

SVL with Recursive Predicates

Henry Blanchette

Contents

1	Grammar	2
2	Well-formedness	3
3	Aliasing	4
3.1	Definitions	4
3.2	Aliasing Context	4
3.3	Constructing an Aliasing Context	6
4	Framing	8
4.1	Definitions	8
4.2	Deciding Framing	9
4.3	Examples	10
5	Satisfiability	18
6	Implication	19
7	Weakest Predonditions	20

1 Grammar

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

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

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

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

3.2 Aliasing Context

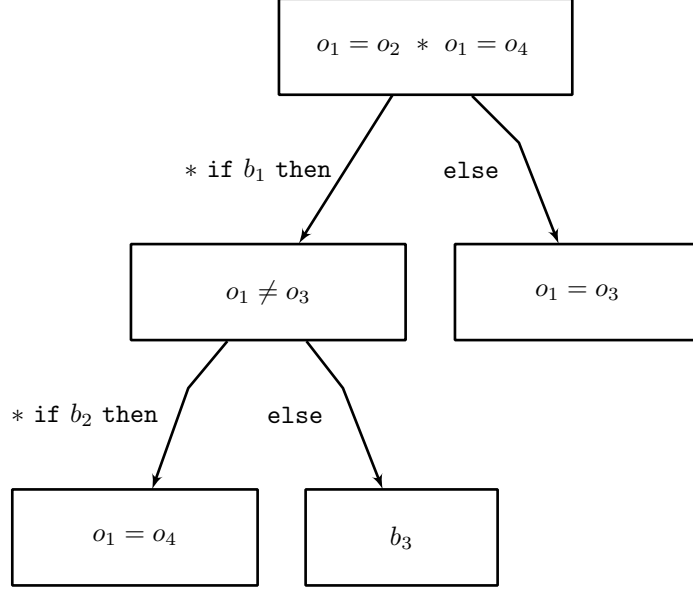
Let ϕ be a formula. The **aliasing context** \mathcal{A} of ϕ is a tree of set of aliasing proposition about aliasing of object variables that appear in ϕ . \mathcal{A} needs to be a tree because the conditional and unfolding sub-formulas that may appear in ϕ allow for branching aliasing contexts not expressible flatly at the top level. In the case of conditionals i.e. sub-formulas of the form **if** e **then** ϕ_1 **else** ϕ_2 , two branches sprout from the original context. In the case of unfoldings i.e. sub-formulas of the form **unfolding** $\alpha_C(\bar{e})$ in ϕ , one branch sprouts from the original context. 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:

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

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 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**

Figure 1: Formula structure tree for ϕ .



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:

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

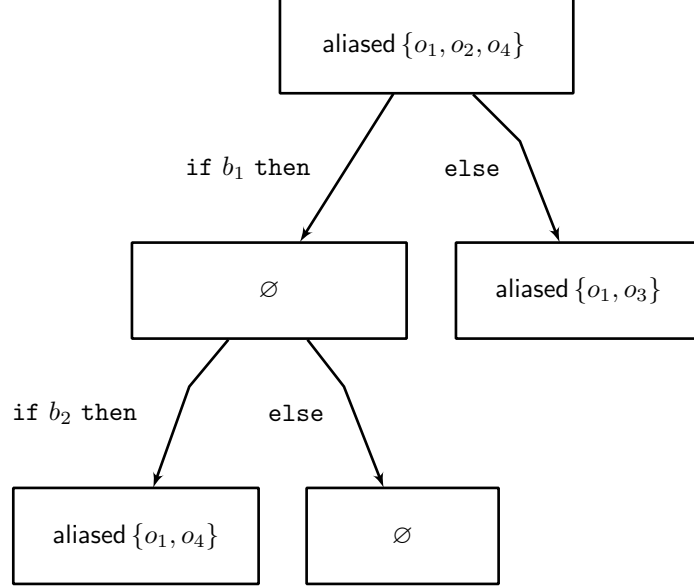
along with the fact that it has no child branches. Usually $\mathcal{A}_{\phi_{\text{root}}}(\phi')$ is abbreviated to $\mathcal{A}(\phi')$ when the top level formula ϕ is implicit and ϕ' is a sub-formula of ϕ_{root} .

An aliasing context \mathcal{A} may entail $\text{aliased}(O)$ 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 $\text{aliased}(O) \notin \mathcal{A}$ yet still the previous judgement holds. For example, this is true when $\exists O' \subset \mathcal{O}$ such that $O \subset O'$ and $\text{aliased}(O') \in \mathcal{A}$. So, the explicit definition for making this judgement is as follows:

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

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

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



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

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

where A is a set of propositions about aliasing and the $l_\alpha : \mathcal{A}_\alpha$ are the nesting aliasing contexts that correspond to the branches of conditionals and unfoldings directly nested in ϕ , the l_α being labels for each child context.

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

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

$\mathcal{A}(\phi) :=$	match ϕ with	
v		$\mapsto \langle \emptyset, \emptyset \rangle$
x		$\mapsto \langle \emptyset, \emptyset \rangle$
$e_1 \&\& e_2$		$\mapsto \mathcal{A}(e_1) \sqcup \mathcal{A}(e_2)$
$e_1 \parallel e_2$		$\mapsto \mathcal{A}(\text{if } e_1 \text{ then true else } e_2)$
$e_1 \oplus e_2$		$\mapsto \langle \emptyset, \emptyset \rangle$
$o_1 = o_2$		$\mapsto \langle \{\text{aliases}\{o_1, o_2\}\}, \emptyset \rangle$
$e_1 \odot e_2$		$\mapsto \langle \emptyset, \emptyset \rangle$
$e.f$		$\mapsto \langle \emptyset, \emptyset \rangle$
$\text{acc}(e.f)$		$\mapsto \langle \emptyset, \emptyset \rangle$
$\phi_1 * \phi_2$		$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\phi_1 \wedge \phi_2$		$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\alpha_C(\bar{e})$		$\mapsto \langle \emptyset, \emptyset \rangle$
$\text{if } e \text{ then } \phi_1 \text{ else } \phi_2$		$\mapsto \langle \emptyset, \{e : \mathcal{A}(e) \sqcup \mathcal{A}(\phi_1), \sim e : (\mathcal{A}(\sim e)) \sqcup \mathcal{A}(\phi_2)\} \rangle$
$\text{unfolding } \alpha_C(\bar{e}) \text{ in } \phi'$		$\mapsto \langle \emptyset, \{\text{unfolding}(\alpha_C(\bar{e})) : \mathcal{A}(\text{unfold } \alpha_C(\bar{e})) \sqcup \mathcal{A}(\phi')\} \rangle$

Note the following:

- The $\sim e$ expression in the result of the rule for $\mathcal{A}(\text{if } e \text{ then } \phi_1 \text{ else } \phi_2)$ means to negate the boolean expression of e
- The $e_1 \parallel e_2$ expression is translated into $\text{if } e_1 \text{ then true else } e_2$ for the purpose of aliasing. So, boolean or operations in formulas yield branching just like conditional expressions.
- The $\text{unfold } \alpha_C(\bar{e})$ expression in the result of the rule for $\mathcal{A}(\text{unfolding } \alpha_C(\bar{e}) \text{ in } \phi')$ is translated to a single unfolding of the body of $\alpha_C(\bar{e})$ with the arguments substituted appropriately.

As examples,

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

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

$$\begin{aligned} \langle A_1, \{l_\alpha : \mathcal{A}_\alpha\} \rangle \sqcup \langle A_2, \{l_\beta : \mathcal{A}_\beta\} \rangle &:= \\ \langle \{\text{aliased}\{o' \mid \forall o' : (A_1 \vdash \text{aliased}\{o, o'\}) \vee (A_2 \vdash \text{aliased}\{o, o'\})\} \mid \forall o\}, \\ \{l_\alpha : \mathcal{A}_\alpha\} \cup \{l_\beta : \mathcal{A}_\beta\} \rangle \\ \langle A_1, \{l_\alpha : \mathcal{A}_\alpha\} \rangle \sqcap \langle A_2, \{l_\beta : \mathcal{A}_\beta\} \rangle &:= \\ \langle \{\text{aliased}\{o' \mid \forall o' : (A_1 \vdash \text{aliased}\{o, o'\}) \wedge (A_2 \vdash \text{aliased}\{o, o'\})\} \mid \forall o\}, \\ \{l_\alpha : \mathcal{A}_\alpha\} \cap \{l_\beta : \mathcal{A}_\beta\} \rangle \end{aligned}$$

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 **accessed**($e.f$).
- to assume $\alpha_C(\bar{e})$, written **assumed**($\alpha_C(\bar{e})$). This allows the a single unrolling of $\alpha_C(\bar{e})$. Explicitly, an instance of **assumed**($\alpha_C(\bar{e})$) in a set of permissions Π may be expanded into $\Pi \cup \mathbf{granted}(\dots)$ where \dots is replaced with a single unrolling of the body of $\alpha_C(\bar{e})$ with the arguments substituted appropriately¹.

Let ϕ be a formula. ϕ may **require** a permission π . For example, the formula $e.f = 1$ requires **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 **acc**($e.f$) grants the permission **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

$$\Pi \models_I \phi$$

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

$$[\phi]$$

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

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

Note that $\vdash_{\text{frm}I} \phi \iff \emptyset \models_I \phi$, in other words ϕ is self-framing if and only if it grants all of the permissions it requires. Or in other words still, $[\phi] = \emptyset$.

¹As demonstrated by this description, **assumed** predicates are really just a useful shorthand and not a fundamentally new type of permission. The only kind fundamental kind of permission is **accessed**.

4.2 Deciding Framing

Deciding $\Pi \models_I \phi$ must take into account the requirements, granted, and aliases contained in Π and the sub-formulas of ϕ . The following recursive algorithm decides $\Pi \models_I \phi_{root}$, 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).

$\Pi \models_I \phi$	\iff	match ϕ with	
		v	$\mapsto \top$
		x	$\mapsto \top$
		$e_1 \oplus e_2$	$\mapsto \Pi \models_I e_1, e_2$
		$e_1 \odot e_2$	$\mapsto \Pi \models_I e_1, e_2$
		$e.f$	$\mapsto (\Pi \models_I e) \wedge (\Pi \vdash \text{accessed}_\phi(e.f))$
		$\text{acc}(e.f)$	$\mapsto (\Pi \models_I e)$
		$\phi_1 \otimes \phi_2$	$\mapsto (\Pi \cup \text{granted}(\phi_2) \models_I \phi_1) \wedge$ $(\Pi \cup \text{granted}(\phi_1) \models_I \phi_2)$
		$\alpha_C(e_1, \dots, e_k)$	$\mapsto \Pi \models_I e_1, \dots, e_k$
		if e then ϕ_1 else ϕ_2	$\mapsto \Pi \models_I e, \phi_1, \phi_2$
		unfolding $\alpha_C(\bar{e})$ in ϕ'	$\mapsto (\Pi \vdash \text{assumed}_\phi(\alpha_C(\bar{e}))) \wedge (\Pi \models_I \phi')$
$\text{granted}(\phi)$	$:=$	match ϕ with	
		e	$\mapsto \emptyset$
		$\text{acc}(e.f)$	$\mapsto \{\text{accessed}(e.f)\}$
		$\phi_1 \otimes \phi_2$	$\mapsto \text{granted}(\phi_1) \cup \text{granted}(\phi_2)$
		$\alpha_C(\bar{e})$	$\mapsto \{\text{assumed}(\alpha_C(\bar{e}))\}$
		if e then ϕ_1 else ϕ_2	$\mapsto \text{granted}(\phi_1) \cap \text{granted}(\phi_2)$
		unfolding $\alpha_C(\bar{e})$ in ϕ'	$\mapsto \text{granted}(\phi')$

Where accessed_ϕ and assumed_ϕ indicate the respective propositions considered within the total alias context (including inherited aliasing contexts). More explicitly,

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

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

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

And so,

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

Example 2

Define

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

Then

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

And so,

$$\begin{aligned}
\vdash_{\text{frm } I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\
&\iff \emptyset \models_I \text{acc}(x.f) * (\text{if } x.f = 1 \text{ then true else acc}(x.f)) \\
&\iff (\text{granted}(\text{if } x.f = 1 \text{ then true else acc}(x.f)) \models_I \text{acc}(x.f)) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f)) \models_I \text{if } x.f = 1 \text{ then true else acc}(x.f)) \\
&\iff (\text{granted}(\text{if } x.f = 1 \text{ then true else acc}(x.f)) \models_I x) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f)) \models_I \text{if } x.f = 1 \text{ then true else acc}(x.f)) \\
&\iff \top \wedge (\text{granted}(\text{acc}(x.f)) \models_I \text{if } x.f = 1 \text{ then true else acc}(x.f)) \\
&\iff \top \wedge (\{\text{accessed}(x.f)\} \models_I \text{if } x.f = 1 \text{ then true else acc}(x.f)) \\
&\iff \top \wedge (\{\text{accessed}(x.f)\} \models_I x.f = 1) \wedge (\{\text{accessed}(x.f)\} \models_I \text{true}) \wedge \\
&\quad (\{\text{accessed}(x.f)\} \models_I \text{acc}(x.f)) \\
&\iff \top \wedge (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{\phi_{\text{root}}}(x.f)) \wedge \\
&\quad \top \wedge (\{\text{accessed}(x.f)\} \models_I x) \\
&\iff \top \wedge \top \wedge \top \wedge \top \\
&\iff \top
\end{aligned}$$

Example 3

Define

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

Then

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

And so,

$$\begin{aligned}
\vdash_{\text{frm}I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\
&\iff \emptyset \models_I \text{acc}(x.f) * x = y * y.f = 1 \\
&\iff (\text{granted}(x = y * y.f = 1) \models_I \text{acc}(x.f)) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f) * y.f = 1) \models_I x = y) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f) * x = y) \models_I y.f = 1) \\
&\iff (\text{granted}(x = y * y.f = 1) \models_I x) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f) * y.f = 1) \models_I x, y) \wedge \\
&\quad (\text{granted}(\text{acc}(x.f) * x = y) \models_I y.f) \\
&\iff \top \wedge \top \wedge (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{\phi_{\text{root}}}(y.f)) \\
&\iff \top \wedge \top \wedge \top \\
&\iff \top
\end{aligned}$$

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);
}
```

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

Then

$$\mathcal{A}(\phi_{\text{root}}) = \langle \emptyset, \{ \text{unfolding}(\text{List}(l)) : \{ \langle \emptyset, \{ t.\text{tail} = \text{null} : \langle \{ \text{aliased} \{ t.\text{tail}, \text{null} \} \}, \emptyset \rangle, t.\text{tail} \neq \text{null} : \langle \emptyset, \emptyset \rangle \rangle \} \} \rangle \rangle$$

And so,

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

Example 5

Define

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

Then

$$\mathcal{A}(\phi_{\text{root}}) = \langle \emptyset, \{x = \text{null} : \langle \{\text{aliased}\{x, \text{null}\}\}, \emptyset \rangle, x \neq \text{null} : \langle \emptyset, \emptyset \rangle \rangle$$

And so,

$$\begin{aligned} \vdash_{\text{frm} I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\ &\iff \emptyset \models_I \text{if } x = \text{null} \text{ then true else } (\text{acc}(x.f) * x.f = 1) \\ &\iff (\emptyset \models_I x = \text{null}) \wedge (\emptyset \models_I \text{true}) \wedge (\emptyset \models_I \text{acc}(x.f) * x.f = 1) \\ &\iff \top \wedge \top \wedge (\text{granted}(x.f = 1) \models_I \text{acc}(x.f)) \wedge (\text{granted}(\text{acc}(x.f)) \models_I x.f = 1) \\ &\iff \top \wedge \top \wedge (\emptyset \models_I \text{acc}(x.f)) \wedge (\{\text{accessed}(x.f)\} \models_I x.f = 1) \\ &\iff \top \wedge \top \wedge (\emptyset \models_I x) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{x.f}(x.f)) \wedge (\{\text{accessed}(x.f)\} \models_I 1) \\ &\iff \top \wedge \top \wedge \top \wedge \top \wedge \top \\ &\iff \top \end{aligned}$$

Example 6

Use the definition of `List` from example 4. Define

$$\begin{aligned}\phi_{\text{root}} &:= \text{acc}(x.f) * \phi_1 * \phi_2 \\ \phi_1 &:= \text{if } x.f = 1 \text{ then } x = y \text{ else true} \\ \phi_2 &:= \text{if } x.f = 1 \text{ then } y.f = 1 \text{ else true}\end{aligned}$$

Then

$$\mathcal{A}(\phi_{\text{root}}) = \langle \emptyset, \{x.f = 1 : \langle \{\text{aliased}\{x, y\}\}, \emptyset \rangle, x.f \neq 1 : \langle \emptyset, \emptyset \rangle \rangle$$

And so,

$$\begin{aligned}\vdash_{\text{frm}I} \phi_{\text{root}} &\iff \emptyset \models_I \text{acc}(x.f) * \phi_1 * \phi_2 \\ &\iff (\text{granted}(\phi_1 * \phi_2) \models_I \text{acc}(x.f)) \wedge (\text{granted}(\text{acc}(x.f) * \phi_2) \models_I \phi_1) \wedge \\ &\quad (\text{granted}(\text{acc}(x.f) * \phi_1) \models_I \phi_2) \\ &\iff (\text{granted}(\phi_1 * \phi_2) \models_I x) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \models_I \text{if } x.f = 1 \text{ then } x = y \text{ else true}) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \models_I \text{if } x.f = 1 \text{ then } y.f = 1 \text{ else true}) \\ &\iff \top \wedge (\{\text{accessed}(x.f)\} \models_I (x.f = 1), (x = y), (\text{true})) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \models_I (x.f = 1), (y.f = 1), (\text{true})) \\ &\iff \top \wedge (\{\text{accessed}(x.f)\} \models_I x.f) \wedge (\{\text{accessed}(x.f)\} \models_I x.f) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \models_I y.f) \\ &\iff \top \wedge (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{\phi_{\text{root}}}(x.f)) \wedge \\ &\quad (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{y.f=1}(y.f)) \\ &\iff \top \wedge \top \wedge \top \tag{\star} \\ &\iff \top\end{aligned}$$

(\star): $\{\text{accessed}(x.f)\} \vdash \text{accessed}_{y.f=1}(y.f) \iff \top$ since $\mathcal{A}(y.f = 1) \vdash \text{aliased}\{x, y\}$ because $\mathcal{A}(y.f = 1)$ and $\mathcal{A}(x = y)$ are combined into a single branch of $\mathcal{A}(\phi_{\text{root}})$, as they have the same conditions.

Example 7

Define

$$\phi_{\text{root}} := \text{if } x = y \text{ then acc}(x.f) \text{ else } x.f = 2$$

Then

$$\mathcal{A}(\phi_{\text{root}}) = \langle \emptyset, \{x = y : \langle \{\text{aliased}\{x, y\}\}, \emptyset \rangle, x \neq y : \langle \emptyset, \emptyset \rangle \} \rangle$$

And so,

$$\begin{aligned} \vdash_{\text{frm} I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\ &\iff \emptyset \models_I \text{if } x = y \text{ then acc}(x.f) \text{ else } x.f = 2 \\ &\iff \emptyset \models_I (x = y), (\text{acc}(x.f)), (x.f = 2) \\ &\iff \top \wedge (\emptyset \models_I x) \wedge (\emptyset \models_I x.f) \\ &\iff \top \wedge \top \wedge (\emptyset \vdash \text{accessed}_{x.f=2}(x.f)) \\ &\iff \top \wedge \top \wedge \perp \\ &\iff \perp \end{aligned}$$

Example 8

Define

$$\text{predicate aliasChoice}(x, y, z) := x = y \parallel x = z$$

$$\begin{aligned}\phi_{\text{root}} &:= \text{acc}(x.f) * \text{aliasChoice}(x, y, z) * \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1 \\ \phi_1 &:= y.f = 1 \parallel z.f = 1\end{aligned}$$

Then

$$\begin{aligned}\mathcal{A}(\phi_{\text{root}}) = \langle \emptyset, \{ &\text{unfolding}(\text{aliasChoice}(x, y, z)) : \\ &\{x = y : \langle \{\text{aliased}\{x, y\}\}, \emptyset \rangle, \\ &x \neq y : \langle \{\text{aliased}\{x, z\}\}, \emptyset \rangle, \\ &y.f = 1 : \langle \emptyset, \emptyset \rangle, \\ &y.f \neq 1 : \langle \emptyset, \emptyset \rangle \} \rangle\end{aligned}$$

Note that the $x = y \parallel x = z$ in the body of `aliasChoice` is translated to `if $x = y$ then true else $x = z$` when construction $\mathcal{A}(\phi_1)$. And so,

$$\begin{aligned}\vdash_{\text{frm}I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\ &\iff \emptyset \models_I \text{acc}(x.f) * \text{aliasChoice}(x, y, z) * \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1 \\ &\iff (\text{granted}(\text{aliasChoice}(x, y, z) * \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1) \models_I \text{acc}(x.f)) \wedge \\ &\quad (\text{granted}(\text{acc}(x.f) * \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1) \models_I \text{aliasChoice}(x, y, z)) \wedge \\ &\quad (\text{granted}(\text{acc}(x.f) * \text{aliasChoice}(x, y, z)) \models_I \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1) \\ &\iff \top \wedge (\{\text{accessed}(x.f)\} \models_I \text{aliasChoice}(x, y, z)) \wedge \\ &\quad (\{\text{accessed}(x.f), \text{assumed}(\text{aliasChoice}(x, y, z))\} \models_I \text{unfolding}(\text{aliasChoice}(x, y, z)) \text{ in } \phi_1) \\ &\iff \top \wedge \top \wedge (\{\text{accessed}(x.f)\} \models_I y.f = 1 \parallel z.f = 1) \\ &\iff \top \wedge \top \wedge (\{\text{accessed}(x.f)\} \models_I y.f) \wedge (\{\text{accessed}(x.f)\} \models_I z.f) \\ &\iff \top \wedge \top \wedge (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{y.f=1}(y.f)) \wedge (\{\text{accessed}(x.f)\} \vdash \text{accessed}_{z.f=1}(z.f)) \\ &\quad (\star) \\ &\iff \top \wedge \top \wedge \perp \wedge \perp \\ &\iff \perp\end{aligned}$$

(\star): The `accessed` to $y.f, z.f$ are not framed because it is statically undetermined which branch of $x = y \parallel x = z$ will be taken. The case could arise that $x = z$ and then when checking the condition $y.f = 1$ there is not access to $y.f$. The idea of the original formula can be correctly captured in one of the following revisions:

$$\begin{aligned}\phi'_{\text{root}} &:= \text{acc}(x.f) * (x = y \parallel x = z) * \text{if } x = y \text{ then } y.f = 1 \text{ else } z.f = 1 \\ \phi'_{\text{root}} &:= \text{acc}(x.f) * \text{if } x = y \text{ then } y.f = 1 \text{ else } (\text{if } x = z \text{ then } z.f = 1 \text{ else false})\end{aligned}$$

For example. the $z.f = 1$ will be framed because the aliasing context of the $x \neq y$ branch of $(x = y \parallel x = z)$ will be combined with the aliasing context of the $x \neq y$ branch of $(\text{if } x = y \text{ then } y.f = 1 \text{ else } z.f = 1)$, yielding `aliased` $\{x, z\}$ in $z.f = 1$. The similar case holds for the $x = y$ branches combining to allow the aliasing to frame $y.f = 1$.

5 Satisfiability

6 Implication

7 Weakest Predonditions