Contents

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

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

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

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

1.2 Aliasing Context

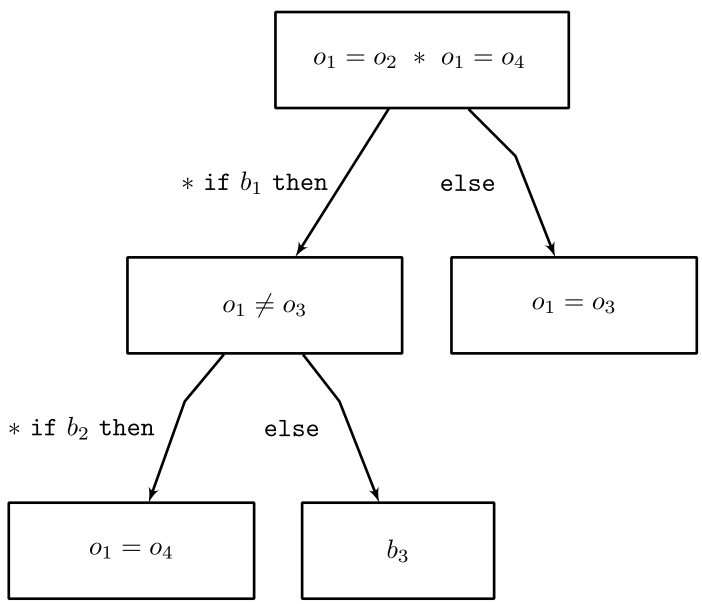
Let ϕ be a formula. The **aliasing context** $\mathcal{A}$ of ϕ is a tree of set of aliasing proposition about aliasing of object variables that appear in ϕ. $\mathcal{A}$ needs to be a tree because the conditional sub-formulas that may appear in ϕ allow for branching aliasing contexts not expressible 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:

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

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.

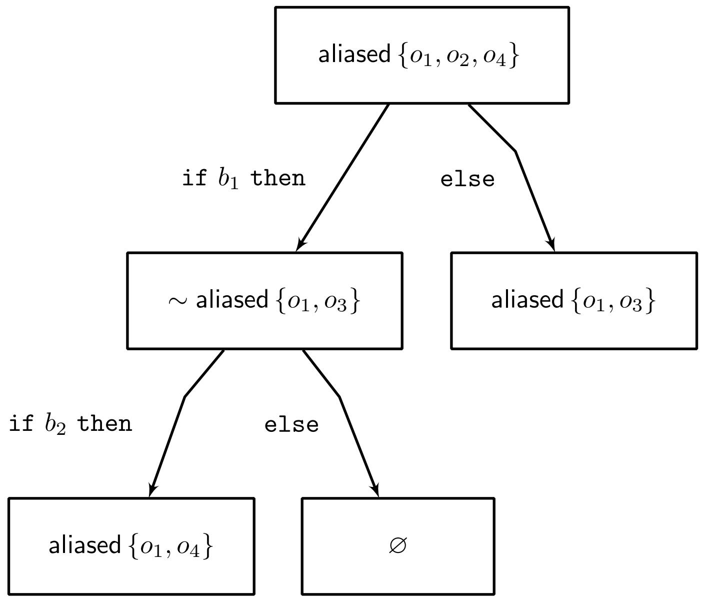


context for the sub-formula $(o_1 = o_4)$ of ϕ is:

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

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

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

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 $A \vdash P$. So, the general aliasing judgments are decided in the following ways for each case:

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

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 $\mathcal{A}(\phi_{\mathsf{root}})$. With the root invariant, the following recursive algorithm constructs $\mathcal{A}(\phi)$ for any sub-formula of ϕ_{root} (including $\mathcal{A}(\phi_{\mathsf{root}})$).

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

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

$$
\begin{aligned}
\langle A_1, \{\mathcal{A}_{1\alpha}\} \rangle \sqcup \langle A_2, \{\mathcal{A}_{2\alpha}\} \rangle = \langle & \{\mathsf{aliased}\,\{o' \mid A_1 \cup A_2 \vdash \mathsf{aliased}\,\{o, o'\}\} \mid o \in \mathcal{O}\} \ \cup \\
& \{\sim \mathsf{aliased}\,\{o', \tilde{o}\} \mid (A_1 \cup A_2 \vdash \mathsf{aliased}\,\{o, o'\}) \wedge \\
& \quad (A_1 \cup A_2 \vdash \sim \mathsf{aliased}\,\{o, \tilde{o}\})\}, \\
& \bigcup \{\mathcal{A}_{1\alpha} \sqcup \mathcal{A}_{2\alpha} \mid \mathcal{A}_{1\alpha}, \mathcal{A}_{2\alpha} \text{ have equivelant conditions}\} \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle A_1, \{\mathcal{A}_{1\alpha}\} \rangle \sqcap \langle A_2, \{\mathcal{A}_{2\alpha}\} \rangle = \langle & \{\mathsf{aliased}\,\{o' \mid A_1 \cap A_2 \vdash \mathsf{aliased}\,\{o, o'\}\} \mid o \in \mathcal{O}\} \ \cup \\
& \{\sim \mathsf{aliased}\,\{o', \tilde{o}\} \mid (A_1 \cap A_2 \vdash \mathsf{aliased}\,\{o, o'\}) \wedge \\
& \quad (A_1 \cap A_2 \vdash \sim \mathsf{aliased}\,\{o, \tilde{o}\})\}, \\
& \bigcup \{\mathcal{A}_{1\alpha} \sqcap \mathcal{A}_{2\alpha} \mid \mathcal{A}_{1\alpha}, \mathcal{A}_{2\alpha} \text{ have equivelant conditions}\} \rangle
\end{aligned}
$$

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 $\mathsf{accessed}(e.f)$.
- to assume $\alpha_C(\overline{e})$, written $\mathsf{assumed}(\alpha_C(\overline{e}))$. This allows the a single unrolling of $\alpha_C(\overline{e})$.

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, $\lfloor \phi \rfloor = \varnothing$.

2.2 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).

$$
\begin{array}{lcll}
\Pi \vDash_I \phi & \iff & \mathsf{match}\ \phi\ \mathsf{with} & \\
 & & v & \mapsto \ \top \\
 & & x & \mapsto \ \top \\
 & & e_1 \oplus e_2 & \mapsto \ \Pi \vDash_I e_1, e_2 \\
 & & e_1 \odot e_2 & \mapsto \ \Pi \vDash_I e_1, e_2 \\
 & & e.f & \mapsto \ (\Pi \vDash_I e) \ \wedge \ (\Pi \vdash \mathsf{accessed}_\phi(e.f)) \\
 & & \mathtt{acc}(e.f) & \mapsto \ (\Pi \vDash_I e) \wedge \ \sim (\Pi \vdash \mathsf{accessed}_\phi(e.f)) \\
 & & \phi_1 * \phi_2 & \mapsto \ (\Pi \cup \mathsf{granted}(\phi_2) \vDash_I \phi_1) \wedge \\
 & & & \quad\ (\Pi \cup \mathsf{granted}(\phi_1) \vDash_I \phi_2) \\
 & & \phi_1 \wedge \phi_2 & \mapsto \ \Pi \vDash_I \phi_1, \phi_2 \\
 & & \alpha_C(e_1, \dots, e_k) & \mapsto \ \Pi \vDash_I e_1, \dots, e_2 \\
 & & \mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2 & \mapsto \ \Pi \vDash_I e, \phi_1, \phi_2 \\
 & & \mathtt{unfolding}\ \alpha_C(\overline{e})\ \mathtt{in}\ \phi' & \mapsto \ (\Pi \vdash \mathsf{assumed}_\phi(\alpha_C(\overline{e}))) \ \wedge \ (\Pi \vDash_I \phi') \\
 & & & \\
\mathsf{granted}(\phi) & := & \mathsf{match}\ \phi\ \mathsf{with} & \\
 & & e & \mapsto \ \varnothing \\
 & & \mathtt{acc}(e.f) & \mapsto \ \{\mathsf{accessed}(e.f)\} \\
 & & \phi_1 * \phi_2 & \mapsto \ \mathsf{granted}(\phi_1) \cup \mathsf{granted}(\phi_2) \\
 & & \phi_1 \wedge \phi_2 & \mapsto \ \mathsf{granted}(\phi_1) \cup^\wedge \mathsf{granted}(\phi_2) \\
 & & \alpha_C(e_1, \dots, e_k) & \mapsto \ \{\mathsf{assumed}(\alpha_C(e_1, \dots, e_k))\} \\
 & & \mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2 & \mapsto \ \mathsf{granted}(\phi_1) \cap \mathsf{granted}(\phi_2) \\
 & & \mathtt{unfolding}\ \alpha_C(e_1, \dots, e_k)\ \mathtt{in}\ \phi' & \mapsto \ \mathsf{granted}(\phi') \\
 & & & \\
\mathsf{aliases}_\phi(o) & := & \{o' \mid \mathcal{A}(\phi) \vdash \mathsf{aliased}\ \{o, o'\}\} &
\end{array}
$$

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

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

2.2.1 Notes

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