

# SVL with Recursive Predicates

Henry Blanchette

## Contents

<b>1</b>	<b>Grammar</b>	<b>2</b>
<b>2</b>	<b>Well-formedness</b>	<b>3</b>
<b>3</b>	<b>Aliasing</b>	<b>4</b>
3.1	Definitions . . . . .	4
3.2	Aliasing Context . . . . .	4
3.3	Constructing an Aliasing Context . . . . .	6
<b>4</b>	<b>Framing</b>	<b>8</b>
4.1	Definitions . . . . .	8
4.2	Deciding Framing . . . . .	9
4.3	Examples . . . . .	10
<b>5</b>	<b>Satisfiability</b>	<b>16</b>
<b>6</b>	<b>Implication</b>	<b>17</b>
<b>7</b>	<b>Weakest Predonditions</b>	<b>18</b>

# 1 Grammar

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

## 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 **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  $\phi$  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:

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

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

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

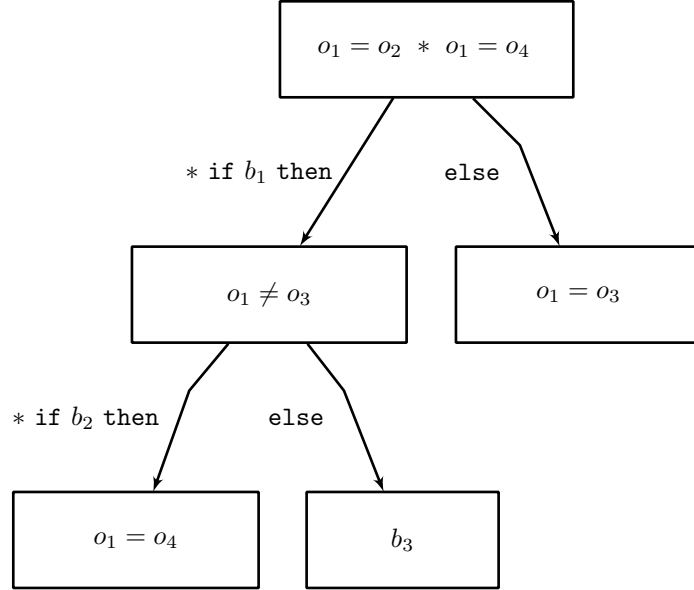
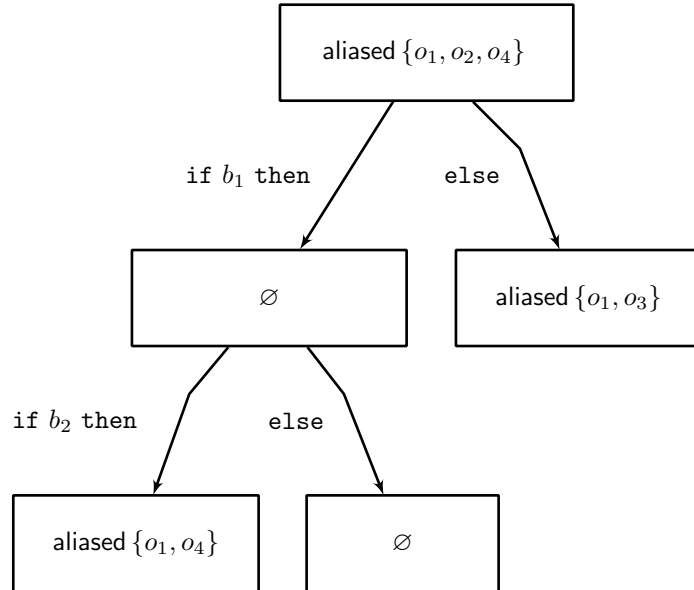


Figure 2:  $\mathcal{A}(\phi)$ , the aliasing context tree for  $\phi$ .



$\phi$  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:

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

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  $\phi$  is implicit and  $\phi'$  is a sub-formula of  $\phi_{\text{root}}$ .

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  $\mathcal{A}$ .

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

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

$\mathcal{A}(\phi) \quad := \quad \text{match } \phi \text{ with}$	
$v$	$\mapsto \langle \emptyset, \emptyset \rangle$
$x$	$\mapsto \langle \emptyset, \emptyset \rangle$
$e_1 \&\& e_2$	$\mapsto \mathcal{A}(e_1) \sqcup \mathcal{A}(e_2)$
$e_1 \parallel e_2$	$\mapsto \mathcal{A}(e_1) \sqcap \mathcal{A}(e_2)$
$e_1 \oplus e_2$	$\mapsto \langle \emptyset, \emptyset \rangle$
$o_1 = o_2$	$\mapsto \langle \{\text{aliases } \{o_1, o_2\}\}, \emptyset \rangle$
$e_1 \odot e_2$	$\mapsto \langle \emptyset, \emptyset \rangle$
$e.f$	$\mapsto \langle \emptyset, \emptyset \rangle$
$\text{acc}(e.f)$	$\mapsto \langle \emptyset, \emptyset \rangle$
$\phi_1 * \phi_2$	$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\phi_1 \wedge \phi_2$	$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\alpha_C(e_1, \dots, e_k)$	$\mapsto \langle \emptyset, \emptyset \rangle$
<b>if</b> $e$ <b>then</b> $\phi_1$ <b>else</b> $\phi_2$	$\mapsto \langle \emptyset, \{\mathcal{A}(e) \sqcup \mathcal{A}(\phi_1), (\mathcal{A}(\sim e)) \sqcup \mathcal{A}(\phi_2)\} \rangle$
<b>unfolding</b> $\alpha_C(e_1, \dots, e_k)$ <b>in</b> $\phi'$	$\mapsto \mathcal{A}(\phi')$

Note that the  $\mathcal{A}(\sim e)$  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  $e$  and then take the aliasing context of that. 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.

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

## 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 **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  $\Pi$  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 appropriately<sup>1</sup>.

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

$$\Pi \models_I \phi$$

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

$$[\phi]$$

Finally, a  $\phi$  is called **self-framing** if and only if for any set of permissions  $\Pi$ ,  $\Pi \models_I \phi$ . The proposition that  $\phi$  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  $\phi$  is self-framing if and only if it grants all of the permissions it requires. Or in other words still,  $[\phi] = \emptyset$ .

---

<sup>1</sup>As 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  $\Pi$  and the sub-formulas of  $\phi$ . 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  $\phi_{root}$  exists at in the program).

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

Where  $\text{accessed}_\phi$  and  $\text{assumed}_\phi$  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

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

Then

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

And so,

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

### 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,

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

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

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

is true since

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

`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);
  :
}
```

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

Then

$$\mathcal{A}(\phi_{\text{root}}) = \langle \{\sim \text{aliased } \{l, \text{null}\}, \text{aliased } \{l.\text{tail}, \text{null}\}\}, \emptyset \rangle$$

And so,

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

## Example 5

Define

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

Then

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

And so,

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

## Example 6

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

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

Then

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

And so,

$$\begin{aligned}\vdash_{\text{frm}I} \phi_{\text{root}} &\iff \emptyset \models_I \phi_{\text{root}} \\ &\iff \emptyset \models_I (\text{if } l = \text{null} \text{ then true else } \phi_1) * \\ &\quad (\text{if } l = \text{null} \text{ then true else } \phi_2) \\ &\iff (\emptyset \models_I l = \text{null}) \wedge (\emptyset \models_I \text{true}) \wedge (\emptyset \models_I \text{acc}(l.\text{head}) * \text{acc}(l.\text{tail}) * \text{List}(l)) \wedge (\emptyset \models_I)\end{aligned}$$

## 5 Satisfiability



## 6 Implication

## 7 Weakest Predonditions