\alpha\} \cap \{e_\beta : \mathcal{A}_\beta\} \rangle
\end{aligned}
$$

# 4 Framing

## 4.1 Definitions

For framing, a formula is considered inside a **permission context**, a set of permissions, where a **permission** $\pi$ is to do one of the following:

- to reference $e.f$, written $\mathsf{accessed}(e.f)$.
- to assume $\alpha_C(\overline{e})$, written $\mathsf{assumed}(\alpha_C(\overline{e}))$. This allows the a single unrolling of $\alpha_C(\overline{e})$. Explicitly, an instance of $\mathsf{assumed}(\alpha_C(\overline{e}))$ in a set of permissions $\Pi$ may be expanded into $\Pi \cup \mathsf{granted}(\dots)$ where $\dots$ is replaced with a single unrolling of the body of $\alpha_C(\overline{e})$ with the arguments substituted appropriately[1].

Let $\phi$ be a formula. $\phi$ may **require** a permission $\pi$. For example, the formula $e.f = 1$ requires $\mathsf{accessed}(e.f)$, because it references $e.f$. The set of all permissions that $\phi$ requires is called the **requirements** of $\phi$. $\phi$ may also **grant** a permission $\pi$. For example, the formula $\mathsf{acc}(e.f)$ grants the permission $\mathsf{accessed}(e.f)$.

Altogether, $\phi$ is **framed** by a set of permissions $\Pi$ if all permissions required by $\phi$ are either in $\Pi$ or granted by $\phi$. The proposition that $\Pi$ frames $\phi$ is written

$$\Pi \vDash_I \phi$$

Of course, $\phi$ may grant some of the permissions it requires but not all. The set of permissions that $\phi$ requires but does not grant is called the **footprint** of $\phi$. The footprint of $\phi$ is written

$$\lfloor \phi \rfloor$$

Finally, a $\phi$ is called **self-framing** if and only if for any set of permissions $\Pi$, $\Pi \vDash_I \phi$. The proposition that $\phi$ is self-framing is written

$$\vdash_{\mathsf{frm}I} \phi$$

Note that $\vdash_{\mathsf{frm}I} \phi \iff \varnothing \vDash_I \phi$, in other words $\phi$ is self-framing if and only if it grants all of the permissions it requires. Or in other words still, $\lfloor \phi \rfloor = \varnothing$.

[1]As demonstrated by this description, $\mathsf{assumed}$ predicates are really just a useful shorthand and not a fundamentally new type of permission. The only kind fundamental kind of permission is $\mathsf{accessed}$.

## 4.2 Deciding Framing

Deciding $\Pi \vDash_I \phi$ must take into account the requirements, granteds, and aliases contained in $\Pi$ and the sub-formulas of $\phi$. The following recursive algorithm decides $\Pi \vDash_I \phi_{root}$, where $\mathcal{A}$ is implicitly assumed to be the top-level aliasing context (where the top-level in this context is the level that $\phi_{root}$ exists at in the program).

$$
\begin{array}{rcll}
\Pi \vDash_I \phi & \iff & \textsf{match } \phi \textsf{ with} & \\
& & v & \mapsto \top \\
& & x & \mapsto \top \\
& & e_1 \oplus e_2 & \mapsto \Pi \vDash_I e_1, e_2 \\
& & e_1 \odot e_2 & \mapsto \Pi \vDash_I e_1, e_2 \\
& & e.f & \mapsto (\Pi \vDash_I e) \ \wedge \ (\Pi \vdash \textsf{accessed}_\phi(e.f)) \\
& & \texttt{acc}(e.f) & \mapsto (\Pi \vDash_I e) \\
& & \phi_1 \circledast \phi_2 & \mapsto (\Pi \cup \textsf{granted}(\phi_2) \vDash_I \phi_1) \ \wedge \\
& & & \quad\ (\Pi \cup \textsf{granted}(\phi_1) \vDash_I \phi_2) \\
& & \alpha_C(e_1, \ldots, e_k) & \mapsto \Pi \vDash_I e_1, \ldots, e_2 \\
& & \texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto \Pi \vDash_I e, \phi_1, \phi_2 \\
& & \texttt{unfolding } \alpha_C(\overline{e}) \texttt{ in } \phi' & \mapsto (\Pi \vdash \textsf{assumed}_\phi(\alpha_C(\overline{e}))) \ \wedge \ (\Pi \vDash_I \phi') \\
\textsf{granted}(\phi) & := & \textsf{match } \phi \textsf{ with} & \\
& & e & \mapsto \varnothing \\
& & \texttt{acc}(e.f) & \mapsto \{\textsf{accessed}(e.f)\} \\
& & \phi_1 \circledast \phi_2 & \mapsto \textsf{granted}(\phi_1) \cup \textsf{granted}(\phi_2) \\
& & \alpha_C(\overline{e}) & \mapsto \{\textsf{assumed}(\alpha_C(\overline{e}))\} \\
& & \texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto \textsf{granted}(\phi_1) \cap \textsf{granted}(\phi_2) \\
& & \texttt{unfolding } \alpha_C(\overline{e}) \texttt{ in } \phi' & \mapsto \textsf{granted}(\phi')
\end{array}
$$

Where $\textsf{accessed}_\phi$ and $\textsf{assumed}_\phi$ indicate the respective propositions considered within the total alias context (including inherited aliasing contexts). More explicitly,

$$
\begin{aligned}
\Pi \vdash \textsf{accessed}_\phi(o.f) &\iff \exists o' \in O : (\mathcal{A}(\phi) \vdash \textsf{aliased}\,\{o, o'\}) \ \wedge \ (\textsf{accessed}(o'.f) \in \Pi) \\
\Pi \vdash \textsf{assumed}_\phi(\alpha_C(e_1, \ldots, e_k)) &\iff (\forall i : (e_i = e'_i) \ \vee \\
&\qquad\quad (\exists o, o' : ((o, o') = (e_i, e'_i)) \wedge \mathcal{A}(\phi) \vdash \textsf{aliased}\,\{o, o'\})) \\
&\quad \wedge \ (\textsf{assumed}(\alpha_C(e'_1, \ldots, e'_k)) \in \Pi)
\end{aligned}
$$

## 4.3 Examples

In the following examples, assume that the considered formulas are well-formed.

### Example 1

Define

$$\phi_{\mathsf{root}} := x = y * \mathtt{acc}(x.f) * \mathtt{acc}(y.f).$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \{\mathsf{aliased}\,\{x, y\}\}\,, \varnothing \rangle.$$

And so,

$$\begin{aligned}
\vdash_{\mathsf{frm}I} \phi_{\mathsf{root}} &\iff \varnothing \vDash_I \phi_{\mathsf{root}} \\
&\iff \varnothing \vDash_I x = y * \mathtt{acc}(x.f) * \mathtt{acc}(y.f) \\
&\iff (\mathsf{granted}(\mathtt{acc}(x.f) * \mathtt{acc}(y.f)) \vDash_I x = y) \wedge \\
&\phantom{\iff}\ (\mathsf{granted}(x = y * \mathtt{acc}(y.f)) \vDash_I \mathtt{acc}(x.f)) \wedge \\
&\phantom{\iff}\ (\mathsf{granted}(x = y * \mathtt{acc}(x.f)) \vDash_I \mathtt{acc}(y.f)) \\
&\iff \top \wedge \\
&\phantom{\iff}\ (\mathsf{granted}(x = y * \mathtt{acc}(y.f)) \vDash_I x) \wedge \\
&\phantom{\iff}\ (\mathsf{granted}(x = y * \mathtt{acc}(x.f)) \vDash_I y) \\
&\iff \top \wedge \top \wedge \top \\
&\iff \top
\end{aligned}$$

**Example 2**

Define

$$\phi_{\mathsf{root}} := \mathtt{acc}(x.f) \ * \ (\mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f))$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \varnothing, \{b : \mathcal{A}(x.f = 1), \sim b : \mathcal{A}(\mathtt{acc}(x.f))\} \rangle$$

And so,

$$\begin{aligned}
&\varnothing \vDash_I \phi_{\mathsf{root}} \\
&\quad \iff \varnothing \vDash_I \mathtt{acc}(x.f) \ * \ (\mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \\
&\quad \iff (\mathsf{granted}(\mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \vDash_I \mathtt{acc}(x.f)) \wedge \\
&\qquad\quad (\mathsf{granted}(\mathtt{acc}(x.f)) \vDash_I \mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \\
&\quad \iff (\mathsf{granted}(\mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \vDash_I x) \wedge \\
&\qquad\quad (\mathsf{granted}(\mathtt{acc}(x.f)) \vDash_I \mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \\
&\quad \iff \top \ \wedge \ (\mathsf{granted}(\mathtt{acc}(x.f)) \vDash_I \mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f)) \\
&\quad \iff \top \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vDash_I \mathtt{if}\ b\ \mathtt{then}\ x.f = 1\ \mathtt{else}\ \mathtt{acc}(x.f))
\end{aligned}$$

## Example 3

Define

$$\phi_{\mathsf{root}} := \mathtt{acc}(x.f) \ * \ x = y \ * \ y.f = 1$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \{\mathsf{aliased}\,\{x,y\}\}\,, \varnothing \rangle$$

And so,

$$\begin{aligned}
\vdash_{\mathsf{frm}I} \phi_{\mathsf{root}} &\iff \varnothing \vDash_I \phi_{\mathsf{root}} \\
&\iff \varnothing \vDash_I \mathtt{acc}(x.f) \ * \ x = y \ * \ y.f = 1 \\
&\iff \varnothing \vDash_I x = y \ * \ \mathtt{acc}(x.f) \ * \ y.f = 1 && (* \text{ is commutative}) \\
&\iff (\mathsf{granted}(\mathtt{acc}(x.f) \ * \ y.f = 1) \vDash_I x = y) \wedge \\
&\qquad (\mathsf{granted}(x = y) \vDash_I \mathtt{acc}(x.f) \ * \ y.f = 1) \\
&\iff \top \ \wedge \ (\mathsf{granted}(x = y) \vDash_I \mathtt{acc}(x.f) \ * \ y.f = 1) \\
&\iff \top \ \wedge \ (\varnothing \vDash_I \mathtt{acc}(x.f) \ * \ y.f = 1) \\
&\iff \top \ \wedge \ (\mathsf{granted}(y.f = 1) \vDash_I \mathtt{acc}(x.f)) \ \wedge \ (\mathsf{granted}(\mathtt{acc}(x.f)) \vDash_I y.f = 1) \\
&\iff \top \ \wedge \ (\varnothing \vDash_I \mathtt{acc}(x.f)) \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vDash_I y.f = 1) \\
&\iff \top \ \wedge \ ((\varnothing \vDash_I e) \wedge \ \sim (\varnothing \vdash \mathsf{accessed}(x.f))) \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vDash_I y.f = 1) \\
&\iff \top \ \wedge \ (\top \ \wedge \ \top) \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vDash_I y.f = 1) \\
&\iff \top \ \wedge \ \top \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vdash \mathsf{accessed}_{x.f}(x.f)) \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vDash_I 1) \\
&\iff \top \ \wedge \ \top \ \wedge \ (\{\mathsf{accessed}(x.f)\} \vdash \mathsf{accessed}_{x.f}(x.f)) \ \wedge \ \top && (\star) \\
&\iff \top \ \wedge \ \top \ \wedge \ (\top) \ \wedge \ \top \\
&\iff \top
\end{aligned}$$

$\star$ is decided to be $\top$ because in the sub-formula $\phi := x.f$,

$$\{\mathsf{accessed}(x.f)\} \vdash \mathsf{accessed}_{\phi}(x.f)$$

is true since

$$(\mathcal{A}(\phi) \vdash \mathsf{aliased}\,\{x,y\}) \ \vdash \ (\{\mathsf{accessed}(x.f)\} \vdash \mathsf{accessed}_{\phi}(x.f))$$

l != null List(l) unfolding List(l) in l.tail == null

### Example 4

Define

$$
\begin{array}{l}
\texttt{class}\ \mathsf{List}\ \{\\
\quad \texttt{int}\ \mathit{head};\\
\quad \mathsf{List}\ \mathit{tail};\\
\quad \texttt{predicate}\ \mathsf{List}(l) =\\
\quad\quad \texttt{acc}(l.\mathit{tail}) *\\
\quad\quad \texttt{if}\ l.\mathit{tail} = \texttt{null}\\
\quad\quad\quad \texttt{then true}\\
\quad\quad\quad \texttt{else}\ \mathsf{List}(l.\mathit{tail});\\
\quad \vdots\\
\}
\end{array}
$$

$$\phi_{\mathsf{root}} := l \neq \texttt{null} \ * \ \mathsf{List}(l) \ * \ \texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null}$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \{\sim \mathsf{aliased}\,\{l, \texttt{null}\}\,, \mathsf{aliased}\,\{l.\mathit{tail}, \texttt{null}\}\}\,, \varnothing \rangle$$

And so,

$$
\begin{array}{rl}
\vdash_{\mathsf{frm}I} \phi_{\mathsf{root}} & \iff \varnothing \vDash_I \phi_{\mathsf{root}}\\
& \iff \varnothing \ \vDash_I \ (l \neq \texttt{null}) \ * \ \mathsf{List}(l) \ * \ (\texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null})\\
& \iff (\mathsf{granted}(\mathsf{List}(l) \ * \ (\texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null})) \ \vDash_I \ l \neq \texttt{null}) \wedge\\
& \quad\ \ (\mathsf{granted}((\texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null}) \ * \ (l \neq \texttt{null})) \ \vDash_I \ \mathsf{List}(l)) \wedge\\
& \quad\ \ (\mathsf{granted}((l \neq \texttt{null}) \ * \ \mathsf{List}(l)) \ \vDash_I \ \texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null})\\
& \iff (\{\mathsf{assumed}(\mathsf{List}(l))\} \ \vDash_I \ l \neq \texttt{null}) \wedge\\
& \quad\ \ (\varnothing \ \vDash_I \ \mathsf{List}(l)) \wedge\\
& \quad\ \ (\{\mathsf{assumed}(\mathsf{List}(l))\} \ \vDash_I \ \texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null})\\
& \iff (\mathsf{granted}(\texttt{acc}(l.\mathit{tail}) * \texttt{if}\ l.\mathit{tail} = \texttt{null then true else}\ \mathsf{List}(l.\mathit{tail})) \ \vDash_I \ l \neq \texttt{null}) \wedge\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(expansion of } \mathsf{assumed} \text{ permission)}\\
& \quad\ \ \top \wedge\\
& \quad\ \ ((\{\mathsf{assumed}(\mathsf{List}(l))\} \vdash \mathsf{assumed}_\phi(\mathsf{List}(l))) \ \wedge \ (\{\mathsf{assumed}(\mathsf{List}(l))\} \vdash l.\mathit{tail} = \texttt{null})\\
& \iff (\{\texttt{acc}(l.\mathit{tail})\} \ \vDash_I \ l \neq \texttt{null}) \wedge\\
& \quad\ \ \top \wedge\\
& \quad\ \ (\top \ \wedge \ ((\mathsf{granted}(\texttt{acc}(l.\mathit{tail}) * \texttt{if}\ l.\mathit{tail} = \texttt{null then true else}\ \mathsf{List}(l.\mathit{tail})) \vdash l.\mathit{tail} = \texttt{null})\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(expansion of } \mathsf{assumed} \text{ permission)}\\
& \iff \top \ \wedge \ \top \ \wedge \ (\top \ \wedge \ (\{\texttt{acc}(l.\mathit{tail})\} \vdash l.\mathit{tail} = \texttt{null})\\
& \iff \top
\end{array}
$$

## Example 5

Define

$$\phi_{\mathsf{root}} := \texttt{if } x = \texttt{null then true else } (\texttt{acc}(x.f) * x.f = 1)$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \varnothing, \{x = \texttt{null} : \langle \{\mathsf{aliased}\,\{x, \texttt{null}\}\}, \varnothing \rangle, x \neq \texttt{null} : \langle \{\sim \mathsf{aliased}\,\{x, \texttt{null}\}\}, \varnothing \rangle\} \rangle$$

And so,

$$\begin{aligned}
\vdash_{\mathsf{frm}\,I} \phi_{\mathsf{root}} &\iff \varnothing \vDash_I \phi_{\mathsf{root}} \\
&\iff \varnothing \vDash_I \texttt{if } x = \texttt{null then true else } (\texttt{acc}(x.f) * x.f = 1) \\
&\iff (\varnothing \vDash_I x = \texttt{null}) \ \wedge \ (\varnothing \vDash_I \texttt{true}) \ \wedge \ (\varnothing \vDash_I \texttt{acc}(x.f) * x.f = 1) \\
&\iff \top \ \wedge \ \top \ \wedge (\mathsf{granted}(x.f = 1) \vDash_I \texttt{acc}(x.f)) \ \wedge (\mathsf{granted}(\texttt{acc}(x.f)) \vDash_I x.f = 1) \\
&\iff \top \ \wedge \ \top \ \wedge (\varnothing \vDash_I \texttt{acc}(x.f)) \ \wedge (\{\mathsf{accessed}(x.f)\} \vDash_I x.f = 1) \\
&\iff \top \ \wedge \ \top \ \wedge (\varnothing \vDash_I x) \wedge \ \sim (\varnothing \vdash \mathsf{accessed}_{\mathsf{acc}(x.f)}(x.f)) \\
&\qquad (\{\mathsf{accessed}(x.f)\} \vdash \mathsf{accessed}_{x.f}(x.f)) \ \wedge (\{\mathsf{accessed}(x.f)\} \vDash_I 1) \\
&\iff \top \ \wedge \ \top \ \wedge \top \ \wedge \ \top \ \top \ \wedge \top \\
&\iff \top
\end{aligned}$$

## Example 6

Use the definition of List from example 4. Define

$$\begin{aligned}
\phi_{\mathsf{root}} &:= (\texttt{if } l = \texttt{null then true else } \phi_1) \, * \\
&\quad\;\, (\texttt{if } l = \texttt{null then true else } \phi_2) \\
\phi_1 &:= \texttt{acc}(l.\mathit{head}) * \texttt{acc}(l.\mathit{tail}) * \mathsf{List}(l) \\
\phi_2 &:= l.\mathit{head} = 5
\end{aligned}$$

Then

$$\begin{aligned}
\mathcal{A}(\phi_{\mathsf{root}}) =& \langle \varnothing, \{ l = \texttt{null} : \mathcal{A}(true) \sqcup \mathcal{A}(true),\ l \neq \texttt{null} : \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2) \} \rangle \\
\mathcal{A}(\phi_1) =& \varnothing \\
\mathcal{A}(\phi_2) =& \varnothing
\end{aligned}$$

And so,

$$\begin{aligned}
\vdash_{\mathsf{frm}I} \phi_{\mathsf{root}} &\iff \varnothing \vDash_I \phi_{\mathsf{root}} \\
&\iff \varnothing \vDash_I (\texttt{if } l = \texttt{null then true else } \phi_1) \, * \\
&\qquad (\texttt{if } l = \texttt{null then true else } \phi_2) \\
&\iff (\varnothing \vDash_I l = \texttt{null}) \ \wedge \ (\varnothing \vDash_I \texttt{true}) \ \wedge \ (\varnothing \vDash_I \texttt{acc}(l.\mathit{head}) * \texttt{acc}(l.\mathit{tail}) * \mathsf{List}(l)) \ \wedge \ (\varnothing \vDash_I )
\end{aligned}$$

# 5 Satisfiability

# 6 Implication

## 7 Weakest Predonditions