# 漸進的型付き多相ラムダ計算

五十嵐 雄 (京都大学)　　関山 太朗 (IBM Research - Tokyo)　　五十嵐 淳 (京都大学)

## Our Goal

Gragual typing between **System F** and the **untyped λ-calculus**

## Gradual Typing [1] — smooth integration of static typing and dynamic typing

Surface language

$$(\lambda x{:}\star\,.\ \mathrm{negate}\ x)\ 42 \rightsquigarrow (\lambda x{:}\star\,.\ \mathrm{negate}\ (x : \star \Rightarrow \mathrm{Int}))\ (42 : \mathrm{Int} \Rightarrow \star) \longrightarrow^{\star} -42$$

Internal language (cast calculus)

1. Relax typechecking for the dynamic type: $\star$
2. Insert casts to detect runtime type errors
3. Reduce in a cast calculus

**Typing rule for application**

$$\frac{\Gamma \vdash s : T_{11} \to T_{12} \qquad \Gamma \vdash t : T_1 \qquad T_{11} \sim T_1}{\Gamma \vdash s\ t : T_{12}}$$

**Type consistency A ~ B**

$$\frac{}{A \sim A} \qquad \frac{}{A \sim \star} \qquad \frac{}{\star \sim A} \qquad \frac{A \sim A' \qquad B \sim B'}{A \to B \sim A' \to B'}$$

**Typing rule for cast**

$$\frac{\Gamma \vdash t : A \qquad A \sim B}{\Gamma \vdash (t : A \Rightarrow B) : B}$$

## Our Work — System $\mathrm{F_G}$ : A gradually typed extension of System F

- Goal: Prove all the five desired criteria [3].
  Remark: Blame for All [2] studies a cast calculus with polymorphism, but does not cover the criteria.
- Approach:
  - Design a new type consistency relation
  - Redesign a cast calculus: System $\mathrm{F_C}$

**The criteria**

- ✓ Conservativity over Untyped λ
- ✓ Conservativity over System F
- ✓ Type safety
- ✓ Blame Theorem
- △ Gradual guarantee

**Progress: 4.5 / 5**

## Technical Detail

### 1. Type consistency with ∀-types

**Intuition:**

similarity between $\star$ and X

$$(\lambda x{:}\star\,.\ x) \qquad (\Lambda X.\ (\lambda x{:}X.\ x))$$
$$\star \to \star \qquad \forall X.X \to X$$

$\star$ could be replaced with type variables.
A non-∀ type containing $\star$ could be used as a ∀-type by replacing $\star$ with type variables.

**Formal definition:**

$$\frac{A \sim B \qquad B \neq \forall Y.B' \qquad X \notin \mathrm{ftv}(B) \qquad B \text{ contains at least one } \star\,.}{\forall X.A \sim B}$$

Remark: ftv(B) is a set of free type variables in B.

**Result:**

$(\lambda x{:}\forall X.\mathrm{Int}.\ x)\ 42$ is ill-typed in System F.

Also ill-typed in System $\mathrm{F_G}$ because $\forall X.\mathrm{Int} \not\sim \mathrm{Int}$.

Proved! **conservativity over typing of System F**

For all terms not containing $\star$ , typing of System $\mathrm{F_G}$ should be the same as that of System F.

### 2. Static and gradual type abstractions

**Background:**

Enfoce parametricity at runtime with *type bindings* [2]

$$\emptyset \triangleright (\Lambda X.\ (\lambda x{:}X.\ (x : X \Rightarrow \star \Rightarrow \mathrm{Int})))\ \mathrm{Int}\ 9$$
$$\longrightarrow X := \mathrm{Int} \triangleright (\lambda x{:}X.\ (x : X \Rightarrow \star \Rightarrow \mathrm{Int}))\ 9$$
$$\longrightarrow X := \mathrm{Int} \triangleright (9 : X \Rightarrow \star \Rightarrow \mathrm{Int}) \longrightarrow \mathrm{error}$$

**Problem:**

Generation of type bindings is sometimes **redundant**.

$$(\Lambda X.\ (\lambda x{:}X.\ (x : X \Rightarrow X)))\ A\ v$$

**Solution:**

distinguish the redundant cases by *static and gradual type abstractions*

$$\emptyset \triangleright (\Lambda X{::}S.\ v)\ A \longrightarrow \emptyset \triangleright v[X := A]$$
$$\emptyset \triangleright (\Lambda X{::}G.\ v)\ A \longrightarrow X := A \triangleright v$$

Our new consistency enforces $X{::}S \not\sim \star$ .

**Results:**

- Clear correspondence to semantics of System F
- Efficient runtime semantics reflecting the distinction (informally demonstrated by a prototype of an interpreter)

## References

[1] Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, pages 81–92, September 2006.
[2] Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. Blame for All. In *POPL*, January 2011.
[3] Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. Refined criteria for gradual typing. In *1st Summit on Advances in Programming Languages*, pages 274–293, 2015.