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.

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

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

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

2.2 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 ϕ 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.

2.3 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

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. Note that, of course, a full alias context that contains $\mathsf{aliasing}\,\{x\}$ need not contain $\sim\sim \mathsf{aliasing}\,\{x\}, \sim\sim\sim\sim \mathsf{aliasing}\,\{x\}, \ldots$ although it does indeed entail these propositions. 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 Framing with Aliasing

For this section framing desicions *do* consider aliasing.

4.1 New Permissions

A particular innacuracy of the framing without aliasing approach was the handling of separate access permissions to the heap. In order to incorporate aliasing alongside heap access, we introduce two new permissions:

- $\mathsf{aliased}(\{x_\alpha\})$ is the permission to assume that each x_α is an alias of each other x_α.
- $\mathsf{accessed}(e.f)$ is the permission to assume that $\mathtt{acc}(e.f)$ has, separately, been asserted.

In the next section the rules for requiring and granting these permissions are detailed. The idea is that $\mathtt{acc}(e.f)$ formulas will require that the permission context entails that it is possible that $\sim \mathsf{accessed}(e.f)$, i.e. it is possible that $e.f$ hasn't already been accessed separately. Then $\mathtt{acc}(e.f)$ grants $\mathsf{accessed}(e.f)$ along with $\mathsf{accessed}(e'.f)$ for any aliases of what's in e's place.

In addition to keeping track of aliased accesses, the idea for involving $\mathsf{aliased}(\{x_\alpha\})$ in the permissions is that assertions of aliasing and non-aliasing in expressions (i.e. $x = y \wedge x \neq y$) can be considered contradictions statically.

4.2 Deciding Framing with Aliasing

Given Π a permission set and ϕ a formula, the proposition that Π **frames** ϕ is written

$$\Pi \vDash_I \phi$$

The following algorithm decides $\Pi \vDash_I \phi$.

$$\begin{array}{lll}
\Pi \vDash_I \phi \iff & \mathsf{match}\ \phi\ \mathsf{with} & \\
& v & \mapsto \top \\
& x & \mapsto \top \\
& x = y & \mapsto \Pi \vdash\sim (\sim \mathsf{aliased}\,\{x, y\}) \\
& x \odot y & \mapsto \Pi \vdash\sim (\sim (\sim \mathsf{aliased}\,\{x, y\})) \\
& e_1 \oplus e_2 & \mapsto (\Pi \sqcup \mathsf{granted}_\Pi(e_1) \vDash_I e_1) \wedge (\Pi \sqcup \mathsf{granted}_\Pi(e_2) \vDash_I e_2) \\
& e_1 \&\& e_2 & \mapsto (\Pi \sqcup \mathsf{granted}_\Pi(e_1) \vDash_I e_1) \wedge (\Pi \sqcup \mathsf{granted}_\Pi(e_2) \vDash_I e_2) \\
& e_2 \mid\mid e_2 & \mapsto (\Pi \sqcup \mathsf{granted}_\Pi(e_1) \vDash_I e_1) \vee (\Pi \sqcup \mathsf{granted}_\Pi(e_2) \vDash_I e_2) \\
& e_2 \odot e_2 & \mapsto (\Pi \sqcup \mathsf{granted}_\Pi(e_1) \vDash_I e_1) \wedge (\Pi \sqcup \mathsf{granted}_\Pi(e_2) \vDash_I e_2) \\
& e.f & \mapsto (\Pi \vDash_I e) \wedge (\Pi \vdash \mathsf{accessed}(e.f)) \\
& \mathtt{acc}(e.f) & \mapsto (\Pi \sqcup \mathsf{granted}(\mathtt{acc}(e.f)) \vDash_I e.f) \wedge (\Pi \vdash\sim (\sim (\sim \mathsf{accessed}(e.f)))) \\
& \phi_1 \circledast \phi_2 & \mapsto \Pi \sqcup \mathsf{granted}_\Pi(\phi_1 \circledast \phi_2) \vDash_I \phi_1, \phi_2 \\
& \alpha_C(e_1, \ldots, e_k) & \mapsto \Pi \sqcup\ \vDash_I e_1, \ldots, e_k \\
& \mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2 & \mapsto (\Pi \vDash_I e) \wedge (\Pi \sqcup \mathsf{granted}_\Pi(e) \vDash_I \phi_1) \\
& & \quad \wedge (\Pi \sqcup \mathsf{not\text{-}granted}_\Pi(e) \vDash_I \phi_2) \\
& \mathtt{unfolding}\ \alpha_C(\overline{e})\ \mathtt{in}\ \phi' & \mapsto (\Pi \vDash_I \alpha_C(\overline{e})) \wedge (\Pi \sqcup \mathsf{granted}(\alpha_C(\overline{e})) \vDash_I \phi')
\end{array}$$

$$
\begin{array}{lllll}
\mathsf{granted}_\Pi(\phi) & := & \mathsf{match}\ \phi\ \mathsf{with} & & \\
 & & v & \mapsto & \varnothing \\
 & & x & \mapsto & \varnothing \\
 & & x = y & \mapsto & \{\mathsf{aliased}(\mathsf{aliases}_\Pi(x) \cup \mathsf{aliases}_\Pi(y))\} \\
 & & & & \quad \sqcup\ \{\sim \mathsf{aliased}\ \{x', \tilde{y}\} \mid x' \in \mathsf{aliases}_\Pi(x), \tilde{y} \in \mathsf{non\text{-}aliases}_\Pi(y)\} \\
 & & & & \quad \sqcup\ \{\sim \mathsf{aliased}\ \{\tilde{x}, y'\} \mid \tilde{x} \in \mathsf{non\text{-}aliases}_\Pi(x), y' \in \mathsf{aliases}_\Pi(y)\} \\
 & & x \oplus y & \mapsto & \{\sim \mathsf{aliased}(\{x'\} \cup \mathsf{aliases}_\Pi(y)) \mid x' \in \mathsf{aliases}_\Pi(x)\} \\
 & & & & \quad \sqcup\ \{\sim \mathsf{aliased}(\{y'\} \cup \mathsf{aliases}_\Pi(x)) \mid y' \in \mathsf{aliases}_\Pi(x)\} \\
 & & e_1 \oplus e_2 & \mapsto & \\
 & & & & \\
 & & e_2 \odot e_2 & \mapsto & \\
 & & \mathtt{acc}(x.f) & \mapsto & \{\mathsf{accessed}(x'.f) \mid x' \in \mathsf{aliases}_\Pi(x)\} \\
 & & \phi_1 * \phi_2 & \mapsto & \mathsf{granted}_\Pi(\phi_1)\ \sqcup^*\ \mathsf{granted}_\Pi(\phi_2) \\
 & & \phi_1 \wedge \phi_2 & \mapsto & \mathsf{granted}_\Pi(\phi_1)\ \sqcup^\wedge\ \mathsf{granted}_\Pi(\phi_2) \\
 & & \alpha_C(\overline{e}) & \mapsto & \{\mathsf{assumed}(\alpha_C(\overline{e}))\} \\
 & & \mathtt{if}\ e\ \mathtt{then}\ \phi_1\ \mathtt{else}\ \phi_2 & \mapsto & (\mathsf{granted}_\Pi(e) \sqcup \mathsf{granted}_\Pi(\phi_1))\ \sqcap\ (\mathsf{not\text{-}granted}_\Pi(e)\ \sqcup\ \mathsf{granted}_\Pi(\phi_2)) \\
 & & \mathtt{unfolding}\ \alpha_C(\overline{e})\ \mathtt{in}\ \phi' & \mapsto & \mathsf{granted}_\Pi(\alpha_C(\overline{e}))\ \sqcup\ \mathsf{granted}_\Pi(\phi')
\end{array}
$$

$$\mathsf{not\text{-}granted}_\Pi(\phi) := \{\sim \pi \mid \pi \in \mathsf{granted}_\Pi(\phi)\}$$

4.3 Notes

- Let P be a proposition. Then, $\sim (\sim P)$ is informally read as "it is possible that P". Likewise, $\sim (\sim (\sim P))$ is informally read as "it is possible that $\sim P$".
- $\sqcup^\wedge$ allows for overlapping $\mathsf{accessed}(e.f)$ permissions. The branches are not necessarily working on the heap separately, so its ok.
- $\sqcup^*$ does not allow for overrlapping $\mathsf{accessed}(e.f)$ permissions. The branches must work on the heap separately, so such permissions conflict.

4.4 Deciding Self-Framing with Aliasing

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

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