

Contents

1	Aliasing	1
1.1	Definitions	1
1.2	Aliasing Context	1
1.3	Constructing an Aliasing Context	3
2	Framing	5
2.1	Definitions	5
2.2	Deciding Framing	6
2.2.1	Notes	6

1 Aliasing

1.1 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 .

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

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

An $O \subset \mathcal{O}$ **non-aliases** if and only if each $o \in O$ refers to separate memory in the heap as each other, written propositionally as

$$\forall o, \tilde{o} : o \neq \tilde{o} \iff \text{non-aliases}(O)$$

1.2 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,

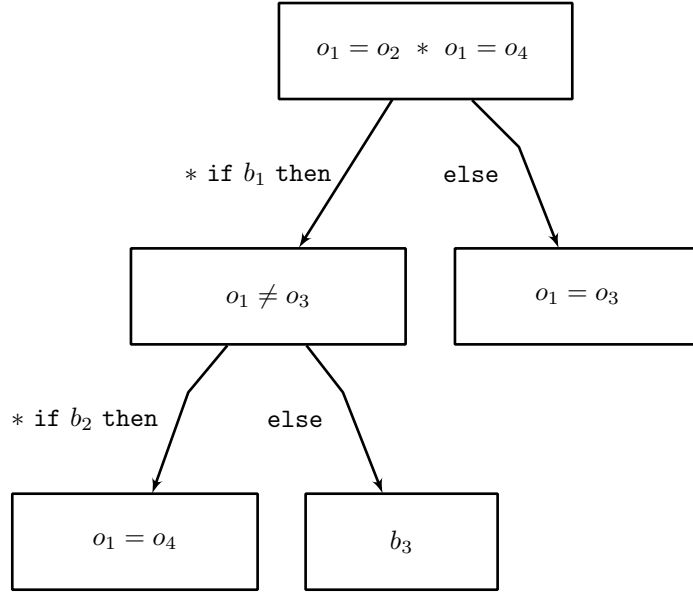
and each branch refers to a branch of a unique conditional in ϕ . The parts of the tree are labeled in such a way that modularly allows a specified sub-formula of ϕ to be matched to the unique aliasing sub-context that corresponds to it.

For example, consider the following formula:

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

ϕ has a formula-structure represented by the tree in figure 1.2. The formula-structure tree for ϕ corresponds node-for-node and edge-for-edge to the aliasing context tree in figure 1.2. where a node inherits all the aliasing assertions of its parents. So for example, the aliasing

Figure 1: Formula structure tree for ϕ .

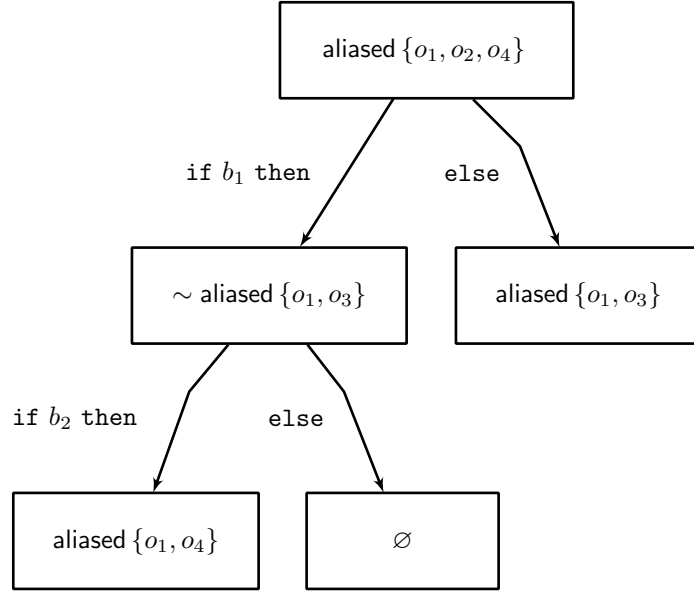


context for the sub-formula $(o_1 = o_4)$ of ϕ is:

$$\mathcal{A}_\phi(o_1 = o_4) := \{\text{aliased } \{o_1, o_2, o_4\}, \sim \text{aliased } \{o_1, o_3\}, \sim \text{aliased } \{o_2, o_3\}, \sim \text{aliased } \{o_3, o_4\}\}$$

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 including aliasing propositions inherited from its ancestors in

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



the aliasing context tree of ϕ . Usually $\mathcal{A}_\phi(\phi')$ is abbreviated to $\mathcal{A}(\phi')$ when the top level formula ϕ is implicit.

An aliasing context \mathcal{A} may entail a proposition P about aliasing. This judgement is written

$$\mathcal{A} \vdash P.$$

For a set of object variables O , such propositions to consider come in two forms

$$P ::= \text{aliases}(O) \mid \sim \text{aliases}(O).$$

Since \mathcal{A} is efficiently represented as a set of propositions about sets, it may be the case that $P \notin \mathcal{A}$ yet still $\mathcal{A} \vdash P$. So, the general aliasing judgments are decided in the following ways for each case:

$$\begin{aligned} \mathcal{A} \vdash \text{aliases}(O) &\iff \exists O' \subset \mathcal{O} : (O \subset O') \wedge (\text{aliases}(O') \in \mathcal{A}) \\ \mathcal{A} \vdash \sim \text{aliases}(O) &\iff \exists O' \subset \mathcal{O} : (O \subset O') \wedge (\sim \text{aliases}(O') \in \mathcal{A}) \end{aligned}$$

1.3 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 alias context is defined structurally as

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

where A is a set of propositions about aliasing and the A_α are the nesting aliasing contexts that correspond to the **then** and **else** branches of conditionals directly nested in ϕ . 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 alias 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 $\mathcal{A}(\phi_{\text{root}})$).

$\mathcal{A}(\phi) \quad := \quad \text{match } \phi \text{ with}$	
v	$\mapsto \langle \emptyset, \emptyset \rangle$
x	$\mapsto \langle \emptyset, \emptyset \rangle$
$e_1 \& e_2$	$\mapsto \mathcal{A}(e_1) \sqcup \mathcal{A}(e_2)$
$e_1 \parallel e_2$	$\mapsto \mathcal{A}(e_1) \sqcap \mathcal{A}(e_2)$
$e_1 \oplus e_2$	$\mapsto \langle \emptyset, \emptyset \rangle$
$o_1 = o_2$	$\mapsto \langle \{\text{aliases } \{o_1, o_2\}\}, \emptyset \rangle$
$o_1 \neq o_2$	$\mapsto \langle \{\sim \text{aliases } \{o_1, o_2\}\}, \emptyset \rangle$
$e_1 \odot e_2$	$\mapsto \langle \emptyset, \emptyset \rangle$
$e.f$	$\mapsto \langle \emptyset, \emptyset \rangle$
$\text{acc}(e.f)$	$\mapsto \langle \emptyset, \emptyset \rangle$
$\phi_1 * \phi_2$	$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\phi_1 \wedge \phi_2$	$\mapsto \mathcal{A}(\phi_1) \sqcup \mathcal{A}(\phi_2)$
$\alpha_C(e_1, \dots, e_k)$	$\mapsto \langle \emptyset, \emptyset \rangle$
if e then ϕ_1 else ϕ_2	$\mapsto \langle \emptyset, \{\mathcal{A}(e) \sqcup \mathcal{A}(\phi_1), (\sim \mathcal{A}(e)) \sqcup \mathcal{A}(\phi_2)\} \rangle$
unfolding $\alpha_C(e_1, \dots, e_k)$ in ϕ'	$\mapsto \mathcal{A}(\phi')$

where the following operations combine alias contexts — context union, \sqcup , and context intersection, \sqcap .

$$\begin{aligned} \langle A_1, \{A_{1\alpha}\} \rangle \sqcup \langle A_2, \{A_{2\alpha}\} \rangle = & \langle \{\text{aliased } \{o' \mid A_1 \cup A_2 \vdash \text{aliased } \{o, o'\}\} \mid o \in \mathcal{O}\} \cup \\ & \{\sim \text{aliased } \{o', \tilde{o}\} \mid \forall o, o', \tilde{o} \in \mathcal{O} : (A_1 \cup A_2 \vdash \text{aliased } \{o, o'\}) \wedge \\ & (A_1 \cup A_2 \vdash \sim \text{aliased } \{o, \tilde{o}\})\}, \\ & \{A_{1\alpha}\} \cup \{A_{2\alpha}\} \rangle \end{aligned}$$

$$\begin{aligned} \langle A_1, \{A_{1\alpha}\} \rangle \sqcap \langle A_2, \{A_{2\alpha}\} \rangle = & \langle \{\text{aliased } \{o' \mid A_1 \cap A_2 \vdash \text{aliased } \{o, o'\}\} \mid o \in \mathcal{O}\} \cup \\ & \{\sim \text{aliased } \{o', \tilde{o}\} \mid \forall o, o', \tilde{o} \in \mathcal{O} : (A_1 \cap A_2 \vdash \text{aliased } \{o, o'\}) \wedge \\ & (A_1 \cap A_2 \vdash \sim \text{aliased } \{o, \tilde{o}\})\}, \\ & \emptyset \rangle \end{aligned}$$

2 Framing

2.1 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 $\alpha_C(\bar{e})$.

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 the permissions it requires.

2.2 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	$\mapsto \top$
		$e_1 \oplus e_2$	$\mapsto \Pi \models_I e_1, e_2$
		$e_1 \odot e_2$	$\mapsto \Pi \models_I e_1, e_2$
		$e.f$	$\mapsto (\Pi \models_I e) \wedge (\Pi \vdash \text{accessed}_\phi(e.f))$
		$\text{acc}(e.f)$	$\mapsto (\Pi \models_I e) \wedge \sim (\Pi \vdash \text{accessed}_\phi(e.f))$
		$\phi_1 * \phi_2$	$\mapsto (\Pi \models_I \phi_1) \wedge (\Pi \cup \text{granted}(\phi_1) \models_I \phi_2)$
		$\phi_1 \wedge \phi_2$	$\mapsto \Pi \models_I \phi_1, \phi_2$
		$\alpha_C(e_1, \dots, e_k)$	$\mapsto \Pi \models_I e_1, \dots, e_k$
		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	$\mapsto \emptyset$
		$\text{acc}(e.f)$	$\mapsto \{\text{accessed}(e.f)\}$
		$\phi_1 * \phi_2$	$\mapsto \text{granted}(\phi_1) \cup \text{granted}(\phi_2)$
		$\phi_1 \wedge \phi_2$	$\mapsto \text{granted}(\phi_1) \cup^\wedge \text{granted}(\phi_2)$
		$\alpha_C(e_1, \dots, e_k)$	$\mapsto \{\text{assumed}(\alpha_C(e_1, \dots, e_k))\}$
		if e then ϕ_1 else ϕ_2	$\mapsto \text{granted}(\phi_1) \cap \text{granted}(\phi_2)$
		unfolding $\alpha_C(e_1, \dots, e_k)$ in ϕ'	$\mapsto \text{granted}(\phi')$
$\text{aliases}_\phi(o)$	$:=$	$\{o' \mid \mathcal{A}(\phi) \vdash \text{aliased} \{o, o'\}\}$	

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

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

2.2.1 Notes

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