

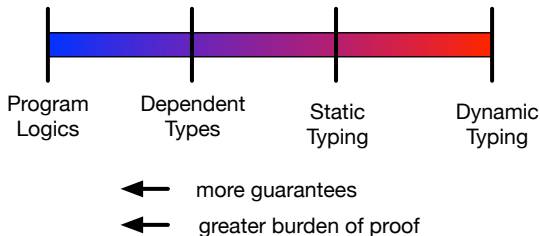
The Recursive Union of Some Gradual Types

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:

```
let
   $f = \lambda y:\text{int}. 1 + y$ 
   $h = \lambda g.g\ 3$ 
in
   $h\ f$ 
→
4
```

Gradual typing protects type invariants

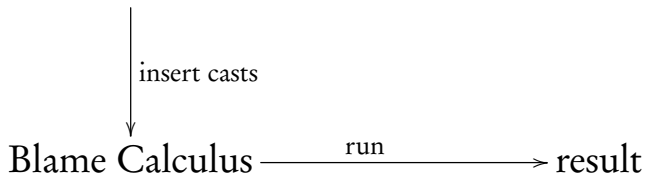
A buggy, partially typed program:

```
let
  f =  $\lambda y:\text{int}. 1 + y$ 
  b =  $(\lambda g.g^1 \text{true}^2)^3$ 
in
   $b^4 f^5$ 
→
  blame -5
```

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

Gradually Typed Lambda Calculus



Translated to the blame calculus

The buggy program:

let

$f = \lambda y:\text{int}. 1 + y$

$b = (\lambda g : \star. (g : \star \xrightarrow{1} \star \rightarrow \star) (\text{true} : \text{bool} \xrightarrow{2} \star)) : \star \rightarrow \star \xrightarrow{3} \star$

in

$(b : \star \xrightarrow{4} \star \rightarrow \star) (f : \text{int} \rightarrow \text{int} \xrightarrow{5} \star)$

\longrightarrow^*

$(f : \text{int} \rightarrow \text{int} \xrightarrow{5} \star \rightarrow \star \xrightarrow{1} \star \rightarrow \star (\text{true} : \text{bool} \xrightarrow{2} \star))$

\longrightarrow^*

$(f ((\text{true} : \text{bool} \xrightarrow{2} \star) : \star \xrightarrow{-1} \star \xrightarrow{-5} \text{int})) : \text{int} \xrightarrow{5} \star \xrightarrow{1} \star$

\longrightarrow^*

blame -5

Desiderata

$IntList = \mu X. Unit + int \times X$

```
letrec prod =  $\lambda ls: IntList.$   
  case  $ls$  of  
    unit  $\Rightarrow$  0  
    |  $(hd, tl) \Rightarrow hd * prod(tl)$   
in  
  prod [(1, (2, unit))]
```

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?

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

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.

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

- ▶ 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.

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

Union types

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

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

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

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

Why the restriction on union types?

$A = (\text{int} \rightarrow \text{int}) \cup (\text{bool} \rightarrow \text{bool})$

$f : A$

case $\rightarrow?(f)$ then

$(\lambda x: A. M)$

else

$(\lambda x: \perp. N)$

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

Unions as partial functions on constructors

$$\boxed{A(c)}$$

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

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

$$\boxed{A - c}$$

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

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

$$\perp - c = \perp$$

Blame calculus with unions

$$\begin{aligned} A, B, C, D & ::= c(\bar{A}) \mid A \cup B \mid \perp \mid \star \\ p, q & ::= +\ell \mid -\ell \\ L, M, N & ::= k \mid op(\bar{M}) \mid \lambda x:A. N \mid L N \mid \\ & (M, N) \mid \text{fst } L \mid \text{snd } L \mid M : A \xrightarrow{p} B \mid \\ & \text{case } c?(L) \text{ then } M \text{ else } N \end{aligned}$$
$$\dots \quad \frac{\Gamma \vdash M : A \quad A \sim B}{\Gamma \vdash M : A \xrightarrow{p} B} \quad \frac{\Gamma \vdash M : A \quad A <: B}{\Gamma \vdash M : B}$$
$$\frac{\Gamma \vdash L : A \quad \Gamma \vdash M : A(c) \rightarrow B \quad \Gamma \vdash N : A - c \rightarrow B}{\Gamma \vdash (\text{case } c?(L) \text{ then } M \text{ else } N) : B}$$

Compatibility for unions

$$\boxed{A \sim B}$$

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

$$\frac{\begin{array}{l} \text{tycons}(A) = \text{tycons}(B) \\ \forall c \in \text{tycons}(A). A(c) \sim B(c) \end{array}}{A \sim B} \quad \frac{}{\perp \sim \perp}$$

Subtyping for unions

$$\boxed{A <: B}$$

$$\frac{\forall c \in \text{tycons}(A). A(c) <: B}{A <: B} \qquad \frac{c(\bar{A}) <: B(c)}{c(\bar{A}) <: B}$$
$$\frac{}{\iota <: \iota} \qquad \frac{A <: C \quad B <: D}{A \times B <: C \times D}$$
$$\frac{C <: A \quad B <: D}{A \rightarrow B <: C \rightarrow D}$$

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

Operational semantics for casting unions

Ground types $G, H ::= c(\bar{x}) \mid G \cup H \mid \perp$

Values $V, W ::= k \mid \lambda x:A. N \mid (V, W) \mid V : G \xRightarrow{p} \star$

$$V : A \xRightarrow{p} B \longrightarrow V : A(c) \xRightarrow{p} B(c)$$

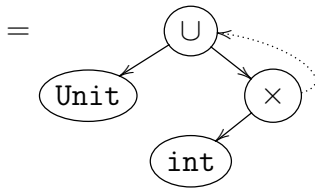
if $c = \text{tycons}(V)$ and $\neg(A = A(c) \wedge B = B(c))$

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

$$V : G \xRightarrow{q} \star \xRightarrow{p} H \longrightarrow \text{blame } p \quad \text{if } G \not<: H$$

Equi-recursive types

$$IntList = \mu X. Unit \cup int \times X$$



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

Compatibility for equi-recursive types

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

$$\frac{\begin{array}{l} \text{tycons}(P) = \text{tycons}(Q) \\ \forall c \in \text{tycons}(P). P(c) \sim Q(c) \end{array}}{P \sim Q} \quad \frac{}{\perp \sim \perp}$$

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

Subtyping for equi-recursive types

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

Subtyping on pre-types

$\Sigma \vdash P <: Q$

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

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

Subtyping

$\Sigma \vdash A <: B$

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

Inspired by Brandt and Henglein (1998).

Operational semantics for casting recursive types

Ground pre-types	γ	::=	$c(\bar{\star}) \mid \gamma \cup \gamma \mid \perp$
Ground types	G, H	::=	$\mu X. \gamma \mid X$
Values	V, W	::=	$k \mid \lambda x:A. N \mid (V, W) \mid V : G \xRightarrow{p} \star$

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

Back to the example

$$\text{IntList} \equiv \mu X. \text{Unit} \cup (\text{int} \times X)$$

$$\text{DynList} \equiv \mu X. \text{Unit} \cup (\star \times \star)$$

$$\longrightarrow^* (1, (2, \text{unit})) : (\star \times \star) \xrightarrow{q} \star \xrightarrow{p} \text{IntList}$$

$$\longrightarrow (1, (2, \text{unit})) : (\star \times \star) \xrightarrow{q} \star \xrightarrow{p} \text{DynList} \xrightarrow{p} \text{IntList}$$

$$\longrightarrow (1, (2, \text{unit})) : \text{DynList} \xrightarrow{p} \text{IntList}$$

$$\longrightarrow (1, (2, \text{unit})) : (\text{Unit} \cup (\star \times \star)) \xrightarrow{p} (\text{Unit} \cup (\text{int} \times \text{IntList}))$$

$$\longrightarrow (1, (2, \text{unit})) : (\star \times \star) \xrightarrow{p} (\text{int} \times \text{IntList})$$

$$\longrightarrow (1 : \star \xrightarrow{p} \text{int}, (2, \text{unit}) : \star \xrightarrow{p} \text{IntList})$$

$$\longrightarrow (1, (2, \text{unit})) : \star \xrightarrow{p} \text{IntList}$$

$$\longrightarrow^* (1, (2, \text{unit}))$$

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

Is there already a name for this distinction?

Conclusion

Thank you to Philip for a great collaboration!