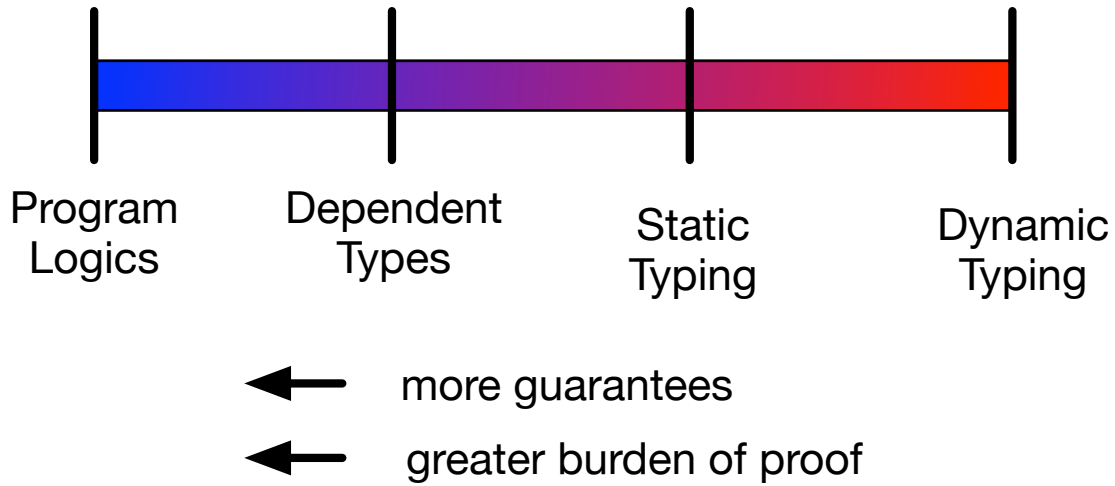
The Recursive Union of Some Gradual Types

Jeremy G. Siek Sam Tobin-Hochstadt
Indiana University, Bloomington

Gradual Typing

Goal: support the entire spectrum in one language.



Anderson and Drossopoulou, WOOD 2003. Ou et al., IFIP TCS 2004. Siek and Taha, SFP 2006. Gronksi et al., SFP 2006. Tobin-Hochstadt and Felleisen, DLS 2006. Matthews and Findler, POPL 2007.

Gradual typing provides fine-grained mixing

A partially typed program:

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

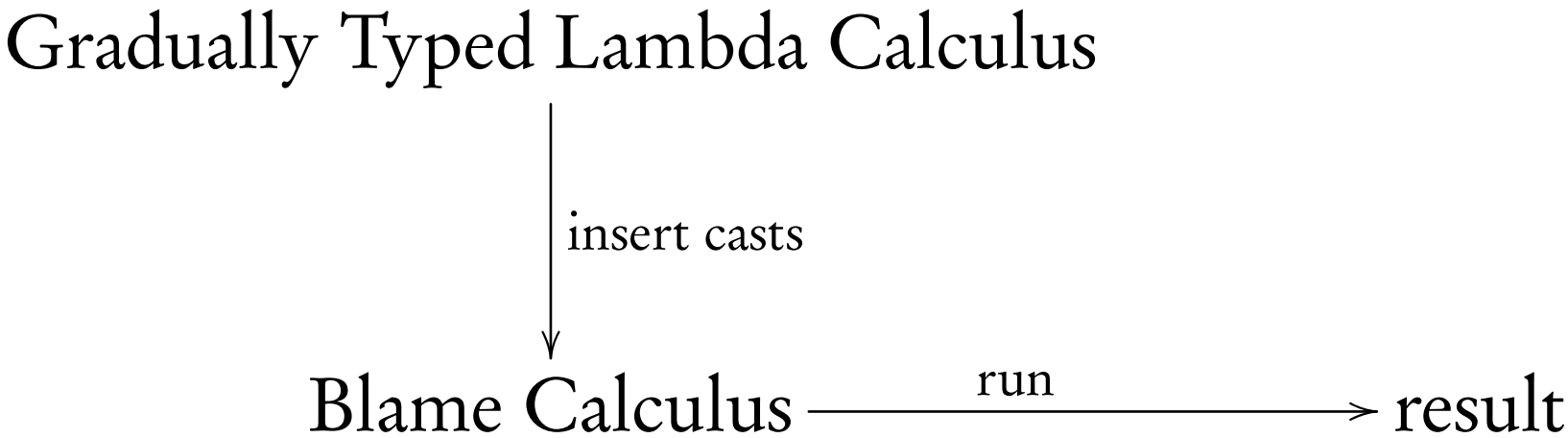
Gradual typing protects type invariants

A buggy, partially typed program:

$$
\begin{array}{l}
\texttt{let} \\
\quad f = \lambda y{:}\texttt{int}.\, 1 + y \\
\quad h = (\lambda g.\, g^{1}\ \texttt{true}^{2})^{3} \\
\texttt{in} \\
\quad h^{4}\ f^{5} \\
\longrightarrow \\
\quad \texttt{blame}\, -5
\end{array}
$$

Wadler & Findler, *Well-typed programs can't be blamed*, ESOP 2009.
Findler & Felleisen, *Contracts for higher-order functions*, ICFP 2002.

Semantics via the Blame Calculus



Translated to the blame calculus

The buggy program:

$$
\begin{array}{l}
\texttt{let} \\
\quad f = \lambda y{:}\texttt{int}.\ 1 + y \\
\quad h = (\lambda g : \star.\ (g : \star \overset{1}{\Rightarrow} \star \to \star)\ (\texttt{true} : \texttt{bool} \overset{2}{\Rightarrow} \star)) : \star \to \star \overset{3}{\Rightarrow} \star \\
\texttt{in} \\
\quad (h : \star \overset{4}{\Rightarrow} \star \to \star)\ (f : \texttt{int} \to \texttt{int} \overset{5}{\Rightarrow} \star)
\end{array}
$$

$$
\begin{array}{l}
\longrightarrow^* \\
\quad (f : \texttt{int} \to \texttt{int} \overset{5}{\Rightarrow} \star \to \star \overset{1}{\Rightarrow} \star \to \star\ (\texttt{true} : \texttt{bool} \overset{2}{\Rightarrow} \star)) \\
\longrightarrow^* \\
\quad (f\ ((\texttt{true} : \texttt{bool} \overset{2}{\Rightarrow} \star) : \star \overset{-1}{\Rightarrow} \star \overset{-5}{\Rightarrow} \texttt{int})) : \texttt{int} \overset{5}{\Rightarrow} \star \overset{1}{\Rightarrow} \star \\
\longrightarrow^* \\
\quad \texttt{blame}\ -5
\end{array}
$$

Desiderata

$$IntList = \mu X.\, \texttt{Unit} + \texttt{int} \times X$$

$$\begin{array}{l}
\texttt{letrec}\ prod = \lambda ls{:}IntList. \\
\quad \texttt{case}\ ls\ \texttt{of} \\
\qquad \texttt{unit} \Rightarrow 0 \\
\quad \mid (hd, tl) \Rightarrow hd * prod(tl) \\
\texttt{in} \\
\quad prod\ \lceil (1, (2, \texttt{unit})) \rceil
\end{array}$$

Racket & Typed Racket supports something like this, and here's the essence of that story in terms of a Blame Calculus.

Tobin-Hochstadt and Felleisen, *The Design and Impl. of Typed Scheme*, POPL 2008.

The problem with disjoint sums

What should the following reduce to?

$$4 : \texttt{int} \Rightarrow \star \Rightarrow (\texttt{int} + \texttt{int})$$

Possible answers:

1. `inl` 4
2. `inr` 4
3. Jeremy, don't ask that question!

Alternative: union types

- The `case` of set-theoretic union (Pierce, 1991) doesn't provide dispatching.

$$(\texttt{case}\ V\ \texttt{of}\ x \Rightarrow N) \longrightarrow [x := V]N$$

- ℂDuce's (Castagna, 2014) typecase is more powerful than Racket's:

$$(x = V \in A\ ?\ M \mid N) \longrightarrow \begin{cases} [x \mapsto V]M & \text{if} \vdash V : A \\ [x \mapsto V]N & \text{if} \vdash V : \neg A \end{cases}$$

- The "true union" type of Cartwright and Fagan (1991) matches our needs. We'll give an updated presentation.

$$\texttt{case}\ c?(V)\ \texttt{then}\ M\ \texttt{else}\ N \longrightarrow \begin{cases} M\ \ V & \text{if } \mathit{tycons}(V) = c \\ N\ \ V & \text{otherwise} \end{cases}$$

Union types

Base types	ι	$::=$	$\texttt{Unit} \mid \texttt{int} \mid \texttt{bool} \mid \cdots$
Type constructors	c	$::=$	$\iota \mid \rightarrow \mid \times$
Types	A, B, C, D	$::=$	$c(\bar{A}) \mid A \cup B \mid \bot$

Unions are restricted to one type per top type-constructor:

$$\frac{\vdash A_i \quad \forall i \in 1..n}{\vdash c(\bar{A})} \qquad \frac{tycons(A) \cap tycons(B) = \emptyset}{\vdash A \cup B} \qquad \frac{}{\vdash \bot}$$

$$\begin{aligned} tycons(c(\bar{A})) &= \{c\} \\ tycons(A \cup B) &= tycons(A) \cup tycons(B) \\ tycons(\bot) &= \emptyset \end{aligned}$$

Why the restriction on union types?

$$A = (\texttt{int} \to \texttt{int}) \cup (\texttt{bool} \to \texttt{bool})$$
$$f : A$$

$$\begin{array}{l}\texttt{case}\ \ \to?(f)\ \texttt{then} \\ \quad (\lambda x{:}A\,.M) \\ \texttt{else} \\ \quad (\lambda x{:}\bot.\,N)\end{array}$$

Racket's type predicates can't eliminate arbitrary unions.

Unions as partial functions on constructors

$\boxed{A(c)}$

$$c'(\bar{A})(c) = \begin{cases} c'(\bar{A}) & \text{if } c = c' \\ \bot & \text{otherwise} \end{cases}$$

$$(A \cup B)(c) = \begin{cases} A(c) & \text{if } c \in \mathit{tycons}(A) \\ B(c) & \text{if } c \in \mathit{tycons}(B) \end{cases}$$

$\boxed{A - c}$

$$c'(\bar{A}) - c = \begin{cases} \bot & \text{if } c = c' \\ c'(\bar{A}) & \text{otherwise} \end{cases}$$

$$(A \cup B) - c = (A - c) \cup (B - c)$$

$$\bot - c = \bot$$

Blame calculus with unions

$$
\begin{array}{lcl}
A, B, C, D & ::= & c(\bar{A}) \mid A \cup B \mid \bot \mid \star \\
p, q & ::= & +\ell \mid -\ell \\
L, M, N & ::= & k \mid op(\bar{M}) \mid \lambda x{:}A.\, N \mid L\ N \mid \\
 & & (M, N) \mid \mathtt{fst}\ L \mid \mathtt{snd}\ L \mid M : A \overset{p}{\Rightarrow} B \mid \\
 & & \mathtt{case}\ c?(L)\ \mathtt{then}\ M\ \mathtt{else}\ N
\end{array}
$$

$$
\ldots \qquad \frac{\Gamma \vdash M : A \qquad A \sim B}{\Gamma \vdash M : A \overset{p}{\Rightarrow} B} \qquad \frac{\Gamma \vdash M : A \qquad A <: B}{\Gamma \vdash M : B}
$$

$$
\frac{\Gamma \vdash L : A \qquad \Gamma \vdash M : A(c) \to B \qquad \Gamma \vdash N : A - c \to B}{\Gamma \vdash (\mathtt{case}\ c?(L)\ \mathtt{then}\ M\ \mathtt{else}\ N) : B}
$$

Compatibility for unions

$\boxed{A \sim B}$

$$A \sim \star \qquad \star \sim A \qquad \frac{A_i \sim B_i \quad \forall i \in 1..n}{c(\bar{A}) \sim c(\bar{B})}$$

$$\frac{tycons(A) = tycons(B) \qquad \forall c \in tycons(A).\ A(c) \sim B(c)}{A \sim B} \qquad \frac{}{\bot \sim \bot}$$

Subtyping for unions

$\boxed{A <: B}$

$$\frac{\forall c \in \mathit{tycons}(A).\ A(c) <: B}{A <: B} \qquad \frac{c(\bar{A}) <: B(c)}{c(\bar{A}) <: B}$$

$$\frac{}{\iota <: \iota} \qquad \frac{A <: C \qquad B <: D}{A \times B <: C \times D} \qquad \frac{C <: A \qquad B <: D}{A \to B <: C \to D}$$

Also, we take $A \cup \bot = \bot \cup A = A$.

Inspired by Vouillon (2004).

Operational semantics for casting unions

$$
\begin{array}{lrcl}
\text{Ground types} & G, H & ::= & c(\bar{\star}) \mid G \cup H \mid \bot \\
\text{Values} & V, W & ::= & k \mid \lambda x{:}A.\, N \mid (V, W) \mid V : G \overset{p}{\Rightarrow} \star
\end{array}
$$

$$
V : A \overset{p}{\Rightarrow} B \longrightarrow V : A(c) \overset{p}{\Rightarrow} B(c) \quad \text{if } c = \mathit{tycons}(V) \text{ and } \neg(A = A(c) \wedge B = B(c))
$$

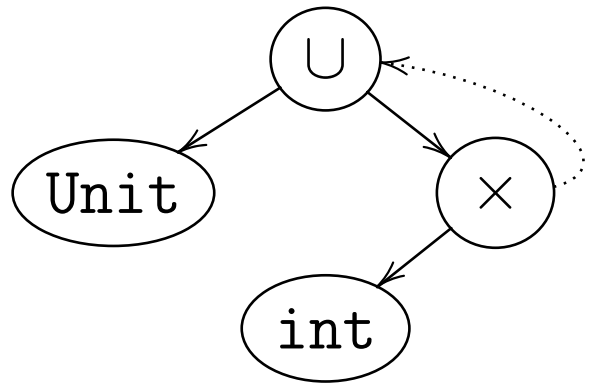
$$
V : G \overset{q}{\Rightarrow} \star \overset{p}{\Rightarrow} H \longrightarrow V \quad \text{if } G <: H
$$

$$
V : G \overset{q}{\Rightarrow} \star \overset{p}{\Rightarrow} H \longrightarrow \mathtt{blame}\, p \quad \text{if } G \not<: H
$$

Equi-recursive types

$$\mathit{IntList} = \mu X.\, \texttt{Unit} \cup \texttt{int} \times X$$

$$=$$



$$\begin{array}{lllll} \text{Pre-types} & P, Q & ::= & c(\bar{A}) \mid P \cup Q \mid \bot \mid \star \\ \text{Types} & A, B, C & ::= & \mu X.\, P \mid X \end{array}$$

Compatibility for equi-recursive types

$$P \sim \star \quad \star \sim Q \qquad \frac{A_i \sim B_i \quad \forall i \in 1..n}{c(\bar{A}) \sim c(\bar{B})} \qquad \boxed{P \sim Q}$$

$$\frac{\begin{array}{c} tycons(P) = tycons(Q) \\ \forall c \in tycons(P).\ P(c) \sim Q(c) \end{array}}{P \sim Q} \qquad \frac{}{\bot \sim \bot}$$

$$\frac{P \sim Q}{\mu X.\, P \sim \mu X.\, Q} \qquad X \sim X \qquad \boxed{A \sim B}$$

Subtyping for equi-recursive types

Subtype Environment $\Sigma \quad ::= \quad \emptyset \mid \Sigma, A <: B$

Subtyping on pre-types $\boxed{\Sigma \vdash P <: Q}$

$$\frac{}{\Sigma \vdash \iota <: \iota} \qquad \frac{\Sigma \vdash A <: C \quad \Sigma \vdash B <: D}{\Sigma \vdash A \times B <: C \times D} \qquad \frac{\Sigma \vdash C <: A \quad \Sigma \vdash B <: D}{\Sigma \vdash A \to B <: C \to D}$$

$$\frac{\forall c \in \mathit{tycons}(\mathcal{C}).\ \Sigma \vdash P(c) <: Q}{\Sigma \vdash P <: Q} \qquad \frac{\Sigma \vdash c(\bar{A}) <: Q(c)}{\Sigma \vdash c(\bar{A}) <: Q}$$

Subtyping $\boxed{\Sigma \vdash A <: B}$

$$\frac{A <: B \in \Sigma}{\Sigma \vdash A <: B} \qquad \frac{A = \mu X.\, P \qquad B = \mu Y.\, Q \qquad \Sigma, A <: B \vdash [X \mapsto A]P <: [Y \mapsto B]Q}{\Sigma \vdash A <: B}$$

Inspired by Brandt and Henglein (1998).

Operational semantics for casting recursive types

$$
\begin{array}{lllll}
\text{Ground pre-types} & \gamma & ::= & c(\overline{\star}) \mid \gamma \cup \gamma \mid \bot \\
\text{Ground types} & G, H & ::= & \mu X.\, \gamma \mid X \\
\text{Values} & V, W & ::= & k \mid \lambda x{:}A.\, N \mid (V, W) \mid V : G \overset{p}{\Rightarrow} \star
\end{array}
$$

$$
V : \mu X.\, A \overset{p}{\Rightarrow} \mu X.\, B \longrightarrow V : [X \mapsto \mu X.\, A]A \overset{p}{\Rightarrow} [X \mapsto \mu X.\, B]B
$$

Back to the example

$$\texttt{IntList} \equiv \mu X.\, \texttt{Unit} \cup (\texttt{int} \times X)$$
$$\texttt{DynList} \equiv \mu X.\, \texttt{Unit} \cup (\star \times \star)$$

$$
\begin{aligned}
\longrightarrow^* \; & (1, (2, \texttt{unit})) : (\star \times \star) \overset{q}{\Rightarrow} \star \overset{p}{\Rightarrow} \texttt{IntList} \\
\longrightarrow \; & (1, (2, \texttt{unit})) : (\star \times \star) \overset{q}{\Rightarrow} \star \overset{p}{\Rightarrow} \texttt{DynList} \overset{p}{\Rightarrow} \texttt{IntList} \\
\longrightarrow \; & (1, (2, \texttt{unit})) : \texttt{DynList} \overset{p}{\Rightarrow} \texttt{IntList} \\
\longrightarrow \; & (1, (2, \texttt{unit})) : (\texttt{Unit} \cup (\star \times \star)) \overset{p}{\Rightarrow} (\texttt{Unit} \cup (\texttt{int} \times \texttt{IntList})) \\
\longrightarrow \; & (1, (2, \texttt{unit})) : (\star \times \star) \overset{p}{\Rightarrow} (\texttt{int} \times \texttt{IntList}) \\
\longrightarrow \; & (1 : \star \overset{p}{\Rightarrow} \texttt{int}, (2, \texttt{unit}) : \star \overset{p}{\Rightarrow} \texttt{IntList}) \\
\longrightarrow \; & (1, (2, \texttt{unit}) : \star \overset{p}{\Rightarrow} \texttt{IntList}) \\
\longrightarrow^* \; & (1, (2, \texttt{unit}))
\end{aligned}
$$

Other things in the paper

- Definition of a gradually typed lambda calculus with unions and equi-recursive types.
- Type checking algorithm for this source language.
- Translation to the blame calculus.
- Proofs of Type Safety and Blame Safety for the blame calculus.

A remark about types

Real Types	**Type Enhancers**
$n : \texttt{int}$	$\{n : \texttt{int} \mid n > 0\}$
$(V, W) : A \times B$	$A \cap B$
$\texttt{inl}\ V, \texttt{inr}\ W : A + B$	$A \cup B$
$(\Lambda X.M) : \forall X.A$	type schemes (ML)
$\texttt{fold}\, M : \mu X.A$	equi-recursive

Is there already a name for this distinction?

Conclusion

Thank you to Philip for a great collaboration!