Three Approaches to Gradual Typing

Ben Greenman, Justin Pombrio, Matthias Felleisen, Preston Tunnell Wilson, Shriram Krishnamurthi, and many others
**Dynamic Typing**

value-level abstractions, enforced at run-time

**Static Typing**

type-level abstractions, checked before run-time

**Gradual Typing**

mix of static & dynamic typing ... somehow
**Dynamic Typing**

*value*-level abstractions, enforced at run-time

**Static Typing**

*type*-level abstractions, checked before run-time

**Gradual Typing**

mix of static & dynamic typing ... somehow
**Dynamic Typing**

*value-level abstractions*, enforced at run-time

**Static Typing**

*type-level abstractions*, checked before run-time

**Gradual Typing**

mix of static & dynamic typing ... somehow
**Dynamic Typing**
value-level abstractions, enforced at run-time

**Static Typing**
type-level abstractions, checked before run-time

**Gradual Typing**
mix of static & dynamic typing ... somehow
Gradual Typing is growing ...

Over 80 publications

Over 20 implementations
Gradual Typing is growing ...

Over 80 publications

Over 20 implementations

But NO common definition of gradual typing – due to different goals and priorities
Gradual Typing is growing ... 

Over 80 publications

Over 20 implementations

But NO common definition of gradual typing – due to different goals and priorities

Little acknowledgment (or analysis!) of the differences
One kind of gradual typing: Migratory Typing (SNAPL'17)

1. Begin with an existing, dynamically-typed language

2. Design an idiomatic type system

3. Allow interaction between the two languages
A few Migratory Typing systems

Gradualtalk  Typed Racket  TPD  Pycket

Pallene  Grace  SafeTS  Reticulated

mypy  Flow  Hack  Pyre  Pytype  rtc  MACLISP

Common Lisp  Strongtalk  TypeScript
Typed Clojure  Typed Lua
A few migratory typing systems

Deep
- Gradualtalk
- Typed Racket
- TPD
- Pycket

Shallow
- Pallene
- Grace
- SafeTS
- Reticulated

Erasure
- mypy
- Flow
- Hack
- Pyre
- Pytype
- rtc
- MACLISP
- Common Lisp
- Strongtalk
- TypeScript
- Typed Clojure
- Typed Lua
THREE APPROACHES TO MIGRATORY TYPING

- Deep
- Shallow
- Erasure
THREE APPROACHES TO MIGRATORY TYPING

Deep (behavioral)

Shallow (transient)

Erasure (optional)
THREE APPROACHES TO MIGRATORY TYPING

Deep (behavioral)

Shallow (transient)

Erasure (optional)

Three strategies for enforcing types at a boundary
THREE APPROACHES TO MIGRATORY TYPING

Deep

Shallow

Erasure
THREE APPROACHES TO MIGRATORY TYPING

Deep types are sound/enforced

Shallow

Erasure
THREE APPROACHES TO MIGRATORY TYPING

Deep: types are sound/enforced

Shallow: typed code cannot get stuck

Erasure
THREE APPROACHES TO MIGRATORY TYPING

Deep
- types are sound/enforced

Shallow
- typed code cannot get stuck

Erasure
- types do not affect behavior
Type Soundness (simplified):

\[ \text{if } \not\vdash e : t \text{ then either:} \]
- \( e \rightarrow^* v \) and \( \not\vdash v : t \)
- \( e \) diverges
- \( e \rightarrow^* \) Error
Type Soundness (simplified):
if $\vdash e : t$ then either:
- $e \rightarrow^* v$ and $\nvdash v : t$
- $e$ diverges
- $e \rightarrow^* \text{Error}$
Type Soundness (simplified):

if ⊢ e : t then either:
- e ->* v and ⊢ v : t
- e diverges
- e ->* Error
Type Soundness (simplified):

if ⊢ e : t then either:

- e →* v and ⊬ v : t
- e diverges
- e →* Error
Deep

if ⊢ e : t then either:
  - e →* v and ⊢ v : t
  - e diverges
  - e →* Error
Deep

if ⊢ e : t then either:
- e →* v and ⊬ v : t
- e diverges
- e →* Error

Shallow

if ⊢ e : t then either:
- e →* v and ⊬ v : C(t)
- e diverges
- e →* Error

Erasure
Deep

if ⊢ e : t then either:
- e →* v and ⊢ v : t
- e diverges
- e →* Error

Shallow

if ⊢ e : t then either:
- e →* v and ⊢ v : C(t)
- e diverges
- e →* Error

Erasure

if ⊢ e : t then either:
- e →* v and ⊢ v
- e diverges
- e →* Error
Is type soundness all-or-nothing?
Is type soundness all-or-nothing?

No! (in a mixed-typed language)
Implementation

Three compilers for the Typed Racket surface language

i.e. three ways of running the same code
How to measure performance?
Experiment

- 10 benchmark programs
- 2 to 10 modules each
- 4 to 1024 configurations each

[docs.racket-lang.org/gtp-benchmarks](docs.racket-lang.org/gtp-benchmarks)
Performance

Overhead vs. Untyped

Num. Type Annotations

- deep
- shallow
- erasure
Performance Implications
Performance Implications

- **Red** add types to remove all critical boundaries
- **Blue** add types sparingly
- **Green** add types anywhere, doesn’t matter
THREE APPROACHES TO MIGRATORY TYPING

Deep

Shallow

Erasure

Soundness
Performance
... Users?
Question 1

1  var t = [4, 4];
2  var x : Number = t;
3  x

Error: line 2 expected Number got [4, 4]  LE  LU  DE  DU
[4, 4]  ○  ○  ○  ○
Question 1

1 \[ \text{var } t = [4, 4]; \]
2 \[ \text{var } x : \text{Number} = t; \]
3 \[ x \]

Error: line 2 expected Number got [4, 4]

<table>
<thead>
<tr>
<th>S.E</th>
<th>Student</th>
<th>MTurk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep</td>
<td>Error: line 2 expected Number got [4, 4]</td>
<td></td>
</tr>
</tbody>
</table>

- **Deep** →* Error: line 2 expected Number got [4, 4]
- **Erasure** →* [4, 4]
- **Shallow** →* same as Deep

\[ \text{L = Like, D = Dislike, E = Expected, U = Unexpected} \]
Developer Survey

Asked software engineers, students, and MTurk workers to rate potential different behaviors for programs

Results show a preference for Deep

More at DLS Tuesday 10:30am The Loft

cs.brown.edu/research/plt/dl/dls2018