How to Evaluate the Performance of Gradual Type Systems

Ben Greenman
Max S. New
Robert Bruce Findler
Matthias Felleisen

Asumu Takikawa
Daniel Feltey
Jan Vitek

Northwestern University
igalia
Northwestern University
Czech Technical University in Prague
CONTRIBUTION:

Our paper presents the first systematic \textit{method} to measure the \textit{performance implications} of a gradual typing system.
**NOTATION**

Program

Component

Dependency
Gradual Typing

Mixed-Typed Program

Statically-typed Component

Dynamically-typed Component

Type Boundary
Typed–Untyped Interaction
Typed–Untyped Interaction

need Integer
Typed–Untyped Interaction

need Integer

42
Typed–Untyped Interaction

need Integer

42
Typed–Untyped Interaction
Typed–Untyped Interaction

need Integer
Typed–Untyped Interaction

need \textbf{Integer} \quad \text{"NaN"}
Typed–Untyped Interaction

need Integer

"NaN"
Typed–Untyped Interaction
Typed–Untyped Interaction

need `Listof(String)`
Typed–Untyped Interaction

need `Listof(String)`

`(list "A" "B" 27)`
Typed–Untyped Interaction

need `Listof(String)`

(list "A" "B" 27)
Typed–Untyped Interaction
Typed–Untyped Interaction

need $\text{Bool} \rightarrow \text{Bool}$
Typed–Untyped Interaction

need $\text{Bool} \to \text{Bool}$

$\lambda$
Typed–Untyped Interaction

need \texttt{Bool->Bool}

\texttt{Bool?} (\texttt{Bool?})

\texttt{#<procedure>}

\texttt{\tau} \leftrightarrow \lambda
Typed–Untyped Interaction

Type boundaries impose a run-time cost!

(Some mixed-typed languages do not enforce types. For these languages, the performance of type boundaries is not an issue.)
Q. What is the overall cost of boundaries in a gradual typing system?
Q. What is the overall cost of boundaries in a \textit{gradual typing system}?

Need a \textbf{method} to measure and evaluate the performance implications of a gradual typing system.
The Method
**D-deliverable**

A configuration is **D-deliverable** if its performance is no worse than a factor of **D** slowdown compared to the baseline.

and \( D \rightarrow \) +\( \times \)
**Method:** exhaustive perf. eval.

1. Typed program

2. Measure all configurations

3. Count D-deliverable cfgs.

Repeat for other programs
A Method for Presenting the Data
Scaling the Method
Exponential Blowup

4 components

6 components

8 components
**Exponential Blowup**

\[ N \text{ components} \Rightarrow 2^N \text{ configurations} \]
SIMPLE RANDOM SAMPLING
SIMPLE RANDOM SAMPLING

1. Sample $O(N)$ configurations
   $\circ N = \text{number of components}$
Simple Random Sampling

1. Sample $O(N)$ configurations
   $\circ N =$ number of components

2. Count D-deliverable cfgs. in the sample
More in Paper
• justification for O(N) sampling
  ◦ N = number of components

• exhaustive method applied to Typed Racket
  ◦ 20 benchmarks, docs.racket-lang.org/gtp-benchmarks

• comparison: TR v6.2, v6.3, & v6.4
  ◦ the method quantifies improvements

• discussion of pathologies
Thank you

Sam Tobin-Hochstadt

For Typed Racket, and for significant improvements to v6.3, v6.4, and beyond.
How to Evaluate the Performance of Gradual Type Systems

Ben Greenman *  
Max S. New  
Robert Bruce Findler  
Matthias Felleisen  

Asumu Takikawa  
Daniel Feltey  
Jan Vitek