The Polymorphic Blame Calculus and Parametricity

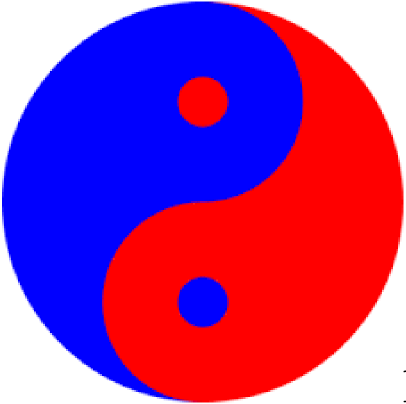
Jeremy G. Siek

Indiana University, Bloomington

University of Strathclyde
August 2015

Integrating static and dynamic typing

Static



Dynamic

Outline

- Quick review of gradual typing
- New: a polymorphic gradually typed lambda calculus
- Review: Poly. Blame Calculus and Parametricity

Gradual typing includes dynamic typing

An untyped program:

$$
\begin{array}{l}
\texttt{let} \\
\quad f = \lambda y.\, \texttt{1} + y \\
\quad h = \lambda g.\, g\ 3 \\
\texttt{in} \\
\quad h\, f \\
\longrightarrow \\
\texttt{4}
\end{array}
$$

Gradual typing includes dynamic typing

A buggy untyped program:

$$
\begin{array}{l}
\texttt{let} \\
\quad f = \lambda y.\, \texttt{1} + y \\
\quad h = \lambda g.\, g\ \texttt{true} \\
\texttt{in} \\
\quad h\, f
\end{array}
$$

$$\longrightarrow$$

$$\texttt{blame}\ \ell_2$$

Just like dynamic typing, the error is caught at run time.

Gradual typing includes static typing

A typed program:

$$
\begin{array}{l}
\texttt{let} \\
\quad f = \lambda y{:}\texttt{int}.\, 1 + y \\
\quad h = \lambda g{:}\texttt{int}{\rightarrow}\texttt{int}.\, g\ 3 \\
\texttt{in} \\
\quad h\, f \\
\longrightarrow \\
4
\end{array}
$$

Gradual typing includes static typing

An ill-typed program:

```
1  let
2     f = λy:int. 1 + y
3     h = λg:int→int. g true
4  in
5     h f
```

Just like static typing, the error is caught at compile time.

Error on line 3, the argument `true` is a Boolean, but function g expects an `int`.

Gradual typing provides fine-grained mixing

A partially typed program:

$$\begin{array}{l} \texttt{let} \\ \quad f = \lambda y{:}\texttt{int}.\ \texttt{1} + y \\ \quad h = \lambda g.\ g\ 3 \\ \texttt{in} \\ \quad h\ f \end{array}$$

$$\longrightarrow$$

$$\texttt{4}$$

Gradual typing protects type invariants

A buggy, partially typed program:

$$
\begin{array}{ll}
1 & \texttt{let} \\
2 & \quad f = \lambda y{:}\texttt{int}.\, \texttt{1} + y \\
3 & \quad h = \lambda g.\, g\ \texttt{true} \\
4 & \texttt{in} \\
5 & \quad h\, f
\end{array}
$$

$$\longrightarrow$$

$$\texttt{blame}\ \ell_3$$

Gradually Typed Lambda Calculus

Extends the STLC with a dynamic type, written $\star$.

$$\begin{array}{llcl} \text{Types} & A, B, C & ::= & \iota \mid A \to B \mid \star \\ \text{Terms} & L, M, N & ::= & c \mid x \mid \lambda x{:}A.\, N \mid L\, M \end{array}$$

Consistency $\boxed{A \sim B}$

$$\frac{}{A \sim \star} \qquad \frac{}{\star \sim B} \qquad \texttt{int} \sim \texttt{int} \qquad \frac{A_1 \sim B_1 \qquad A_2 \sim B_2}{A_1 \to A_2 \sim B_1 \to B_2}$$

Term Typing $\boxed{\Gamma \vdash M : A}$

$$\ldots \qquad \frac{\Gamma \vdash L : A \to B \qquad C \sim A \qquad \Gamma \vdash M : C}{\Gamma \vdash L\, M : B} \qquad \frac{\Gamma \vdash L : \star \qquad \Gamma \vdash M : C}{\Gamma \vdash L\, M : \star}$$

Outline

- Quick review of gradual typing
- **New: a polymorphic gradually typed lambda calculus**
- Review: Poly. Blame Calculus and Parametricity

Gradual typing and polymorphism

Use polymorphic code in an untyped context:

$$\begin{array}{l}\texttt{let}\\ \quad pos = \lambda x.\, x > 0\\ \quad app = \Lambda X.\, \Lambda Y.\, \lambda f{:}X{\to}Y.\, \lambda x{:}X.\, f\ x\\ \texttt{in}\\ \quad app\ pos\ 1\end{array}$$

Use untyped code in a polymorphic context:

$$\begin{array}{l}\texttt{let}\\ \quad pos : \texttt{int}{\to}\texttt{bool} = \lambda x{:}\texttt{int}.\, x > 0\\ \quad app = \lambda f.\, \lambda x.\, f\ x\\ \texttt{in}\\ \quad app\ \texttt{int}\ \texttt{bool}\ pos\ 1\end{array}$$

Gradually Typed Polymorphic Lambda Calculus

$$\begin{array}{llcl} \text{Types} & A, B, C & ::= & \iota \mid A \to B \mid \star \mid X \mid \forall X.\, A \\ \text{Terms} & L, M, N & ::= & c \mid x \mid \lambda x{:}A.\, N \mid L\, M \mid \Lambda X.\, N \mid L\, A \end{array}$$

Consistency $\boxed{A \sim B}$

$$\cdots \quad \frac{X \in \Gamma}{\Gamma \vdash X \sim X} \quad \frac{\Gamma, X \vdash A \sim B}{\Gamma \vdash \forall X.\, A \sim \forall X.\, B}$$

$$\frac{\Gamma, X \vdash A \sim B}{\Gamma \vdash A \sim \forall X.\, B} \quad \frac{\Gamma, X \vdash A \sim B}{\Gamma \vdash \forall X.\, A \sim B}$$

Term typing

$$\cdots \quad \frac{\Gamma \vdash L : \forall X.\, B}{\Gamma \vdash L\, A : B[X \mapsto A]} \quad \frac{\Gamma \vdash L : \star}{\Gamma \vdash L\, A : \star}$$

Consistency examples

$$\forall X.\, X \to X \sim \forall Y.\, Y \to Y$$

$$\forall X.\, X \to X \sim \star \qquad \star \sim \forall X.\, X \to X$$

$$\forall X.\, X \to X \sim \star \to \star \qquad \star \to \star \sim \forall X.\, X \to X$$

$$\forall X.\, X \to X \not\sim \texttt{int} \to \texttt{int} \qquad \texttt{int} \to \texttt{int} \not\sim \forall X.\, X \to X$$

$$\forall X.\, X \to X \not\sim \texttt{int} \to \texttt{bool} \qquad \texttt{int} \to \texttt{bool} \not\sim \forall X.\, X \to X$$

What about converting poly. to simple?

One might also want implicit conversion from polymorphic types to simple types, such as

$$\forall X.\, X \to X \Rightarrow \texttt{int} \to \texttt{int}$$

That is a separate concern from gradual typing. We could handle it with a subtyping rule

$$\frac{A[X \mapsto C] <: B}{\forall X.\, A <: B}$$

Then, for the type checking algorithm, combine subtyping and consistency as in Siek and Taha [2007].

Polymorphic type inference and containment,
John C. Mitchell, Information and Computation 1988.
Gradual Type for Objects, Siek and Taha, ECOOP 2007.

Translation semantics (cast insertion)

The semantics is defined by translation to the Polymorphic Blame Calculus.

Cast Insertion $\boxed{\Gamma \vdash M \leadsto M' : A}$

$$\ldots \qquad \frac{\Gamma \vdash L \leadsto L' : \star}{\Gamma \vdash L\ A \leadsto (L' : \star \overset{p}{\Rightarrow} \forall X.\, \star)\ A : \star}$$

Outline

- Quick review of gradual typing
- New: a polymorphic gradually typed lambda calculus
- **Review: Poly. Blame Calculus and Parametricity**

Semantics of casting from poly. to untyped

Recall the example:

$$
\begin{array}{l}
\texttt{let} \\
\quad pos = \lambda x.\, x > 0 \\
\quad app = \Lambda X.\, \Lambda Y.\, \lambda f{:}X \to Y.\, \lambda x{:}X.\, f\ x \\
\texttt{in} \\
\quad app\ pos\ 1
\end{array}
$$

So we have the cast:

$$app : \forall X.\, \forall Y.\, (X \to Y) \to X \to Y \stackrel{p}{\Rightarrow} \star$$

The Polymorphic Blame Calculus handles such casts by instantiating with $\star$.

$$V : (\forall X.\, A) \stackrel{p}{\Rightarrow} B \longrightarrow (V\ \star) : A[X \mapsto \star] \stackrel{p}{\Rightarrow} B$$

Blame for All. Ahmed et al. POPL 2011

Semantics of casting from untyped to poly.

Recall the example:

$$\begin{array}{l} \texttt{let} \\ \quad pos : \texttt{int}{\to}\texttt{bool} = \lambda x{:}\texttt{int}.\, x > 0 \\ \quad app = \lambda f.\, \lambda x.\, f\ x \\ \texttt{in} \\ \quad app\ \texttt{int}\ \texttt{bool}\ pos\ 1 \end{array}$$

So we have the cast:

$$app : \star \stackrel{p}{\Rightarrow} \forall X.\, \forall Y.\, (X{\to}Y){\to}X{\to}Y$$

The Polymorphic Blame Calculus handles such casts by generalizing.

$$V : A \stackrel{p}{\Rightarrow} (\forall X.\, B) \longrightarrow \Lambda X.\, (V : A \stackrel{p}{\Rightarrow} B) \qquad \text{if } X \notin \mathrm{ftv}(A)$$

Semantics of casts and parametricity

Consider casting the constant function

$$K = \lambda x{:}\star\,.\,\lambda y{:}\star\,.\,x$$

to the following polymorphic types

$$K_1 \equiv K : \star\!\to\star\to\!\star \overset{p}{\Rightarrow} \forall X.\,\forall Y.\,X\!\to\! Y\!\to\! X$$
$$K_2 \equiv K : \star\!\to\star\to\!\star \overset{q}{\Rightarrow} \forall X.\,\forall Y.\,X\!\to\! Y\!\to\! \color{red}{Y}$$

and the following scenarios:

$(K_1\ \texttt{int}\ \texttt{bool})\ 1\ \texttt{false} \longrightarrow^{*} 1$ $\quad(K_2\ \texttt{int}\ \texttt{bool})\ 1\ \texttt{false} \longrightarrow^{*}$

$(K_1\ \texttt{int}\ \texttt{int})\ 1\ 2 \longrightarrow^{*} 1$ $\quad(K_2\ \texttt{int}\ \texttt{int})\ 1\ 2 \longrightarrow^{*}$

Instantiation as type substition

Recall the traditional reduction rule:

$$(\Lambda X.\, N)\; A \longrightarrow N[X \mapsto A]$$

$$K_2 \equiv K : \star\!\to \star \to\!\star \overset{q}{\Rightarrow} \forall X.\, \forall Y.\, X\!\to\! Y\!\to\! {\color{red}Y}$$

$$\begin{aligned}
&\big(K_2\ \texttt{int}\ \texttt{bool}\big)\ 1\ \texttt{false}\\
\longrightarrow^*&\big(K : \star\!\to \star \to\!\star \overset{p}{\Rightarrow} \texttt{int}\!\to\!\texttt{bool}\!\to\!\texttt{bool}\big)\ 1\ \texttt{false}\\
\longrightarrow^*&\ 1 : \texttt{int} \Rightarrow \star \overset{p}{\Rightarrow} \texttt{bool}\\
\longrightarrow&\ \texttt{blame}\, p
\end{aligned}$$

so far so good...

The problem with type substitution

$$K_2 \equiv K : \star \to \star \to \star \overset{q}{\Rightarrow} \forall X.\, \forall Y.\, X \to Y \to Y$$

The second scenario for K_2:

$$\begin{aligned}
& (K_2\ \texttt{int}\ \texttt{int})\ 1\ 2 \\
\longrightarrow^{*} & (K : \star \to \star \to \star \overset{p}{\Rightarrow} \texttt{int} \to \texttt{int} \to \texttt{int})\ 1\ 2 \\
\longrightarrow^{*} & 1 : \texttt{int} \Rightarrow \star \overset{p}{\Rightarrow} \texttt{int} \\
\longrightarrow & 1
\end{aligned}$$

but a polymorphic function of type $\forall X.\, \forall Y.\, X \to Y \to Y$
must return its second argument, not first!

Solution: don't substitute, seal

$$(\Lambda X.\, V)\ A \longrightarrow \nu X{\mapsto}A.\ V$$

The example revisited:

$$K_2 \equiv K : \star{\to}\star{\to}\star \overset{q}{\Rightarrow} \forall X.\,\forall Y.\,X{\to}Y{\to}Y$$

$$\begin{aligned}
& (K_2\ \mathtt{int}\ \mathtt{int})\ 1\ 2 \\
\longrightarrow^*\ & (\nu X{\mapsto}\mathtt{int}.\ \nu Y{\mapsto}\mathtt{int}.\ K : \star{\to}\star{\to}\star \overset{p}{\Rightarrow} X{\to}Y{\to}Y)\ 1\ 2 \\
\longrightarrow^*\ & \nu X{\mapsto}\mathtt{int}.\ \nu Y{\mapsto}\mathtt{int}.\ 1 : X \Rightarrow \star \overset{p}{\Rightarrow} Y \\
\longrightarrow\ & \mathtt{blame}\,p
\end{aligned}$$

Types are not sets, James H. Morris, Jr., POPL 1973.

What to do with escaping seals?

$$\begin{aligned} & (\Lambda X.\, \lambda x{:}X.\, x : X \overset{p}{\Rightarrow} \star)\ \texttt{int}\ 2 \\ \longrightarrow^* \ & \nu X{\mapsto}\texttt{int}.\ 2 : X \overset{p}{\Rightarrow} \star \\ \longrightarrow \ & \texttt{blame}\, p_\nu \end{aligned}$$

Contrast with

$$\begin{aligned} & (\Lambda X.\, \lambda x{:}X.\, \texttt{inl}\, x\, \texttt{as}\, (X + \texttt{bool}))\ \texttt{int}\ 2 \\ \longrightarrow^* \ & \texttt{inl}\ 2\ \texttt{as}\ (\texttt{int} + \texttt{bool}) \end{aligned}$$

Why not?

$$\nu X{\mapsto}A.\ (V : X \overset{p}{\Rightarrow} \star) \longrightarrow (\nu X{\mapsto}A.\ V) : A \overset{p}{\Rightarrow} \star$$

Properties of the Polymorphic Blame Calculus

- ✓ Type Safety
- ✓ Blame Theorem
- ✓ Subtyping Theorem (weak version)
- □ Subtyping Theorem (strong version)
- □ Parametricity

Blame Theorem

Theorem (Blame Theorem)

Let M be a program with a subterm $N : A \stackrel{p}{\Rightarrow} B$ where the cast is labelled by the only occurrence of p in M, and $\bar{p}$ does not appear in M.

- *If $A <:^{+} B$, then $M \not\longrightarrow^{*}$ $\texttt{blame}\, p$.*
- *If $A <:^{-} B$, then $M \not\longrightarrow^{*}$ $\texttt{blame}\, \bar{p}$.*
- *If $A <:_{n} B$, then $M \not\longrightarrow^{*}$ $\texttt{blame}\, p$.*
- *If $B <:_{n} A$, then $M \not\longrightarrow^{*}$ $\texttt{blame}\, \bar{p}$.*

Subtyping Theorem

Theorem (Subtyping Theorem)

Let M be a program with a subterm $N : A \stackrel{p}{\Rightarrow} B$ where the cast is labelled by the only occurrence of p in M, and $\bar{p}$ does not appear in M.

- *If $A <: B$, then $M \not\longrightarrow^* \texttt{blame}\, p$ and $M \not\longrightarrow^* \texttt{blame}\, \bar{p}$.*

Weak version:

$$\frac{A[X \mapsto \star] <: B}{(\forall X.\, A) <: B}$$

(Proved in STOP 2009.)

Strong version:

$$\frac{A[X \mapsto T] <: B}{(\forall X.\, A) <: B}$$

(Incorrect proof in POPL 2011.)

Jack of all trades

Conjecture (Jack-of-All-Trades)

If $\Delta \vdash V : \forall X. A$ *and* $A[X{\mapsto}C] \prec B$ *(and hence* $A[X{\mapsto}\star] \prec B$*) then*

$$(V\ C : A[X{\mapsto}C] \overset{p}{\Rightarrow} B) \quad \sqsubseteq \quad (V\ \star : A[X{\mapsto}\star] \overset{p}{\Rightarrow} B).$$

Speculating about parametricity

Logical Relation

Terms

$$E[A]\delta k = \{(M, N) \mid \exists V W.\, M{\Downarrow_j} V, N{\Downarrow_j} W, (V, W) \in V[A]\delta(k{-}j)\}$$

Values

$$\begin{aligned}
V[\mathtt{int}]\delta k &= \{(n, n) \mid n \in \mathbf{Z}\} \\
V[A_1 + A_2]\delta k &= \{(\mathtt{inj}_i V, \mathtt{inj}_i W) \mid i \in 1..2, (V, W) \in V[A_i]\delta k\} \\
&\dots \\
V[\forall X.\, A]\delta k &= \{(V_1, V_2) \mid \forall R.\, (V_1[\cdot], V_2[\cdot]) \in E[A]\delta(X \mapsto R)k\} \\
V[X]\delta k &= \delta(X)\; k \\
V[\star]\delta(1 + k) &= \{(V : G \Rightarrow \star, W : G \Rightarrow \star) \mid (V, W) \in V[G]\delta k\}
\end{aligned}$$

Parametricity

Conjecture (Soundness of the Logical Relation)

If $\Delta; \Gamma \vdash M \approx N : A$, *then* $\Delta; \Gamma \vdash M =_{ctx} N : A$.

Conjecture (Fund. Theorem of Logical Relations)

If $\Delta; \Gamma \vdash M : A$, *then* $\Delta; \Gamma \vdash M \approx M : A$.