On the Essence of Gradual Typing

Benjamin Chung, Jan Vitek

PRL, Northeastern University

May 14, 2016

Languages

Everyone wants to add types to their own language:

- Javascript
- Python
- Racket
- PHP
- Lua
- Ruby
- C#

Mismatches

Gradual type systems and their languages are tightly coupled.

$$\frac{t \neq ! C}{\text{undefined} <: t}$$

Javascript

$$\frac{\begin{array}{c}\Gamma \mid \Sigma \vdash e_s : \textbf{Class}\ er_s \qquad \Gamma \vdash er_s \backslash m_i^p \qquad \Gamma \vdash m_i^o \in er_s \\ \Gamma, \texttt{this} : \tau_{obj}, x_i^p : \tau_{1_i}^p \overline{f : \tau_f} \mid \Sigma \vdash e_i^p : \tau_{2_i}^p \\ \Gamma, \texttt{this} : \tau_{obj}, x_i^o : \tau_{1_i}^o, \overline{f : \tau_f} \mid \Sigma \vdash e_i^o : \tau_{2_i}^o \\ \Gamma \vdash v_{f_i} : \tau_{f_i} \qquad er = er_s \oplus \{\overline{(m^p : \tau_1^p \to \tau_2^p)}\} \qquad \tau_{obj} = \textbf{Object}\ \mathsf{R}[\![er]\!]\end{array}}{\Gamma \vdash \texttt{class}(e_s)\ \{\overline{f : \tau_f := v_f}\ \ \overline{m^p(x^p : \tau_1^p) : \tau_2^p\ e^p}\ \ \overline{m^o(x^o : \tau_1^o) : \tau_2^o\ e^o}\} : \textbf{Class}\ er}$$

Racket

- Hard to apply to a new language
- Key details are completely obscured
- Difficult to compare to one another

Our mistake

- We understand gradual typing, right?
- Porting some different approaches to one language can't be that hard, surely.
- It can't take that long to do, could be interesting.
- 1 year later ....

Calculi

$$
\begin{array}{lll}
e & ::= & x \\
 & | & e.f \\
 & | & e.f := e \\
 & | & e.m(\overline{e}) \\
 & | & \textbf{new}\ C(\overline{e}) \\
 & | & \langle t \rangle e
\end{array}
\qquad
\begin{array}{lll}
fd & ::= & f : t \\
md & ::= & m(\overline{x : t}) : t\ \{\, e \,\} \\
c & ::= & \textbf{class}\ C\ \{\, \overline{fd}\ \overline{md} \,\} \\
mt & ::= & m(\overline{t}) : t \\
t & ::= & \star \\
 & | & \{\overline{mt}\}
\end{array}
$$

Core System

- Gradual typing systems are generally extremely similar, the differences are small.
- To avoid duplication, we use a common core system.
- Broadly similar to Gradual Typing for Objects, Siek and Taha 2007.
- Based on cast insertion, supports structural subtyping, uses the guarded semantics.

Static Typing

$$\frac{\Gamma \vdash e : \star \qquad \overline{\Gamma \vdash e_1 : t}}{\Gamma \vdash e.m(\overline{e_1}) : \star}$$

$$\frac{\Gamma \vdash e : t_1 \qquad m(\overline{t_2}) : t \in t_1 \qquad \overline{\Gamma \vdash e_1 : t_2}}{\Gamma \vdash e.m(\overline{e_1}) : t}$$

Synthetic Cast Insertion

$$\frac{\Gamma \vdash e_1 \hookrightarrow e_2 \Rightarrow t_1 \qquad m(\overline{t_2}) : t_3 \in t_1 \qquad \overline{\Gamma \vdash e_3 \rightsquigarrow e_4 \Leftarrow t_2}}{\Gamma \vdash e_1.m(\overline{e_2}) \hookrightarrow e_3.m(\overline{e_4}) \Rightarrow t_3}$$

$$\frac{\Gamma \vdash e_1 \hookrightarrow e_3 \Rightarrow \star \qquad \overline{\Gamma \vdash e_2 \rightsquigarrow e_4 \Leftarrow \star}}{\Gamma \vdash e_1.m(\overline{e_2}) \hookrightarrow \langle \{m^\star(\overline{\star}) : \star\} \rangle e_3.m^\star(\overline{e_4}) \Rightarrow \star}$$

Analytic Cast Insertion

Typically a function of the specific system in question.

$$\frac{\Gamma \vdash e_1 \hookrightarrow e_2 \Rightarrow t_2 \qquad t_2 <: t_1}{\Gamma \vdash e_1 \rightsquigarrow e_2 \Leftarrow t_1}$$

$$\frac{\Gamma \vdash e_1 \hookrightarrow e_2 \Rightarrow \star}{\Gamma \vdash e_1 \rightsquigarrow \langle t_1 \rangle e_2 \Leftarrow t_1}$$

Class Translation

- For every field $f : t$, add getter and setter methods $f() : t$ and $f!(t) : t$
- For every method $m(\bar{t}) : t$ (including getters and setters), add a guard method $m^{\star}(\bar{\star}) : \star$, that casts the inputs to the fully specified type and the output to $\star$.

Types

Strongscript provides programmer control over the soundness guarantee.
It adds a weak type

$$\mathsf{t} ::= \ldots \mid {}^{*}\{\overline{\mathsf{mt}}\}$$

Uses of this type are not strictly checked!

Statics

$$\frac{}{t_1 <: {}^{*}t_1} \qquad \frac{t_1 <: t_2}{{}^{*}t_1 <: {}^{*}t_2}$$

$$\frac{\Gamma \vdash e_1 \hookrightarrow e_1' \Rightarrow {}^{*}t_1 \qquad \overline{\Gamma \vdash e_2 \rightsquigarrow e_2' \Leftarrow \star}}{\Gamma \vdash e_1.m(\overline{e_2}) \hookrightarrow e_1'.m^{\star}(\overline{e_2'}) \Rightarrow \star} \qquad \frac{\Gamma \vdash e_1 \hookrightarrow e_2 \Rightarrow {}^{*}t_2}{\Gamma \vdash e_1 \rightsquigarrow \langle t_1 \rangle e_2 \Leftarrow t_1}$$

Statics

Same types as the base, new type checking.
We model module typing by requiring that a class be either fully untyped (no typed field or method declarations) or fully typed.
Typed code is restricted from creating new instances of untyped classes.

Cast insertion is now only ran on the untyped code.