

漸進的型付き多相ラムダ計算

五十嵐 雄 (京都大学)

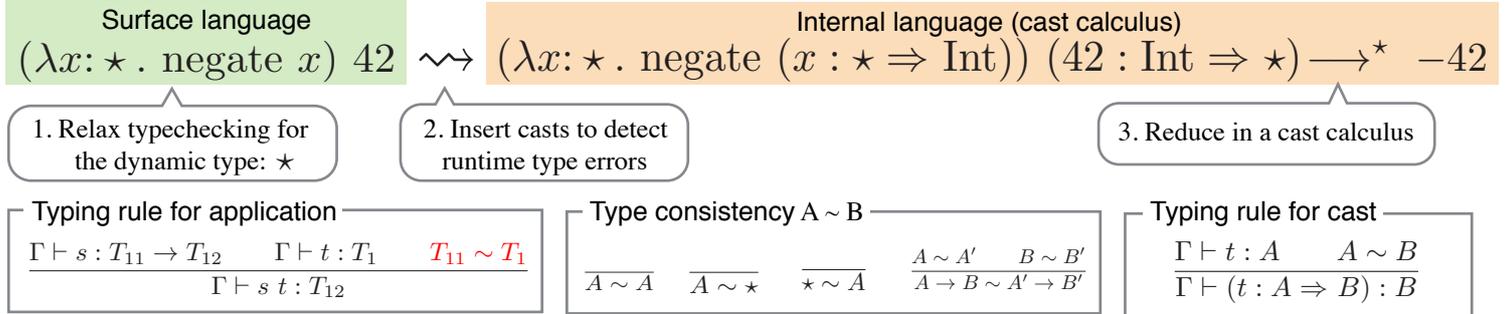
関山 太朗 (IBM Research - Tokyo)

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



Our Work — System 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 F_G

The criteria

- ✓ Conservativity over Untyped λ ✓ Blame Theorem
- ✓ Conservativity over System F Δ Gradual guarantee
- ✓ Type safety **Progress: 4.5 / 5**

Technical Detail

1. Type consistency with \forall -types

Intuition:

similarity between \star and X

$$\begin{array}{cc} (\lambda x: \star . x) & (\Lambda X. (\lambda x: X. x)) \\ \star \rightarrow \star & \forall X. X \rightarrow X \end{array}$$

\star could be replaced with type variables.

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

Formal definition:

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

Remark: $\text{ftv}(B)$ is a set of free type variables in B .

Result:

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

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

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

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

2. Static and gradual type abstractions

Background:

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

$$\begin{array}{l} \emptyset \triangleright (\Lambda X. (\lambda x: X. (x : X \Rightarrow \star \Rightarrow \text{Int}))) \ \text{Int} \ 9 \\ \longrightarrow X := \text{Int} \triangleright (\lambda x: X. (x : X \Rightarrow \star \Rightarrow \text{Int})) \ 9 \\ \longrightarrow X := \text{Int} \triangleright (9 : X \Rightarrow \star \Rightarrow \text{Int}) \longrightarrow \text{error} \end{array}$$

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.