Deep and Shallow Types for Gradual Languages

Ben Greenman
2022-06-16

Northeastern
-> Brown*
-> Utah
Q. Should your PL be typed or untyped?
Q. Should your PL be typed or untyped?

**Gradual typing says yes to both**

"best" of two worlds
Great Idea!
Inspired MANY Languages Over 16+ Years
Great Idea!
Inspired MANY Languages Over 16+ Years
Great Idea!
Inspired MANY Languages Over 16+ Years

No agreement on the semantics of gradual types
4 leading semantics because of a tradeoff: type guarantees vs. performance costs vs. expressiveness
Static

C#

Nom

SafeTS

JS

TS*

StrS.

THORN

Concrete

limited interop w/ untyped

Natural

Grift

TPD

Transient

Grace

Ne

Erasure

AS

mypy

PyRE

hack

PyType

PYRIGHT

STRONGTALK

TS

Lua
Concrete

Limited interop w/ untyped

Natural

Grift
TPD

Erasure

Unsound interop
Starting Point

**Natural**
- strong, slow types
- Deep

**Transient**
- fast, wrong types
- Shallow
Starting Point

RQ. Can Natural and Transient interoperate?

Natural

strong, slow types

Deep

Transient

fast, wrong types

Shallow
RQ. Can Natural and Transient interoperate?

Motivations:
- ease the guarantees vs. performance tradeoff
- no loss of expressiveness; same static types
Starting Point

RQ. Can Natural and Transient interoperate?

**Motivations:**
- ease the guarantees vs. performance tradeoff
- no loss of expressiveness; same static types

Orthogonal to basic improvements:

- **Pycket**
  - OOPSLA'17
- **Corpse Reviver**
  - POPL'21
Key Technical Question:
How to Enforce Types at Boundaries?
Key Technical Question:
How to Enforce Types at Boundaries?

Deep Typed

Shallow Typed

Untyped

While preserving their formal properties

OOPSLA'19  ICFP'18
Key Technical Question:
How to Enforce Types at Boundaries?

While preserving their formal properties

OOPSLA'19  ICFP'18
Key Technical Question:
How to Enforce Types at Boundaries?

Type Soundness
Complete Monitoring
Deep Typed

? ?

Type Soundness
Shallow Typed

? ?

Untyped

While preserving their formal properties

OOPSLA'19  ICFP'18
Key Technical Question:
How to Enforce Types at Boundaries?

Type Soundness
Complete Monitoring
Deep Typed

Type Soundness
Shallow Typed

Dyn. Soundness
Untyped

While preserving their formal properties

OOPSLA'19  ICFP'18
Key Technical Question:
How to Enforce Types at Boundaries?
Key Technical Question:
How to Enforce Types at Boundaries?

First of all:
Q. How does **Natural** enforce **Deep** types?
Q. How does **Transient** enforce **Shallow** types?
Q. How does Natural enforce Deep types?
Q. How does Natural enforce Deep types?

A. Use wrappers to guard boundaries
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries.
Q. How does Natural enforce Deep types?

A. Use wrappers to guard boundaries
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries

```
Q. How does Natural enforce Deep types?

A. Use wrappers to guard boundaries

Deep Typed  ────► Untyped

Int -> Int  ◀◄ fun x . e

wrap    ◀◄ Vectorof Int

vec 1 2 3  ◀◄ wrap
```
Q. How does **Transient** enforce **Shallow** types?
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks
Q. How does Transient enforce Shallow types?

A. With no wrappers but many tiny shape checks

Check function calls, vector refs, etc.
Deep Typed → 1. wrap → Untyped
Shallow Typed → 1. wrap → Untyped
1. wrap

Deep Typed

Shallow Typed

2. shape check

Untyped
Deep Typed

1. wrap

Shallow Typed

2. shape check

Untyped
1. **wrap**

2. **shape check**

**Typed to Typed = no check?**

**No!**
What If: No Checks Between Deep and Shallow

Example 1:
Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```python
def f0(n : Int):
    n + 2
```
What If: No Checks Between Deep and Shallow

Example 1:
Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```python
def f0(n : Int):
    n + 2
```

sends it to untyped code ...

```python
def f1 = f0
```
What If: No Checks Between Deep and Shallow

Example 1:
Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,
```
def f0(n : Int):
    n + 2
```
sends it to untyped code ...
```
def f1 = f0
```
and back, with a new type.
```
f2 : Str -> Str
def f2 = f1
```
What If: No Checks Between Deep and Shallow

Example 1:
Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,
```
def f0(n : Int):
    n + 2
```
sends it to untyped code ...
```
def f1 = f0
```
and back, with a new type.
```
def f2 = f1
```

Types say \( f2 : \text{Str} \rightarrow \text{Str} \)
Checks say \( f2 \) is a function
What If: No Checks Between Deep and Shallow

Example 1:
Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```
def f0(n : Int):
    n + 2
```

sends it to untyped code ...

```
def f1 = f0
```

and back, with a new type.

```
f2 : Str -> Str
def f2 = f1
```

Deep gets a 'bad' function

```
f3 : Str -> Str
def f3 = f2
```
What If: No Checks Between Deep and Shallow

Deep makes a function,

\[
\text{def } \ g0(h : \text{Int} \rightarrow \text{Int}): \\
\quad h(3)
\]

Example 2:
Shallow can send a Deep value to Untyped code
What If: No Checks Between Deep and Shallow

Example 2:
Shallow can send a Deep value to Untyped code

Deep makes a function,
```
def g0(h : Int -> Int):
    h(3)
```
sends it to Shallow,
```
g1 : (Int -> Int) -> Int
def g1 = g0
```
What If: No Checks Between Deep and Shallow

Example 2:
Shallow can send a Deep value to Untyped code

Deep makes a function,
```python
def g0(h : Int -> Int):
    h(3)
```
sends it to Shallow,
```python
g1 : (Int -> Int) -> Int
def g1 = g0
```
which sends it to untyped
```python
def g2 = g1
g2("not a function")
```
1. wrap

Typed to Typed = no check?
No!

2. shape check

Deep Typed

Shallow Typed

Untyped
Deep Typed

Shallow Typed

Untyped

1. wrap

2. shape check

3. wrap
In paper: model, type soundness, complete monitoring

- Deep Typed
- Shallow Typed
- Untyped

1. wrap
2. shape check
3. wrap
Implementation

Typed Racket

Deep Typed

Shallow Typed

Untyped
Implementation

Typed Racket

A Transient Semantics for Typed Racket
Programming'22

Deep Typed

Shallow Typed

Untyped
Implementation

Deep Typed

Shallow Typed

Untyped

In paper: general lessons (no macros)
Evaluation

Guarantees vs. Performance vs. Expressiveness
Evaluation

Guarantees vs. Performance vs. Expressiveness
GTP Benchmarks
21 programs

docs.racket-lang.org/gtp-benchmarks
GTP Benchmarks
21 programs

docs.racket-lang.org/gtp-benchmarks

Ex: One program with 3 components
GTP Benchmarks
21 programs

Ex: One program with 3 components

8 Typed / Untyped points ($2^N$)
GTP Benchmarks
21 programs

docs.racket-lang.org/gtp-benchmarks

Ex: One program with 3 components

8 Typed / Untyped points (2^N)

27 Deep / Shallow / Untyped points (3^N)
Better Performance

Q. How many points run fastest with a Deep + Shallow mix?
Better Performance

Q. How many points run fastest with a Deep + Shallow mix?
Better Performance

Q. How many points run fastest with a Deep + Shallow mix?

<table>
<thead>
<tr>
<th>Program</th>
<th>Deep (%)</th>
<th>Shallow (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>forth</td>
<td>12%</td>
<td>zordoz 47%</td>
</tr>
<tr>
<td>fsm</td>
<td>38%</td>
<td>lnm 66%</td>
</tr>
<tr>
<td>fsmoo</td>
<td>31%</td>
<td>suffixtree 48%</td>
</tr>
<tr>
<td>mbta</td>
<td>19%</td>
<td>kcfa 55%</td>
</tr>
<tr>
<td>morsecode</td>
<td>25%</td>
<td>snake 46%</td>
</tr>
<tr>
<td>zombie</td>
<td>6%</td>
<td>take5 36%</td>
</tr>
<tr>
<td>dungeon</td>
<td>31%</td>
<td>acquire 64%</td>
</tr>
<tr>
<td>jpeg</td>
<td>38%</td>
<td>tetris 62%</td>
</tr>
</tbody>
</table>
Better Performance

Q. What is the worst-case overhead?

Deep or Shallow
**Better Performance**

Q. What is the worst-case overhead?

<table>
<thead>
<tr>
<th></th>
<th>Deep</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>sieve</td>
<td>2.97x</td>
<td></td>
</tr>
<tr>
<td>forth</td>
<td>5.43x</td>
<td></td>
</tr>
<tr>
<td>fsm</td>
<td>1.91x</td>
<td></td>
</tr>
<tr>
<td>fsmoo</td>
<td>4.25x</td>
<td></td>
</tr>
<tr>
<td>mbta</td>
<td>1.71x</td>
<td></td>
</tr>
<tr>
<td>morsecode</td>
<td>1.3x</td>
<td></td>
</tr>
<tr>
<td>zombie</td>
<td>31x</td>
<td></td>
</tr>
<tr>
<td>dungeon</td>
<td>3.16x</td>
<td></td>
</tr>
<tr>
<td>jpeg</td>
<td>1.56x</td>
<td></td>
</tr>
<tr>
<td>zordoz</td>
<td>2.58x</td>
<td></td>
</tr>
<tr>
<td>lnm</td>
<td>1.17x</td>
<td></td>
</tr>
<tr>
<td>suffixtree</td>
<td>5.8x</td>
<td></td>
</tr>
<tr>
<td>kcfa</td>
<td>1.24x</td>
<td></td>
</tr>
<tr>
<td>snake</td>
<td>7.61x</td>
<td></td>
</tr>
<tr>
<td>take5</td>
<td>2.97x</td>
<td></td>
</tr>
<tr>
<td>acquire</td>
<td>1.42x</td>
<td></td>
</tr>
<tr>
<td>tetris</td>
<td>5.44x</td>
<td></td>
</tr>
<tr>
<td>synth</td>
<td>4.2x</td>
<td></td>
</tr>
<tr>
<td>gregor</td>
<td>1.51x</td>
<td></td>
</tr>
<tr>
<td>quadT</td>
<td>7.23x</td>
<td></td>
</tr>
<tr>
<td>quadU</td>
<td>7.45x</td>
<td></td>
</tr>
</tbody>
</table>
### Better Performance

Q. What is the worst-case overhead?

<table>
<thead>
<tr>
<th>Program</th>
<th>Deep</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>sieve</td>
<td>2.97x</td>
<td>16x</td>
</tr>
<tr>
<td>forth</td>
<td>5.43x</td>
<td>5800x</td>
</tr>
<tr>
<td>fsm</td>
<td>1.91x</td>
<td>2.24x</td>
</tr>
<tr>
<td>fsmoo</td>
<td>4.25x</td>
<td>420x</td>
</tr>
<tr>
<td>mbta</td>
<td>1.71x</td>
<td>1.91x</td>
</tr>
<tr>
<td>morsecode</td>
<td>1.3x</td>
<td>1.57x</td>
</tr>
<tr>
<td>zombie</td>
<td>31x</td>
<td>46x</td>
</tr>
<tr>
<td>dungeon</td>
<td>3.16x</td>
<td>15000x</td>
</tr>
<tr>
<td>jpeg</td>
<td>1.56x</td>
<td>23x</td>
</tr>
<tr>
<td>zordoz</td>
<td>2.58x</td>
<td>2.63x</td>
</tr>
<tr>
<td>lnms</td>
<td>1.17x</td>
<td>1.23x</td>
</tr>
<tr>
<td></td>
<td>Deep</td>
<td>or</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>----</td>
</tr>
<tr>
<td>sieve</td>
<td>2.97x</td>
<td>16x</td>
</tr>
<tr>
<td>forth</td>
<td>5.43x</td>
<td>5800x</td>
</tr>
<tr>
<td>fsm</td>
<td>1.91x</td>
<td>2.24x</td>
</tr>
<tr>
<td>fsmoo</td>
<td>4.25x</td>
<td>420x</td>
</tr>
<tr>
<td>mbta</td>
<td>1.71x</td>
<td>1.91x</td>
</tr>
<tr>
<td>morsecode</td>
<td>1.3x</td>
<td>1.57x</td>
</tr>
<tr>
<td>zombie</td>
<td>31x</td>
<td>46x</td>
</tr>
<tr>
<td>dungeon</td>
<td>3.16x</td>
<td>15000x</td>
</tr>
<tr>
<td>jpeg</td>
<td>1.56x</td>
<td>23x</td>
</tr>
<tr>
<td>zordoz</td>
<td>2.58x</td>
<td>2.63x</td>
</tr>
<tr>
<td>lnm</td>
<td>1.17x</td>
<td>1.23x</td>
</tr>
</tbody>
</table>

H.O. values and many elim. forms

Q. What is the worst-case overhead?
Better Performance
Better Performance

Overall: switching between Deep and Shallow can avoid perf. bottlenecks

- **Deep** near the top, to maximize the benefits of types
- **Shallow** in the middle, to minimize the cost of boundaries
Conclusion
Context: Different GT strategies exist (for good reason!)
Context: Different GT strategies exist (for good reason!)

Inquiry: Can two extreme strategies interoperate?

- Deep types via Natural (wrappers)
- Shallow types via Transient (no wrappers)
Context: Different GT strategies exist (for good reason!)

Inquiry: Can two extreme strategies interoperate?
- **Deep** types via **Natural** (wrappers)
- **Shallow** types via **Transient** (no wrappers)

Contribution: Yes! In a way that:
- preserves their formal guarantees
- leads to better overall performance
- lets TR express additional programs
A New Dimension for Gradual Typing

- Deep
- Shallow
- Untyped
A New Dimension for Gradual Typing
A New Dimension for Gradual Typing

Q. More regions along the spectrum?
A New Dimension for Gradual Typing

Q. More regions along the spectrum?

Q. Better cooperation b/w Deep and Shallow?
A New Dimension for Gradual Typing

Q. More regions along the spectrum?
Q. Better cooperation b/w Deep and Shallow?
Q. Solve the N^2 interop problem?
The End

Coming soon to Racket
https://racket-lang.org

Pull Request  https://github.com/racket/typed-racket/pull/948
Research Repo  https://github.com/bennn/g-pldi-2022