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 ϕ be a formula. The **aliasing context** $\mathcal{A}$ of ϕ is a tree of set of aliasing proposition about aliasing of object variables that appear in ϕ. $\mathcal{A}$ needs to be a tree because the conditional sub-formulas that may appear in ϕ 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 ϕ. The parts of the tree are labeled in such a way that modularly allows a specified sub-formula of ϕ 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. ϕ has a formula-structure represented by the tree in figure 3.2. The formula-structure tree for

Figure 1: Formula structure tree for ϕ.

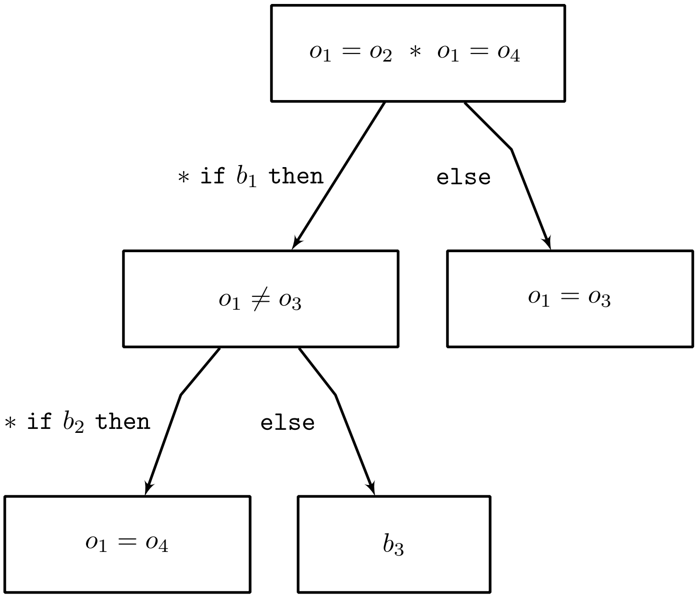
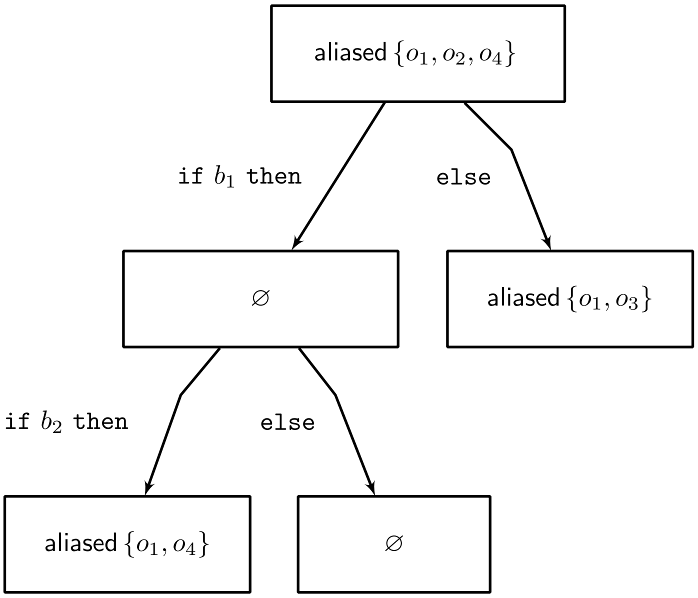


Figure 2: $\mathcal{A}(\phi)$, the aliasing context tree for ϕ.



ϕ corresponds node-for-node and edge-for-edge to the aliasing context tree in figure 3.2.

More generally, for ϕ a formula and ϕ' a sub-formula of ϕ, write $\mathcal{A}_\phi(\phi')$ as the **total aliasing context** of ϕ' which includes aliasing propositions inherited from its ancestors in the aliasing context tree of ϕ. 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 ϕ 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 ϕ is implicit and ϕ' is a sub-formula of ϕ_{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 ϕ is a tree, where nodes represent local aliasing contexts and branches represent the branches of conditional sub-formulas nested in ϕ. 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 then and else branches of conditionals directly nested in ϕ, the e_α 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 ϕ_{root}, the aliasing context of ϕ_{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 ϕ_{root}

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

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

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

$$\begin{aligned}
&\langle A_1, \{e_{1\alpha} : \mathcal{A}_{1\alpha}\} \rangle \sqcup \langle A_2, \{e_{2\alpha} : \mathcal{A}_{2\alpha}\} \rangle := \\
&\quad\quad \langle \{\mathsf{aliased}\,\{o' \mid A_1 \cup A_2 \vdash \mathsf{aliased}\,\{o, o'\}\} \mid \forall o\} \ \cup \\
&\quad\quad\quad \{\sim \mathsf{aliased}\,\{o', \tilde{o}\} \mid \forall o, o', \tilde{o} : (A_1 \cup A_2 \ \vdash \ \mathsf{aliased}\,\{o, o'\}, \sim \mathsf{aliased}\,\{o, \tilde{o}\})\}, \\
&\quad\quad \{e_\alpha : \mathcal{A}_{1\alpha} \sqcup \mathcal{A}_{2\alpha} \mid e_\alpha \iff e_{1\alpha}, e_{2\alpha}\} \rangle
\end{aligned}$$

$$\begin{aligned}
&\langle A_1, \{e_{1\alpha} : \mathcal{A}_{1\alpha}\} \rangle \sqcap \langle A_2, \{e_{2\alpha} : \mathcal{A}_{2\alpha}\} \rangle := \\
&\quad\quad \langle \{\mathsf{aliased}\,\{o' \mid A_1 \cap A_2 \vdash \mathsf{aliased}\,\{o, o'\}\} \mid \forall o\} \ \cup \\
&\quad\quad\quad \{\sim \mathsf{aliased}\,\{o', \tilde{o}\} \mid \forall o, o', \tilde{o} : (A_1 \cap A_2 \ \vdash \ \mathsf{aliased}\,\{o, o'\}, \sim \mathsf{aliased}\,\{o, \tilde{o}\})\}, \\
&\quad\quad \{e_\alpha : \mathcal{A}_{1\alpha} \sqcap \mathcal{A}_{2\alpha} \mid e_\alpha \iff e_{1\alpha}, e_{2\alpha}\} \rangle
\end{aligned}$$

3.4 Inconsistent Aliasing Contexts

As defined, it is possible for inconsistent aliasing contexts to arise. As a simple example, the formula $\phi := x = y \ \&\&\ x \neq y$ would yield the aliasing context

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

An inconsistent aliasing context is unsatisfiable, so if such an aliasing context arises then the root formula is considered not well-formed, and an Inconsistent aliasing context exception is raised. This causes an error rather than just treats the formula as unsatisfiable. Thus, framing, satisfiability, and so on can be decided under the assumption that all aliasing contexts are consistent.

4 Framing

4.1 Definitions

For framing, a formula is considered inside a **permission context**, a set of permissions, where a **permission** π 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 Π 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 ϕ be a formula. ϕ may **require** a permission π. For example, the formula $e.f = 1$ requires $\mathsf{accessed}(e.f)$, because it references $e.f$. The set of all permissions that ϕ requires is called the **requirements** of ϕ. ϕ may also **grant** a permission π. For example, the formula $\mathtt{acc}(e.f)$ grants the permission $\mathsf{accessed}(e.f)$.

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

$$\Pi \vDash_I \phi$$

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

$$\lfloor \phi \rfloor$$

Finally, a ϕ is called **self-framing** if and only if for any set of permissions Π, $\Pi \vDash_I \phi$. The proposition that ϕ 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 ϕ 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 Π and the sub-formulas of ϕ. 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 ϕ_{root} exists at in the program).

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

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

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

4.2.1 Notes

- TODO: explain how non-object-variable expressions cannot alias to anything (thus the e.f case in granted and required)

4.3 Examples

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

Example 1

Define

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

Then

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

And so,

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

$(\star)$ is decided to be $\bot$, thus yielding the entire conjunct to be decided $\bot$, because in the sub-formula $\phi := \mathsf{acc}(x.f)$,

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

contradicts the requirement of ϕ that

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

Example 2

Define

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

Then

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

And so,

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

$(\star)$ is decided to be $\bot$ because in the sub-formula $\phi := \mathtt{acc}(x.f)$,

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

contradicts the requirement of ϕ that

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

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,

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

$\star$ 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

```
class List {
    int head;
    List tail;
    predicate List(l) =
        acc(l.tail) *
        if l.tail = null
            then true
            else List(l.tail);
    ⋮
}
```

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

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{aligned}
\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 \\
& (\mathsf{granted}((\texttt{unfolding}\ \mathsf{List}(l)\ \texttt{in}\ l.\mathit{tail} = \texttt{null}) \;*\; (l \neq \texttt{null}))\ \vDash_I\ \mathsf{List}(l)) \wedge \\
& (\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 \\
& (\varnothing\ \vDash_I\ \mathsf{List}(l)) \wedge \\
& (\{\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}\ \texttt{then}\ \texttt{true}\ \texttt{else}\ \mathsf{List}(l.\mathit{tail}))\ \vDash_I\ l \neq \texttt{null}) \wedge \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{(expansion of } \mathsf{assumed} \text{ permission)} \\
& \top \wedge \\
& ((\{\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 \\
& \top \wedge \\
& (\top\ \wedge\ ((\mathsf{granted}(\texttt{acc}(l.\mathit{tail}) * \texttt{if}\ l.\mathit{tail} = \texttt{null}\ \texttt{then}\ \texttt{true}\ \texttt{else}\ \mathsf{List}(l.\mathit{tail})) \vdash l.\mathit{tail} = \texttt{null}) \\
& \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{aligned}$$

Example 5

Define

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

Then

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

And so,

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

Example 6

Use the definition of List from example 4. Define

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

Then

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

And so,

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

5 Satisfiability

6 Implication

7 Weakest Predonditions