Framing Rules

Henry Blanchette

1 Definitions

Note: in this document "formula" refers to "precise formula," however gradual formulas will eventually be supported.

A **permission** is to either access a field, written $\mathsf{access}(e.f)$, or to assume a predicate holds of its arguments, written $\mathsf{assume}(\alpha_C(\overline{e}))$.

A formula ϕ **requires** a permission π if ϕ contains an access or assumption that π premits. The set of all permissions that ϕ requires (the set of permissions required to frame ϕ) is called the **requirements** of ϕ.

A formula ϕ **grants** permission π if it contains an adjuct that yields π.

A set of permissions Π **frames** a formula ϕ if and only if ϕ requires only permissions contained in Π, written

$$\Pi \vDash_I \phi.$$

The **footprint** of a formula ϕ is the smallest permission mask that frames ϕ, written

$$\lfloor \phi \rfloor.$$

A formula ϕ is **self-framing** if and only if for any set of permissions ϕ, $\Pi \vDash_I \phi$, written

$$\vdash_{\mathsf{frm}I} \phi.$$

In other words, ϕ is self-framing if and only if it grants all the permissions that it requires.

2 Aliasing

The **alias status** of a pair of identifiers x, y is exactly one of the following: $\mathsf{aliases}$, $\mathsf{non\text{-}aliases}$, $\mathsf{undetermined\text{-}aliases}$.

- x, y are $\mathsf{aliases}$ if they refer to the same memory in the heap.
- x, y are $\mathsf{non\text{-}aliases}$ if they do not refer to the same memory in the heap.
- x, y are $\mathsf{?\text{-}aliases}$ if they may or may not refer to the same memory in the heap.

An **alias class** is a pair $[S, I]$ where S is an alias status and I is a set of identifiers where each pair of identifiers in I have alias status S. $[id]$ is the alias class of id where id is an identifier. It is possible to keep track of $\mathsf{?\text{-}aliases}$ classes but instead, for the sake of simplicity, two identifiers are considered $\mathsf{?\text{-}aliases}$ if they are neither asserted as $\mathsf{aliases}$ nor $\mathsf{non\text{-}aliases}$ by an alias class.

A set of alias classes $\{[S_\alpha, I_\alpha]\}$ is **overlapping**, written overlapping($\{[S_\alpha, I_\alpha]\}$), if and only if

$$\bigcap I_\alpha \neq \varnothing.$$

A set of alias statuses $\{S_\alpha\}$ is **compatible** if and only if

$$\forall S \in \{S_\alpha\} : \forall \alpha : S_\alpha = S$$

A set of alias classes A is **compatible** if and only if

$$\forall \{[S_\alpha, I_\alpha]\} \subset A : \{[S_\alpha, I_\alpha]\} \text{ is overlapping} \implies \{S_\alpha\} \text{ is compatible}$$

This is to say that a set of compabile alias classes must not assert that a pair of identifiers are both $\mathsf{aliases}$ and $\mathsf{non\text{-}aliases}$ — every overlapping set of alias classes is compatible.

Two compatible set of alias classes A, A' with a compatible union can be **merged** via the following

$$A \uplus A' := \mathsf{simplify}(A \cup A')$$

where

$$\begin{aligned} \mathsf{simplify}(\{[S_\alpha, I_\alpha]\}) &:= \left\{ \left[S, \bigcap I_{\alpha_i}\right] \;\middle|\; \forall \alpha : \{\alpha_i\} = \mathsf{LOS}(\alpha)) \wedge \forall S_{\alpha_i} : S = S_{\alpha_i} \right\} \\ \mathsf{LOS}(\alpha) &:= \{\alpha_i\}\,, \text{ the largest subset of } \{\alpha\} \text{ such that } \alpha \in \{\alpha_i\} \text{ and } \{I_{\alpha_i}\} \text{ is overlapping} \end{aligned}$$

This simplification combines all overlapping alias classes, where each combination of alias classes results in a compatible alias class because $A \cup A'$ is compatible (as required by $\forall S_{\alpha_i} : S = S_{\alpha_i}$).

The following algorithm accumulates the alias classes for a given formula ϕ within the context set of alias classes A. If at any point of the algorith a merge is attempted on incompatible sets of alias classes, an exception is thrown.

$$\mathsf{alias\text{-}classes}(A, \phi) := \mathsf{match}\ \phi\ \mathsf{with} \left| \begin{array}{lcl} e & \mapsto & \varnothing \\ \mathsf{acc}(e.f) & \mapsto & A \uplus \{[\mathsf{non\text{-}aliases}, \{e.f\}]\} \\ \phi & \mapsto & \varnothing \end{array} \right.$$

3 Deciding Framing

The following algorithm decides $\Pi \vDash_I \phi$ for a given set of permissions Π and formula ϕ.

$$
\Pi \vDash_I \phi \iff \textsf{match } \phi \textsf{ with } \left|
\begin{array}{lcl}
v, x & \mapsto & \top \\
e_1 \oplus e_2 & \mapsto & \Pi \vDash_I e_1, e_2 \\
e_1 \odot e_2 & \mapsto & \Pi \vDash_I e_1, e_2 \\
e.f & \mapsto & \Pi \vDash_I e \land \texttt{acc}(e.f) \in \Pi \\
\texttt{acc}(e.f) & \mapsto & \Pi \vDash_I e \\
\phi_1 \circledast \phi_2 & \mapsto & \Pi \cup \textsf{granted}(\phi_1 \circledast \phi_2) \vDash_I \phi_1, \phi_2 \\
\alpha_C(\overline{e}) & \mapsto & \Pi \vDash_I \overline{e} \\
\texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto & \Pi \vDash_I e, \phi_1, \phi_2 \\
\texttt{unfolding } \alpha_C(\overline{e}) \texttt{ in } \phi & \mapsto & \textsf{assume}(\alpha_C(\overline{e})) \in \Pi \land \Pi \vDash_I \alpha_C(\overline{e}) \land \Pi \vDash_I \phi
\end{array}
\right.
$$

The following algorithm produces the set of permissions granted by a given formula ϕ.

$$
\textsf{granted}(\phi) := \textsf{match } \phi \textsf{ with } \left|
\begin{array}{lcl}
e & \mapsto & \varnothing \\
\texttt{acc}(e.f) & \mapsto & \{\textsf{access}(e.f)\} \\
\phi_1 \circledast \phi_2 & \mapsto & \textsf{granted}(\phi_1) \cup \textsf{granted}(\phi_2) \\
\alpha_C(\overline{e}) & \mapsto & \{\textsf{assume}(\alpha_C(\overline{e}))\} \\
\texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto & \textsf{granted}(\phi_1) \cap \textsf{granted}(\phi_2) \\
\texttt{unfolding } \alpha_C(\overline{e}) \texttt{ in } \phi & \mapsto & \textsf{granted}(\phi)
\end{array}
\right.
$$

3.1 Notes

- The conditional expression e in a formula of the form ($\texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2$) is considered indeteminant for the purposes of statically deciding framing.
- The body formula ϕ in a formula of the form ($\texttt{unfolding acc}_C(\overline{e}) \texttt{ in } \phi$) does not have to make use of the $\textsf{assume}(\texttt{acc}_C(\overline{e}))$ required by the structure.

4 Deciding Self-Framing

The following algorithm decides $\vdash_{\mathsf{frm}I} \phi$ for a given formula ϕ.

$$\vdash_{\mathsf{frm}I} \phi \iff \varnothing \vDash_I \phi$$