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

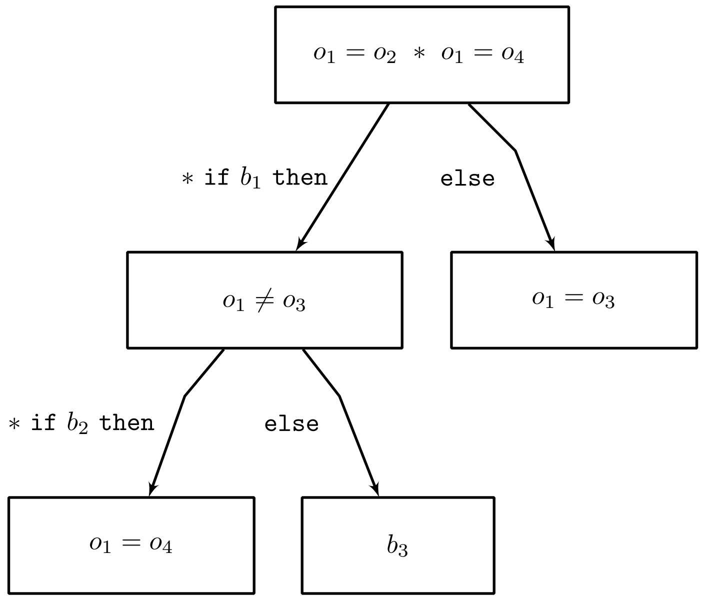
$$\forall o, \tilde{o} : o \neq \tilde{o} \iff \mathsf{non\text{-}aliases}(O)$$

1.2 Aliasing Context

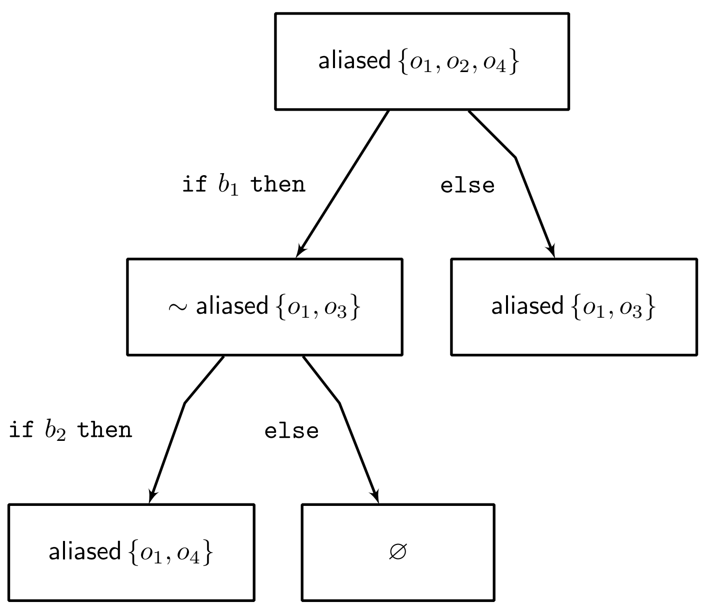
Let ϕ be a formula. The **aliasing context** $\mathcal{A}$ of ϕ is a tree of set of aliasing proposition about aliasing of object variables that appear in ϕ. $\mathcal{A}$ needs to be a tree because the conditional sub-formulas that may appear in ϕ allow for branching aliasing contexts not expressible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

$$\phi := (o_1 = o_2) * (\texttt{if}\ (b_1)\ \texttt{then}\ ((o_1 \neq o_3) * (\texttt{if}\ (b_2)\ \texttt{then}\ (o_1 = o_4)\ \texttt{else}\ (b_3)))\ \texttt{else}\ (o_1 = o_3)) * (o_1 = o_4)$$

ϕ has a formula-structure represented by the following tree:



The formula-structure tree for ϕ corresponds node-for-node and edge-for-edge to the following aliasing context:



where a node inherits all the aliasing assertions of its parents. So for example, the aliasing 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', including inheritance from its place in the aliasing context of ϕ. Usually $\mathcal{A}_\phi(\phi')$ is abbreviated to $\mathcal{A}(\phi')$ when the top level formula ϕ is implicit.

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 the permissions it requires.

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).

$$
\begin{array}{lcll}
\Pi \vDash_I \phi & := & \textsf{match } \phi \textsf{ with} & \\
 & & v & \mapsto \\
 & & x & \mapsto \\
 & & e_1 \&\& e_2 & \mapsto \\
 & & e_1 \mid\mid e_2 & \mapsto \\
 & & e_1 \oplus e_2 & \mapsto \\
 & & x = y & \mapsto \\
 & & e_1.f_1 = e_2.f_2 & \mapsto \\
 & & e.f \neq e'.f' & \mapsto \\
 & & e_1 \odot e_2 & \mapsto \\
 & & e.f & \mapsto \\
 & & \texttt{acc}(e.f) & \mapsto \\
 & & \phi_1 * \phi_2 & \mapsto \\
 & & \phi_1 \wedge \phi_2 & \mapsto \\
 & & \alpha_C(e_1, \ldots, e_k) & \mapsto \\
 & & \texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto \\
 & & \texttt{unfolding } \alpha_C(e_1, \ldots, e_k) \texttt{ in} \phi' & \mapsto
\end{array}
$$