# 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 $\phi$ **requires** a permission $\pi$ if $\phi$ contains an access or assumption that $\pi$ premits. The set of all permissions that $\phi$ requires (the set of permissions required to frame $\phi$) is called the **requirements** of $\phi$.

A formula $\phi$ **grants** permission $\pi$ if it contains an adjuct that yields $\pi$.

A set of permissions $\Pi$ **frames** a formula $\phi$ if and only if $\phi$ requires only permissions contained in $\Pi$, written

$$\Pi \vDash_I \phi.$$

The **footprint** of a formula $\phi$ is the smallest permission mask that frames $\phi$, written

$$\lfloor \phi \rfloor.$$

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

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

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

## 2 Deciding Framing without Aliasing

For this section framing desicions do not consider aliasing, for the sake of an introduction.

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

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

The following algorithm collects the set of permissions granted by a given formula $\phi$.

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

### 2.1 Notes

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

### 2.2 Deciding Self-Framing without Aliasing

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

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

# 3 Aliasing

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

- The $x_\alpha$ are $\mathsf{aliases}$ if each $x_\alpha$ refers to the same memory in the heap.
- The $x_\alpha$ are $\mathsf{non\text{-}aliases}$ if each $x_\alpha$ refers to distinct memory in the heap.
- The $x_\alpha$ are $\mathsf{undermined\text{-}aliases}$ if they may be $\mathsf{aliases}$ or $\mathsf{non\text{-}aliases}$.

An **alias class** is a pair $[S, I]$ where $S$ is an alias status and $I$ is a set of identifiers where the identifiers of $I$ have alias status $S$. $\mathsf{identifiers}(\{[S_\alpha, I_\alpha]\}) := \bigcup I_\alpha$ is the set of identifiers of a set of alias classes. It is possible to keep track of $\mathsf{undermined\text{-}aliases}$ classes. However, for the sake of efficiency, some give identifiers are considered $\mathsf{undermined\text{-}aliases}$ if no subset of them are asserted as $\mathsf{aliases}$ nor $\mathsf{non\text{-}aliases}$ by any alias class.

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

$$\bigcup I_\alpha \neq \varnothing.$$

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.

Given two compatible sets of alias classes $A, A'$, the compatibility of $A \cup A'$ can be considered, written $A \uplus A'$. Deciding $A \uplus A'$ reduces to computing $\mathsf{simplify}(A \cup A')$ which either preserves compatibility or raises an exception, where

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

For each set of overlapping alias classes, $\mathsf{simplify}$ either combines then or throws an exception.

### 3.1 Deciding Alias Class Compatibility

A set of alias classes $A$ is **compatible** with a formula $\phi$ if and only if $A$ is compatible with the aliasing assertions yielded by $\phi$, written $A \uplus \phi$. The following algorithm decides $A \uplus \phi$.

$$
A \uplus \phi \iff \textsf{match } \phi \textsf{ with} \left|
\begin{array}{lll}
e & \mapsto & A \uplus \textsf{asserted}(e) \\
\texttt{acc}(e.f) & \mapsto & A \uplus \{[\textsf{non-aliases}, \{e\} \cup \textsf{identifiers}(A)]\} \\
\phi_1 \circledast \phi_2 & \mapsto & A \uplus \textsf{asserted}(\phi_1) \uplus \textsf{asserted}(\phi_2) \\
\phi_1 \wedge \phi_2 & \mapsto & (A \uplus \phi_1) \wedge (A \uplus \phi_2) \\
\texttt{if } e \texttt{ then } \phi_1 \texttt{ else } \phi_2 & \mapsto & \textsf{if } A \uplus \textsf{asserted}(e) \\
 & & \quad \textsf{then } A \uplus \textsf{asserted}(e) \uplus \textsf{asserted}(\phi_1) \\
 & & \quad \textsf{else } A \uplus \neg\, \textsf{asserted}(e) \uplus \textsf{asserted}(\phi_2) \\
\phi & \mapsto & A \uplus \textsf{asserted}(\phi)
\end{array}
\right.
$$

where

$$
\begin{aligned}
\neg A &:= \{[\neg S_\alpha, I_\alpha] \mid [S_\alpha, I_\alpha] \in A\}\,, \\
\neg\, \textsf{aliases} &:= \textsf{non-aliases}, \\
\neg\, \textsf{non-aliases} &:= \textsf{aliases}.
\end{aligned}
$$

The following algorithm collects the set of alias classes asserted by a given formula $\phi$.

$$
\textsf{asserted}(\phi) := \textsf{match } \phi \textsf{ with} \left|
\begin{array}{lll}
x = y & \mapsto & \{[\textsf{aliases}, \{x, y\}]\} \\
x \odot y & \mapsto & \{[\textsf{non-aliases}, \{e\}]\} \\
e & \mapsto & \varnothing \\
\texttt{unfolding } \alpha(\overline{e}) \texttt{ in } \phi & \mapsto & A \uplus \phi
\end{array}
\right.
$$

The following algorithm collects the set of identifiers in a given set of alias classes $A$.

$$
\textsf{identifiers}(\{[S_\alpha, I_\alpha]\}) := \bigcup I_\alpha
$$

# 4 Deciding Framing with Aliasing

For this section framing desicions *do* consider aliasing. A set of permissions $\Pi$ and a set of alias classes $A$ frames a formula $\phi$ if and only if $\phi$ requires only permission contained in $\Pi$ and $\mathsf{alias\text{-}classes}(A, \phi)$ is compatible.

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

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

The following algorithm collects the set of permissions granted by a given formula $\phi$.

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

## 4.1 Notes

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

## 4.2 Deciding Self-Framing with Aliasing

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

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