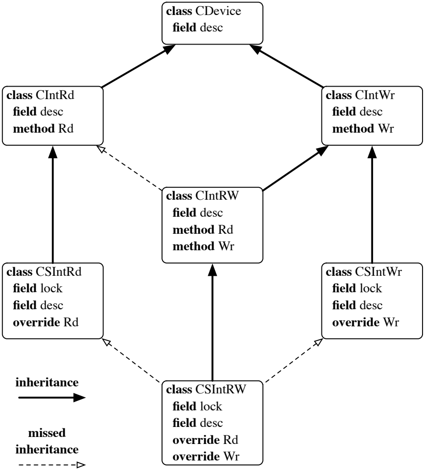
A foundation for trait-based metaprogramming

Aaron Turon

Department of Computer Science
University of Chicago

Joint work with John Reppy

A problem for single inheritance:



What are traits?

A *trait* is a partial class implementation: a flat collection of *provided* methods.

- Methods invoked by a trait but not provided are *required* methods.
- Traits cannot introduce state – they can only provide methods.

Introduced in [Schärli *et al.*; ECOOP'03].

What are traits?

A *trait* is a partial class implementation: a flat collection of *provided* methods.

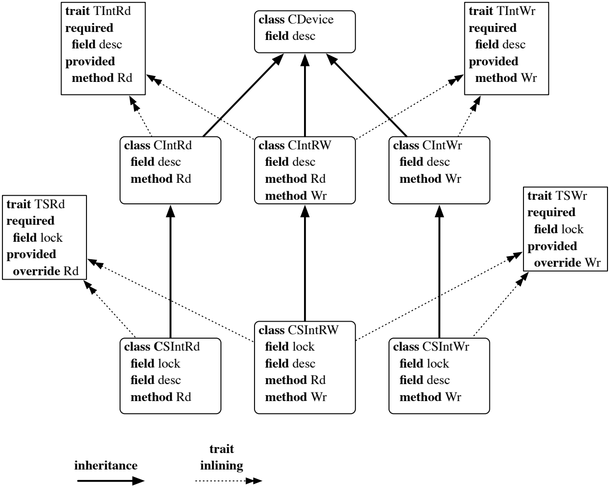
- Methods invoked by a trait but not provided are *required* methods.
- Traits cannot introduce state – they can only provide methods.

Introduced in [Schärli *et al.*; ECOOP'03].

Key idea

Trait *composition* occurs outside the inheritance hierarchy.
Composition is symmetric.

A trait-based solution:



Trait operations

Besides composition, traits support *aliasing* and *excluding* methods to resolve conflicts.

Trait operations

Besides composition, traits support *aliasing* and *excluding* methods to resolve conflicts.

Exclude and compose

TCPoint = TPoint + (TColored **exclude** toString)

Trait operations

Besides composition, traits support *aliasing* and *excluding* methods to resolve conflicts.

Exclude and compose

TCPoint = TPoint + (TColored **exclude** toString)

Alias, exclude and compose

TCPoint = {
  provides toString() : string {
    self.strP() + ": " + **self**.strC();
  }
} + ((TPoint **alias** toString **as** strP) **exclude** toString)
+ ((TColored **alias** toString **as** strC) **exclude** toString)

Deep operations

The **alias** and **exclude** operations are *shallow*: they do not affect the bodies of other methods in the trait.

Deep aliasing $\Longrightarrow$ renaming   Deep exclusion $\Longrightarrow$ hiding

Deep operations

The **alias** and **exclude** operations are *shallow*: they do not affect the bodies of other methods in the trait.

Deep aliasing $\Longrightarrow$ renaming   Deep exclusion $\Longrightarrow$ hiding

Hide and compose

TCPoint = TPoint + (TColored **hide** toString)

Deep operations

The **alias** and **exclude** operations are *shallow*: they do not affect the bodies of other methods in the trait.

Deep aliasing $\Longrightarrow$ renaming   Deep exclusion $\Longrightarrow$ hiding

Hide and compose

TCPoint = TPoint + (TColored **hide** toString)

Rename and compose

TCPoint = {
    provides toString() : string {
        self.strP() + ": " + **self**.strC();
    }
  } + (TPoint **rename** toString **to** strP)
   + (TColored **rename** toString **to** strC)

Our research: develop a statically-typed trait calculus giving a semantics for method hiding and renaming.

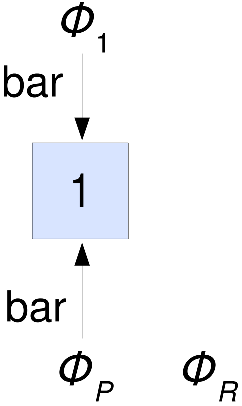
- Built on the Fisher-Reppy polymorphic trait calculus [Fisher & Reppy 2003].
- Uses Riecke-Stone dictionaries [Riecke & Stone 2002] to provide a realistic model of the deep operations.
- Provides more accurate trait types (requirements are tracked *per-method*, rather than per-trait).

Syntactic forms for runtime trait values

$$
\begin{array}{rcll}
\phi & ::= & \{r \mapsto i \ ^{r \in \mathcal{R}}\} & \text{dictionary} \\
Mv & ::= & \{i \mapsto \mu v_i \ ^{i \in \mathcal{I}}\} & \text{method suite value} \\
\mu v & ::= & [E;\ \phi;\ \lambda x.e] & \text{method value} \\
tv & ::= & \langle\!| \ Mv;\ \phi_P;\ \phi_R \ |\!\rangle & \text{trait value}
\end{array}
$$

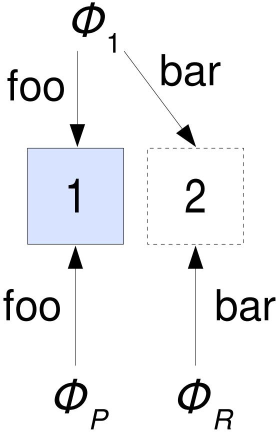
where r ranges over trait method requirements (both self and super) and i ranges over slots.

Note: these are simplified versions of the forms in the paper.



TBar provides bar.

$$\phi_P = \{\text{bar} \mapsto 1\}$$
$$\phi_R = \{\}$$
$$\phi_1 = \{\text{bar} \mapsto 1\}$$



TFoo provides foo, requires bar.

$$\phi_P = \{\text{foo} \mapsto 1\}$$
$$\phi_R = \{\text{bar} \mapsto 2\}$$
$$\phi_1 = \{\text{foo} \mapsto 1,\ \text{bar} \mapsto 2\}$$

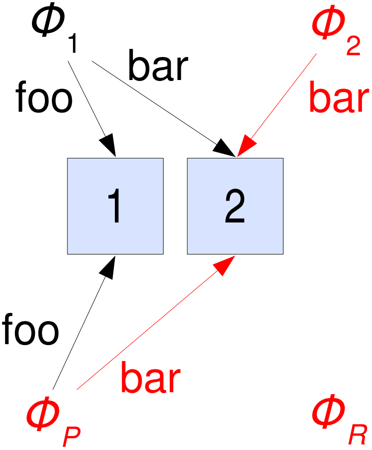
$$
\frac{
\begin{array}{c}
E \vdash T_1 \longrightarrow \llparenthesis\ {}^{\mathcal{I}_1}Mv_1;\ {}^{\mathcal{M}_1}\phi_{P_1};\ {}^{\mathcal{R}_1}\phi_{R_1}\ \rrparenthesis \\
E \vdash T_2 \longrightarrow \llparenthesis\ {}^{\mathcal{I}_2}Mv_2;\ {}^{\mathcal{M}_2}\phi_{P_2};\ {}^{\mathcal{R}_2}\phi_{R_2}\ \rrparenthesis \\
\mathcal{M}_1 \pitchfork \mathcal{M}_2 \quad Mv_2 = \{i \mapsto [E_i;\ \phi_i;\ e_i]^{\ i \in \mathcal{I}_2}\} \\
\varphi_P = \{\phi_{P_2}(m) \mapsto \phi_{R_1}(m)^{\ m \in \mathcal{M}_2 \cap \mathcal{R}_1}\} \\
\varphi_R = \{\phi_{R_2}(m) \mapsto \phi_{P_1}(m)^{\ m \in \mathcal{R}_2 \cap \mathcal{M}_1}\} \\
\mathcal{I}_1' = \mathcal{I}_1 \cup \mathrm{rng}(\phi_{R_1}) \quad \mathcal{I}_2' = \mathcal{I}_2 \cup \mathrm{rng}(\phi_{R_2}) \\
\varphi_F = \mathrm{FS}(\mathcal{I}_2' \setminus \mathrm{dom}(\varphi_R \cup \varphi_P),\ \mathcal{I}_1') \\
\varphi = \varphi_P \cup \varphi_R \cup \varphi_F \quad \phi_P = \phi_{P_1} \cup (\varphi \circ \phi_{P_2}) \\
\phi_R = (\phi_{R_1} \cup (\varphi \circ \phi_{R_2})) \setminus (\mathcal{M}_1 \cup \mathcal{M}_2) \\
Mv = Mv_1 \cup \{\varphi(i) \mapsto [E_i;\ \varphi \circ \phi_i;\ e_i]^{\ i \in \mathcal{I}_2}\}
\end{array}
}{
E \vdash T_1 + T_2 \longrightarrow \llparenthesis\ Mv;\ \phi_P;\ \phi_R\ \rrparenthesis
}
$$

$$\frac{\begin{array}{c} E \vdash T_1 \longrightarrow \langle\!|\ {}^{\mathcal{I}_1}Mv_1;\ {}^{\mathcal{M}_1}\phi_{P_1};\ {}^{\mathcal{R}_1}\phi_{R_1}\ |\!\rangle \\ E \vdash T_2 \longrightarrow \langle\!|\ {}^{\mathcal{I}_2}Mv_2;\ {}^{\mathcal{M}_2}\phi_{P_2};\ {}^{\mathcal{R}_2}\phi_{R_2}\ |\!\rangle \\ \mathcal{M}_1 \pitchfork \mathcal{M}_2 \quad Mv_2 = \{i \mapsto [E_i;\ \phi_i;\ e_i]^{\ i \in \mathcal{I}_2}\} \\ \varphi_P = \{\phi_{P_2}(m) \mapsto \phi_{R_1}(m)^{\ m \in \mathcal{M}_2 \cap \mathcal{R}_1}\} \\ \varphi_R = \{\phi_{R_2}(m) \mapsto \phi_{P_1}(m)^{\ m \in \mathcal{R}_2 \cap \mathcal{M}_1}\} \\ \mathcal{I}_1' = \mathcal{I}_1 \cup \mathrm{rng}(\phi_{R_1}) \quad \mathcal{I}_2' = \mathcal{I}_2 \cup \mathrm{rng}(\phi_{R_2}) \\ \varphi_F = \mathrm{FS}(\mathcal{I}_2' \setminus \mathrm{dom}(\varphi_R \cup \varphi_P),\ \mathcal{I}_1') \\ \varphi = \varphi_P \cup \varphi_R \cup \varphi_F \quad \phi_P = \phi_{P_1} \cup (\varphi \circ \phi_{P_2}) \\ \phi_R = (\phi_{R_1} \cup (\varphi \circ \phi_{R_2})) \setminus (\mathcal{M}_1 \cup \mathcal{M}_2) \\ Mv = Mv_1 \cup \{\varphi(i) \mapsto [E_i;\ \varphi \circ \phi_i;\ e_i]^{\ i \in \mathcal{I}_2}\} \end{array}}{E \vdash T_1 + T_2 \longrightarrow \langle\!|\ Mv;\ \phi_P;\ \phi_R\ |\!\rangle}$$

$$
\frac{
\begin{array}{c}
E \vdash T_1 \longrightarrow \langle\!\!\langle\ {}^{\mathcal{I}_1}Mv_1;\ {}^{\mathcal{M}_1}\phi_{P_1};\ {}^{\mathcal{R}_1}\phi_{R_1}\ \rangle\!\!\rangle \\
E \vdash T_2 \longrightarrow \langle\!\!\langle\ {}^{\mathcal{I}_2}Mv_2;\ {}^{\mathcal{M}_2}\phi_{P_2};\ {}^{\mathcal{R}_2}\phi_{R_2}\ \rangle\!\!\rangle \\
\mathcal{M}_1 \pitchfork \mathcal{M}_2 \quad Mv_2 = \{i \mapsto [E_i;\ \phi_i;\ e_i]\ ^{i \in \mathcal{I}_2}\} \\
\varphi_P = \{\phi_{P_2}(m) \mapsto \phi_{R_1}(m)\ ^{m \in \mathcal{M}_2 \cap \mathcal{R}_1}\} \\
\varphi_R = \{\phi_{R_2}(m) \mapsto \phi_{P_1}(m)\ ^{m \in \mathcal{R}_2 \cap \mathcal{M}_1}\} \\
\mathcal{I}_1' = \mathcal{I}_1 \cup \mathrm{rng}(\phi_{R_1}) \quad \mathcal{I}_2' = \mathcal{I}_2 \cup \mathrm{rng}(\phi_{R_2}) \\
\varphi_F = \mathrm{FS}(\mathcal{I}_2' \setminus \mathrm{dom}(\varphi_R \cup \varphi_P),\ \mathcal{I}_1') \\
\varphi = \varphi_P \cup \varphi_R \cup \varphi_F \quad \phi_P = \phi_{P_1} \cup (\varphi \circ \phi_{P_2}) \\
\phi_R = (\phi_{R_1} \cup (\varphi \circ \phi_{R_2})) \setminus (\mathcal{M}_1 \cup \mathcal{M}_2) \\
Mv = Mv_1 \cup \{\varphi(i) \mapsto [E_i;\ \varphi \circ \phi_i;\ e_i]\ ^{i \in \mathcal{I}_2}\}
\end{array}
}{
E \vdash T_1 + T_2 \longrightarrow \langle\!\!\langle\ Mv;\ \phi_P;\ \phi_R\ \rangle\!\!\rangle
}
$$

$$\frac{\begin{array}{c}
E \vdash T_1 \longrightarrow \langle\!|\ {}^{\mathcal{I}_1}Mv_1;\ {}^{\mathcal{M}_1}\phi_{P_1};\ {}^{\mathcal{R}_1}\phi_{R_1}\ |\!\rangle \\
E \vdash T_2 \longrightarrow \langle\!|\ {}^{\mathcal{I}_2}Mv_2;\ {}^{\mathcal{M}_2}\phi_{P_2};\ {}^{\mathcal{R}_2}\phi_{R_2}\ |\!\rangle \\
\mathcal{M}_1 \pitchfork \mathcal{M}_2 \quad Mv_2 = \{i \mapsto [E_i;\ \phi_i;\ e_i]^{\ i\in\mathcal{I}_2}\} \\
\varphi_P = \{\phi_{P_2}(m) \mapsto \phi_{R_1}(m)^{\ m\in\mathcal{M}_2\cap\mathcal{R}_1}\} \\
\varphi_R = \{\phi_{R_2}(m) \mapsto \phi_{P_1}(m)^{\ m\in\mathcal{R}_2\cap\mathcal{M}_1}\} \\
\mathcal{I}_1' = \mathcal{I}_1 \cup \mathrm{rng}(\phi_{R_1}) \quad \mathcal{I}_2' = \mathcal{I}_2 \cup \mathrm{rng}(\phi_{R_2}) \\
\varphi_F = \mathrm{FS}(\mathcal{I}_2' \setminus \mathrm{dom}(\varphi_R \cup \varphi_P),\ \mathcal{I}_1') \\
\color{red}{\varphi = \varphi_P \cup \varphi_R \cup \varphi_F} \quad \phi_P = \phi_{P_1} \cup (\varphi \circ \phi_{P_2}) \\
\phi_R = (\phi_{R_1} \cup (\varphi \circ \phi_{R_2})) \setminus (\mathcal{M}_1 \cup \mathcal{M}_2) \\
Mv = Mv_1 \cup \{\varphi(i) \mapsto [E_i;\ \varphi\circ\phi_i;\ e_i]^{\ i\in\mathcal{I}_2}\}
\end{array}}{E \vdash T_1 + T_2 \longrightarrow \langle\!|\ Mv;\ \phi_P;\ \phi_R\ |\!\rangle}$$

$$
\frac{
\begin{array}{c}
E \vdash T_1 \longrightarrow \langle\!| \ {}^{\mathcal{I}_1}Mv_1;\ {}^{\mathcal{M}_1}\phi_{P_1};\ {}^{\mathcal{R}_1}\phi_{R_1}\ |\!\rangle \\
E \vdash T_2 \longrightarrow \langle\!| \ {}^{\mathcal{I}_2}Mv_2;\ {}^{\mathcal{M}_2}\phi_{P_2};\ {}^{\mathcal{R}_2}\phi_{R_2}\ |\!\rangle \\
\mathcal{M}_1 \pitchfork \mathcal{M}_2 \quad Mv_2 = \{i \mapsto [E_i;\ \phi_i;\ e_i]^{\ i\in\mathcal{I}_2}\} \\
\varphi_P = \{\phi_{P_2}(m) \mapsto \phi_{R_1}(m)^{\ m\in\mathcal{M}_2\cap\mathcal{R}_1}\} \\
\varphi_R = \{\phi_{R_2}(m) \mapsto \phi_{P_1}(m)^{\ m\in\mathcal{R}_2\cap\mathcal{M}_1}\} \\
\mathcal{I}_1' = \mathcal{I}_1 \cup \mathrm{rng}(\phi_{R_1}) \quad \mathcal{I}_2' = \mathcal{I}_2 \cup \mathrm{rng}(\phi_{R_2}) \\
\varphi_F = \mathrm{FS}(\mathcal{I}_2' \setminus \mathrm{dom}(\varphi_R \cup \varphi_P),\ \mathcal{I}_1') \\
\varphi = \varphi_P \cup \varphi_R \cup \varphi_F \quad \phi_P = \phi_{P_1} \cup (\varphi \circ \phi_{P_2}) \\
\phi_R = (\phi_{R_1} \cup (\varphi \circ \phi_{R_2})) \setminus (\mathcal{M}_1 \cup \mathcal{M}_2) \\
Mv = Mv_1 \cup \{\varphi(i) \mapsto [E_i;\ \varphi \circ \phi_i;\ e_i]^{\ i\in\mathcal{I}_2}\}
\end{array}
}{
E \vdash T_1 + T_2 \longrightarrow \langle\!|\ Mv;\ \phi_P;\ \phi_R\ |\!\rangle
}
$$

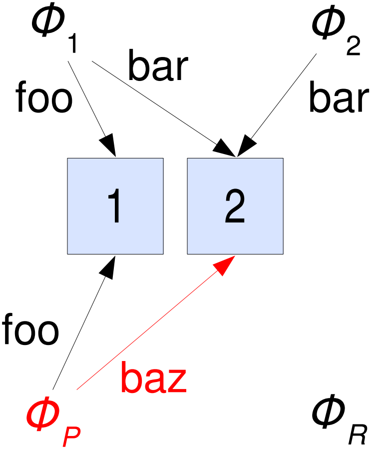


$$\text{TFooBar} = \text{TFoo} + \text{TBar}$$

$$\varphi = \{1 \mapsto 2\}$$

$$\phi_P = \{\text{foo} \mapsto 1,\ \text{bar} \mapsto 2\}$$
$$\phi_R = \{\}$$
$$\phi_1 = \{\text{foo} \mapsto 1,\ \text{bar} \mapsto 2\}$$
$$\phi_2 = \{\text{bar} \mapsto 2\}$$

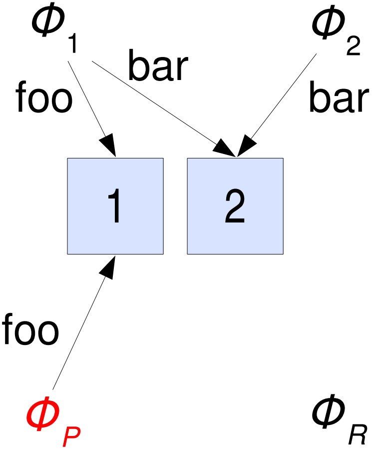
$$\frac{\begin{array}{c} E \vdash T \longrightarrow \langle\!|\ {}^{\mathcal{I}}Mv;\ {}^{\mathcal{M}}\phi_P;\ {}^{\mathcal{R}}\phi_R\ |\!\rangle \\ r \in \mathcal{M} \quad r' \notin \mathcal{M} \quad r' \notin \mathcal{R} \\ \phi'_P = (\phi_P \setminus r)[r' \mapsto \phi_P(r)] \end{array}}{E \vdash T\ \textbf{rename}\ r\ \textbf{to}\ r' \longrightarrow \langle\!|\ Mv;\ \phi'_P;\ \phi_R\ |\!\rangle}$$



TFooBaz =
TFooBar **rename** bar **to** baz

$\phi_P = \{\text{foo} \mapsto 1,\ \text{baz} \mapsto 2\}$
$\phi_R = \{\}$
$\phi_1 = \{\text{foo} \mapsto 1,\ \text{bar} \mapsto 2\}$
$\phi_2 = \{\text{bar} \mapsto 2\}$

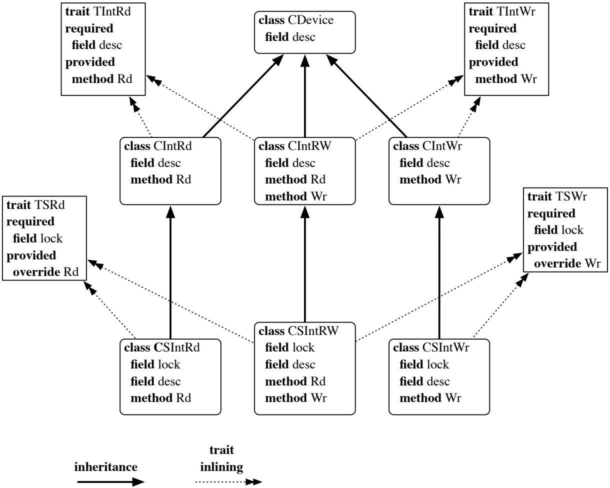
$$\frac{E \vdash T \longrightarrow \llparenthesis\ {}^{\mathcal{I}}Mv;\ {}^{\mathcal{M}}\phi_P;\ {}^{\mathcal{R}}\phi_R\ \rrparenthesis \qquad m \in \mathcal{M}}{E \vdash T\ \textbf{hide}\ m \longrightarrow \llparenthesis\ Mv;\ \phi_P \setminus m;\ \phi_R\ \rrparenthesis}$$



$\mathsf{TFoo}' = \mathsf{TFooBaz}$ **hide** baz

$\phi_P = \{\mathsf{foo} \mapsto 1\}$
$\phi_R = \{\}$
$\phi_1 = \{\mathsf{foo} \mapsto 1,\ \mathsf{bar} \mapsto 2\}$
$\phi_2 = \{\mathsf{bar} \mapsto 2\}$

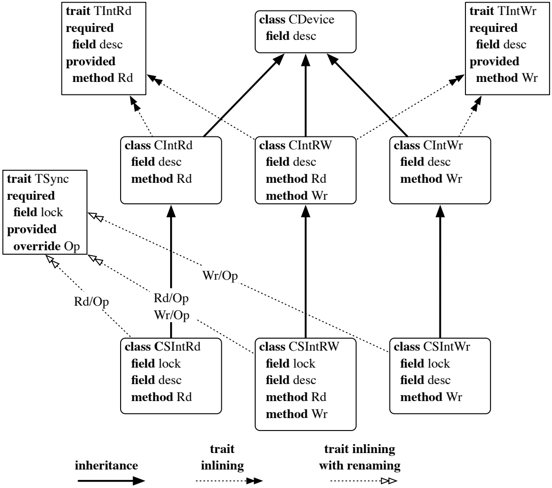
Recall the trait-based solution:



The TSync example

trait TSync<ty1, ty2> = {
    provides Op(x : ty1) : ty2 {
        self.lock.Acquire();
        super.Op(x) **before**
            self.lock.Release();
    }
    requires field lock : LockObj
}

This time with renaming:



λ?

Trait-based metaprogramming

Our calculus provides a *foundation* for "trait-based metaprogramming" by formalizing a new notion of substitution:

TSync **rename** Op **to** Read

Trait-based metaprogramming

Our calculus provides a *foundation* for "trait-based metaprogramming" by formalizing a new notion of substitution:

$$[\![\text{TSync Read}]\!] = \text{TSync } \textbf{rename} \text{ Op } \textbf{to} \text{ Read}$$

Trait-based metaprogramming

Our calculus provides a *foundation* for "trait-based metaprogramming" by formalizing a new notion of substitution:

$$\mathsf{TSync} \;=\; \lambda \mathsf{Op}.\textbf{trait}\ \{\ \textbf{provides}\ \mathsf{Op} \cdots\ \}$$

$$\left[\!\!\left[\mathsf{TSync\ Read}\right]\!\!\right] = \mathsf{TSync}\ \textbf{rename}\ \mathsf{Op}\ \textbf{to}\ \mathsf{Read}$$

Method name abstraction is only a first step!

Research challenge: design a concrete metalanguage using this notion of substitution.

- Java's **synchronized** keyword – instantiate TSync?
- Pointcuts and other AOP techniques?
- Type-directed application of abstracted traits?

Summary

Hiding and *renaming* are the deep analogs to excluding and aliasing.

- Useful for conflict resolution.
- Useful in isolation (for privacy and fixing “wrong” names).
- Renaming yields a new notion of substitution.

We model these features with a statically typed trait calculus and prove type soundness (see the tech report for proof details).

Our calculus provides a rigorous notion of substitution that can be used to build a trait metalanguage.

Some related work

- *The Programming Language Jigsaw: Mixins, Modularity and Multiple Inheritance*. Bracha, G. Dissertation.
- *Featherweight-trait Java: A trait-based extension for FJ*. Liquori, L. and A. Spiwack. Tech report.
- *Chai: Traits for Java-like languages.* Smith, C. and S. Drossopoulou. ECOOP'05.
- Aspect-oriented programming.

Thank you.