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 Framing without Aliasing

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

2.1 Deciding Framing without Aliasing

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

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

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

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

2.2 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 ($\texttt{unfolding}\ \texttt{acc}_C(\overline{e})\ \texttt{in}\ \phi$) does not have to make use of the $\mathsf{assume}(\texttt{acc}_C(\overline{e}))$ required by the structure.

2.3 Deciding Self-Framing without Aliasing

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

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

3 Aliasing (New)

Let I be the set of identifiers. A pair of identifiers $x, y \in I$ are **unique** if they are not the same identifier. The proposition that x, y are unique is written $\mathsf{unique}(x, y)$ (note that this is importantly different notation from $x = y$). The proposition that two identifiers refer to the same memory in the heap is written $x = y$. A set of identifiers $\{x_\alpha\}$ is **aliasing** if and and only if each x_α refers to the same memory in the heap i.e.

$$x_{\alpha_1} = \cdots = x_{\alpha_k} \text{ where } \{\alpha\} = \{\alpha_1, \ldots, \alpha_k\}.$$

The proposition that $\{x_\alpha\}$ is aliasing is written $\mathsf{aliasing}\,\{x_\alpha\}$.

An **aliasing context** is a set A of aliasing propositions. As a set of propositions, the consistency of A can be considered. Explicitly, A is consistent if and only if

$$\nexists x, y \in I : \mathsf{unique}(x, y) \wedge A \vdash \mathsf{aliasing}\,\{x, y\} \wedge \sim \mathsf{aliasing}\,\{x, y\}$$

The proposition that A is consistent is written $\mathsf{consistent}(A)$.

An alias context is **overlapping** if and only if there exist at least two unique sets of identifiers such that they have a non-empty intersection and both are asserted aliasing in A i.e.

$$\exists I_1, I_2 \subset I : (I_1 \neq I_2) \wedge (I_1 \cap I_2 \neq \varnothing) \wedge (\mathsf{aliasing}(I_1) \in A) \wedge (\mathsf{aliasing}(I_2) \in A)$$

An overlapping alias context is inefficient for deciding the propositions that it entails. Fortunately the framing-deciding algorthm I present ensures that its tracked alias context never becomes becomes overlapping.

A alias context A is **full** if and only if

$$\forall I_\alpha \subset I : A \vdash P(I_\alpha) \implies \exists\ P(I_{\alpha'}) \in A : I_\alpha \subset I_{\alpha'}$$

where P is an aliasing predicate (either $\mathsf{aliasing}$ or $\sim \mathsf{aliasing}$). In other words, a full alias context is the most efficient representation of its total propositional strength. This is useful for efficient computation, as demonstrated in the following.

Given A an alias context and $x \in I$ an identifier, define:

$$\begin{aligned} \mathsf{aliases\text{-}of}(x) &:= \text{the largest set such that } x \in \mathsf{aliases\text{-}of}(x) \wedge A \vdash \mathsf{aliasing}(\mathsf{aliases\text{-}of}(x)) \\ \mathsf{not\text{-}aliases\text{-}of}(x) &:= \text{the largest set such that } \forall x' \in \mathsf{not\text{-}aliases\text{-}of}(x) : A \vdash \sim \mathsf{aliasing}\,\{x, y'\} \end{aligned}$$

If A is non-overlapping and full, the computation of $\mathsf{aliases\text{-}of}(x)$ is simply the extraction from A the proposition that asserts aliasing of a set of identifiers that contains x and the computation of $\mathsf{not\text{-}aliases\text{-}of}(x)$ is the collection of all identifiers other than x mentioned in propositions of A that assert the negation of aliasing with x. For example,

$$A := \{\mathsf{aliasing}\,\{x, y\}, \mathsf{aliasing}\,\{z\}, \sim \mathsf{aliasing}\,\{x, z\}, \sim \mathsf{aliasing}\,\{y, z\}\}$$

id	aliases-of(id)	not-aliases-of(id)
x	$\{x, y\}$	$\{z\}$
y	$\{x, y\}$	$\{z\}$
z	$\{z\}$	$\{x, y\}$

4 Aliasing (Old)

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_α are $\mathsf{aliases}$ if each x_α refers to the same memory in the heap.
- The x_α are $\mathsf{non\text{-}aliases}$ if each x_α refers to distinct memory in the heap.
- The x_α 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

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

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

4.1 Deciding Alias Class Compatibility

A set of alias classes A is **compatible** with a formula ϕ if and only if A is compatible with the aliasing assertions yielded by ϕ, 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 ϕ.

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

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

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

5 Framing with Aliasing

For this section framing desicions *do* consider aliasing.

5.1 Deciding Framing with Aliasing

5.2 Notes

5.3 Deciding Self-Framing with Aliasing

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