SVL with Recursive Predicates

Henry Blanchette

Contents

1 Grammar

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

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 $\text{null} : C$ 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:

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

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
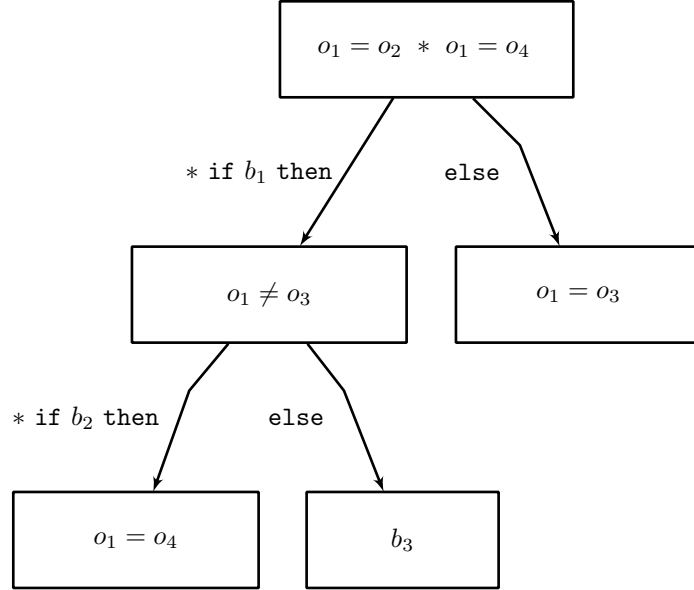
Figure 1: Formula structure tree for ϕ.

$$o_1 = o_2 \;*\; o_1 = o_4$$

$*$ if b_1 then   else

$$o_1 \neq o_3$$

$$o_1 = o_3$$

$*$ if b_2 then   else

$$o_1 = o_4$$

$$b_3$$

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

$$\text{aliased } \{o_1, o_2, o_4\}$$

if b_1 then   else

$$\varnothing$$

$$\text{aliased } \{o_1, o_3\}$$

if b_2 then   else

$$\text{aliased } \{o_1, o_4\}$$

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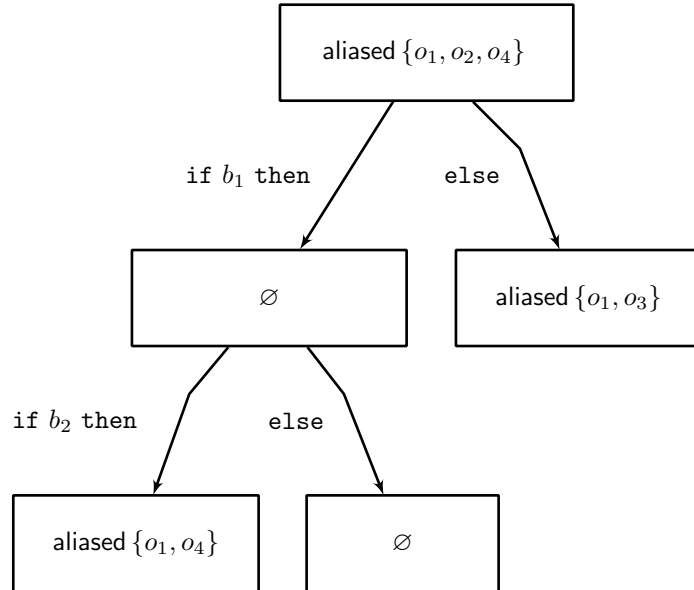
ϕ corresponds node-for-node and edge-for-edge to the aliasing context tree in figure 3.2.

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

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

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

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

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** π is to do one of the following:

- to reference $e.f$, written $\mathsf{accessed}(e.f)$.

- to assume $\alpha_C(\bar{e})$, written $\mathsf{assumed}(\alpha_C(\bar{e}))$. This allows the a single unrolling of $\alpha_C(\bar{e})$. Explicitly, an instance of $\mathsf{assumed}(\alpha_C(\bar{e}))$ 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 $\alpha_C(\bar{e})$ 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 $\mathsf{acc}(e.f)$ 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).

$$
\begin{aligned}
\Pi \vDash_I \phi \quad \Longleftrightarrow \quad & \text{match } \phi \text{ with} \\
& v && \mapsto && \top \\
& x && \mapsto && \top \\
& e_1 \oplus e_2 && \mapsto && \Pi \vDash_I e_1, e_2 \\
& e_1 \odot e_2 && \mapsto && \Pi \vDash_I e_1, e_2 \\
& e.f && \mapsto && (\Pi \vDash_I e) \ \wedge \ (\Pi \vdash \mathsf{accessed}_\phi(e.f)) \\
& \mathtt{acc}(e.f) && \mapsto && (\Pi \vDash_I e) \\
& \phi_1 \circledast \phi_2 && \mapsto && (\Pi \cup \mathsf{granted}(\phi_2) \vDash_I \phi_1) \ \wedge \\
& && && (\Pi \cup \mathsf{granted}(\phi_1) \vDash_I \phi_2) \\
& \alpha_C(e_1, \ldots, e_k) && \mapsto && \Pi \vDash_I e_1, \ldots, e_2 \\
& \mathtt{if } e \mathtt{ then } \phi_1 \mathtt{ else } \phi_2 && \mapsto && \Pi \vDash_I e, \phi_1, \phi_2 \\
& \mathtt{unfolding } \alpha_C(\bar{e}) \mathtt{ in } \phi' && \mapsto && (\Pi \vdash \mathsf{assumed}_\phi(\alpha_C(\bar{e}))) \ \wedge \ (\Pi \vDash_I \phi') \\[1em]
\mathsf{granted}(\phi) \quad := \quad & \text{match } \phi \text{ with} \\
& e && \mapsto && \varnothing \\
& \mathtt{acc}(e.f) && \mapsto && \{\mathsf{accessed}(e.f)\} \\
& \phi_1 \circledast \phi_2 && \mapsto && \mathsf{granted}(\phi_1) \cup \mathsf{granted}(\phi_2) \\
& \alpha_C(\bar{e}) && \mapsto && \{\mathsf{assumed}(\alpha_C(\bar{e}))\} \\
& \mathtt{if } e \mathtt{ then } \phi_1 \mathtt{ else } \phi_2 && \mapsto && \mathsf{granted}(\phi_1) \cap \mathsf{granted}(\phi_2) \\
& \mathtt{unfolding } \alpha_C(\bar{e}) \mathtt{ in } \phi' && \mapsto && \mathsf{granted}(\phi')
\end{aligned}
$$

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

$$
\Pi \vdash \mathsf{accessed}_\phi(o.f) \iff \exists o' \in O : (\mathcal{A}(\phi) \vdash \mathsf{aliased} \{o, o'\}) \ \wedge \ (\mathsf{accessed}(o'.f) \in \Pi)
$$

$$
\begin{aligned}
\Pi \vdash \mathsf{assumed}_\phi(\alpha_C(e_1, \ldots, e_k)) \iff & (\forall i : (e_i = e_i') \ \vee \\
& (\exists o, o' : ((o, o') = (e_i, e_i')) \wedge \mathcal{A}(\phi) \vdash \mathsf{aliased} \{o, o'\})) \\
& \wedge \ (\mathsf{assumed}(\alpha_C(e_1', \ldots, e_k')) \in \Pi)
\end{aligned}
$$

4.3 Examples

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

Example 1

Define

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

Then

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

And so,

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

Example 2

Define

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

Then

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

And so,

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

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,

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

Example 4

Define

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

$$\phi_{\mathsf{root}} := \mathsf{List}(l) \;*\; \mathtt{unfolding}\ \mathsf{List}(l)\ \mathtt{in}\ l.head = 1$$

Then

$$\mathcal{A}(\phi_{\mathsf{root}}) = \langle \varnothing, t.tail = \mathtt{null} : \langle \{\mathsf{aliased}\ \{t.tail, \mathtt{null}\}\}, \varnothing \rangle,\ t.tail \neq \mathtt{null} : \langle \varnothing, \varnothing \rangle \rangle$$

And so,

$$
\begin{aligned}
\vdash_{\mathsf{frm}I} \phi_{\mathsf{root}} \iff\ & \varnothing \vDash_I \phi_{\mathsf{root}} \\
\iff\ & \varnothing\ \vDash_I\ \mathsf{List}(l)\ *\ \mathtt{unfolding}\ \mathsf{List}(l)\ \mathtt{in}\ l.head = 1 \\
\iff\ & (\mathsf{granted}(\mathtt{unfolding}\ \mathsf{List}(l)\ \mathtt{in}\ l.head = 1) \vDash_I \mathsf{List}(l))\ \wedge \\
& (\mathsf{granted}(\mathsf{List}(l)) \vDash_I \mathtt{unfolding}\ \mathsf{List}(l)\ \mathtt{in}\ l.head = 1) \\
\iff\ & \top\ \wedge\ (\{\mathsf{assumed}(\mathsf{List}(l))\} \vDash_I \mathtt{unfolding}\ \mathsf{List}(l)\ \mathtt{in}\ l.head = 1) \\
\iff\ & \top\ \wedge\ (\mathsf{granted}(l \neq \mathtt{null}\ *\ \mathsf{acc}(l.head)\ *\ \mathsf{acc}(l.tail)\ * \\
& \quad \mathtt{if}\ l.tail = \mathtt{null}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathsf{List}(l.tail)) \vDash_I \\
& \hspace{8cm} (\text{expansion of } \mathsf{assumed}(\mathsf{List}(l))) \\
& l.head = 1) \\
\iff\ & \top\ \wedge\ (\{\mathsf{accessed}(l.head), \mathsf{accessed}(l, tail)\}) \vDash_I l.head = 1) \\
\iff\ & \top\ \wedge\ (\{\mathsf{accessed}(l.head), \mathsf{accessed}(l, tail)\}) \vdash \mathsf{accessed}_{\phi_{\mathsf{root}}}(l.head)) \\
\iff\ & \top\ \wedge\ \top \\
\iff\ & \top
\end{aligned}
$$

Example 5

Define

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

Then

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

And so,

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

Example 6

Use the definition of List from example 4. Define

$$\phi_{\text{root}} := (\texttt{if } l = \texttt{null then true else } \phi_1) *$$
$$(\texttt{if } l = \texttt{null then true else } \phi_2)$$
$$\phi_1 := \texttt{acc}(l.head) * \texttt{acc}(l.tail) * \textsf{List}(l)$$
$$\phi_2 := l.head = 5$$

Then

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

And so,

$$\vdash_{\textsf{frm}I} \phi_{\text{root}} \iff \varnothing \vDash_I \phi_{\text{root}}$$
$$\iff \varnothing \vDash_I (\texttt{if } l = \texttt{null then true else } \phi_1) *$$
$$(\texttt{if } l = \texttt{null then true else } \phi_2)$$
$$\iff (\varnothing \vDash_I l = \texttt{null}) \wedge (\varnothing \vDash_I \texttt{true}) \wedge (\varnothing \vDash_I \texttt{acc}(l.head) * \texttt{acc}(l.tail) * \textsf{List}(l)) \wedge (\varnothing \vDash_I)$$

15

5   Satisfiability

6   Implication

7 Weakest Predonditions