The Behavior of Gradual Types: A User Study

Preston Tunnell Wilson¹
Ben Greenman² ★
Justin Pombrio¹
Shriram Krishnamurthi¹
Intuition: Gradual Typing
Intuition: Gradual Typing

Dynamic Typing

un(i)typed components
value safe
Intuition: Gradual Typing

Dynamic Typing

un(i)typed components
value safe

Static Typing

typed components
type safe
Intuition: Gradual Typing

**Dynamic Typing**
- un(i)typed components
- value safe

**Static Typing**
- typed components
- type safe

**Gradual Typing**
- typed + untyped components
- .... safe?
Gradual Typing Helps Programmers?

Combining static and dynamic typing within the same language offers clear benefits to programmers.
We conjecture that a programmer would like the guarantee that the values produced by their components are never used in violation to the interface specifications ...
Gradual Typing Helps Programmers?

... [run-time checks] inspect the top-level type (or type-tag) of each value, ensuring safe interaction and providing the expected type safety to programmers.
A programmer may favour unsound monitoring over wrappers that change the semantics of their program.
Gradual Typing Helps Programmers?

Being sound, Safe TypeScript endows types with many of the properties that Java or C# programmers might expect but not find in TypeScript.

The system lives up to all expectations that developers have of sound language implementations.

... programmers should be able to add or remove type annotations without any unexpected impacts on their program.
Data to Support Claims?
Data to Support Claims?

DLS'14  
ESOP'12  
POPL'17  
ECOOP'17  
POPL'15  
SNAPL'17  
SNAPL'15  
POPL'08  
OOPSLA'17  
ECOOP'14
Three Different Approaches!

- Deep
- Shallow
- Erasure

DLS'14
ESOP'12
POPL'17
ECOOP'17
POPL'15
SNAPL'17
SNAPL'15
POPL'08
OOPSLA'17
ECOOP'14
Three Different Approaches!

Deep types are sound/enforced

Shallow

Erasure
Three Different Approaches!

- **Deep**
  - Types are sound/enforced

- **Shallow**
  - Typed code cannot get stuck

- **Erasure**
Three Different Approaches!

Deep

- types are sound/enforced

Shallow

- typed code cannot get stuck

Erasure

- types do not affect behavior
Three Different Approaches!

- Deep: types are sound/enforced
- Shallow: typed code cannot get stuck
- Erasure: types do not affect behavior

- DLS'14
- ESOP'12
- POPL'17
- ECOOP'17
- POPL'15
- SNAPL'17
- SNAPL'15
- POPL'08
- OOPSLA'17
- ECOOP'14
In this paper:

We begin to address the lack of data with a developer survey contrasting the different approaches to gradual typing

Deep vs. Shallow Erasure
Survey based on 8 example programs
Survey Prompt

We are designing a language that mixes typed and untyped code.
Survey Prompt

We are designing a language that mixes typed and untyped code.

We want your opinion on what should happen when untyped values flow into typed expressions.
We are designing a language that mixes typed and untyped code.

We want your opinion on what should happen when untyped values flow into typed expressions.

Our language has static type checking but does not have type inference ... The following [programs] pass the static type checker.
Survey Prompt

We are designing a language that mixes typed and untyped code.
We want your opinion on what should happen when untyped values flow into typed expressions.
Our language has static type checking but does not have type inference ... The following
[programs] pass the static type checker.

We are not looking for feedback on syntax.
We are designing a language that mixes typed and untyped code.

We want your opinion on what should happen when untyped values flow into typed expressions. Our language has static type checking but does not have type inference ... The following [programs] pass the static type checker. We are not looking for feedback on syntax.
Question 7

```javascript
1 var x : Array(String) = ["hi" "bye"];
2 var y = x;
3 var z : Array(Number) = y;
4 z[0] = 42;
5 var a : Number = z[1];
6 a
```
Question 7

```javascript
var x : Array(String) = ["hi", "bye"];
var y = x;
var z : Array(Number) = y;
z[0] = 42;
var a : Number = z[1];
a
```
Question 7

```javascript
1 | var x : Array(String) = ["hi" "bye"];  
2 | var y = x;  
3 | var z : Array(Number) = y;  
4 | z[0] = 42;  
5 | var a : Number = z[1];  
6 | a
```
Question 7

```javascript
1 var x : Array(String) = ["hi" "bye"];  
2 var y = x;  
3 var z : Array(Number) = y;  
4 z[0] = 42;  
5 var a : Number = z[1];  
6 a
```
Question 7

```javascript
var x : Array(String) = ["hi" "bye"];  
var y = x;  
var z : Array(Number) = y;  
z[0] = 42;  
var a : Number = z[1];
```

a
Question 7

```
1 var x : Array(String) = ["hi" "bye"];
2 var y = x;
3 var z : Array(Number) = y;
4 z[0] = 42;
5 var a : Number = z[1];
6 a

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye"
"bye"
```
Question 7

```
1 var x: Array(String) = ["hi" "bye"];  
2 var y = x;  
3 var z: Array(Number) = y;  
4 z[0] = 42;  
5 var a: Number = z[1];  
6 a
```

Error: line 4 expected String got 42  
Error: line 5 expected Number got "bye"  
"bye"
Question 7

1 | var x : Array(String) = ["hi" "bye"];  
2 | var y = x;  
3 | var z : Array(Number) = y;  
4 | z[0] = 42;  
5 | var a : Number = z[1];  
6 | a

Error: line 4 expected String got 42  
Error: line 5 expected Number got "bye" "bye"
1 var x: Array(String) = ["hi" "bye" ];
2 var y = x;
3 var z : Array(Number) = y;
4 z[0] = 42;
5 var a : Number = z[1];
6 a

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye" "bye"
Question 7

```javascript
var x: Array(String) = ["hi" "bye"];
var y = x;
var z: Array(Number) = y;
z[0] = 42;
var a: Number = z[1];
a
Error: line 4 expected String got 42
Error: line 5 expected Number got "bye"
"bye"
```

<table>
<thead>
<tr>
<th>Expected</th>
<th>Like</th>
<th>Dislike</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 7

```
1  var x : Array(String) = ["hi" "bye" ];
2  var y = x;
3  var z : Array(Number) = y;
4  z[0] = 42;
5  var a : Number = z[1];
6  a
```

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye"
"bye"
Question 5

```javascript
var obj0 = {
    k = 0;
    add = function(i : Number) { k = i };
};
var t = "hello";
obj0.add(t);
var k : String = obj0.k;
k

Error: line 1 expected Number got "hello"
"hello"
```

Two distinct behaviors
Question 5

Two distinct behaviors

```javascript
var obj0 = {
  k = 0;
  add = function(i : Number) { k = i };
};
var t = "hello";
obj0.add(t);
var k : String = obj0.k;
k

Error: line 1 expected Number got "hello"
"hello"
(Deep, Shallow)
(Erasure)
```
var obj0 = {
    k = 0;
    add = function(i : Number) { k = i };
}
var t = "hello";
obj0.add(t);
var k : String = obj0.k;
k

Error: line 1 expected Number got "hello"
Question 5

```
1 var obj0 = {
   k = 0;
   add = function(i : Number) { k = i };
2 var t = "hello";
3 obj0.add(t);
4 var k : String = obj0.k;
5 k
```

Error: line 1 expected Number got "hello" "hello"
Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.
Summary: Survey Design

Solicit opinions on the semantics of a "new" language

Gather Like × Expect preference on eight small programs designed to classify approaches

Explicit followup about optional typing

Survey Prompt

We are designing a language that mixes typed and untyped code.

Question N

```
var x : Array(String) = ["hi" "bye"];  
var y = x;  
var z : Array(Number) = y;  
z[0] = 42;  
var a : Number = z[1];  
a
```

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye" "bye"

Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.
Summary: Survey Design

Solicit opinions on the semantics of a "new" language

Gather Like × Expect preference on eight small programs designed to classify approaches

Explicit followup about optional typing

Survey Prompt

We are designing a language that mixes typed and untyped code.

Question N

```javascript
1 var x : Array(String) = ["hi" "bye"];  
2 var y = x;  
3 var z : Array(Number) = y;  
4 z[0] = 42;  
5 var a : Number = z[1];  
6 a
```

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye"

"bye"

Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.
Goal: cover all interactions
Survey Design

Solicit opinions on the semantics of a "new" language

Gather *Like* × *Expect* preference on eight small programs designed to classify approaches

Explicit followup about optional typing

Survey Prompt

We are designing a language that mixes typed and untyped code.

Question N

```plaintext
1 var x : Array(String) = ["hi" "bye"];  0 0 0 0
2 var y = x;                            0 0 0 0
3 var z : Array(Number) = y;            0 0 0 0
4 z[0] = 42;                            0 0 0 0
5 var a : Number = z[1];               0 0 0 0
6 a                                   0 0 0 0
```

Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.
Distribution
Distribution

- Software Engineers
- CS Students
- MTurk workers
Distribution

- **34 participants**
- **17 participants**
- **90 participants (96 filtered)**
Results
Question 1

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

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

<table>
<thead>
<tr>
<th>LE</th>
<th>LU</th>
<th>DE</th>
<th>DU</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[4, 4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
</tr>
</tbody>
</table>
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]
Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.
Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.

Software Engineers: 44% agree
CS Students: 12% agree
MTurk workers: 51% agree
Conclusions

Deep \rightarrow \text{Liked + Expected} \quad \checkmark
Conclusions

Deep → Liked + Expected
Shallow
Erasure → Disliked + Unexpected
Conclusions

**Conjecture:** programmers would Like + Expect *correct blame*.

- **Deep** → **Liked + Expected** ✓
- **Shallow** → **Disliked + Unexpected** ×
- **Erasure** → **Liked + Expected** ×
Takeaways

Unless there’s a strong reason, choose Deep

Programmers seem to expect it!

Non-Deep languages must document their design and rationale

Start with the survey examples

[cs.brown.edu/research/plt/dl/dls2018]
Threats to Validity

Indirect questions ("new" language)

Possible ambiguity:
- lack of type inference
- interpretation of code / error outputs
- runtime vs. static errors
Threats to Generalizability

Selective engineer + student populations

Very diverse MTurk population

Other implications:
  runtime performance
  quality of error messages
Concrete Types

concrete = every value carries a runtime type

Limits expressiveness of "untyped" code

Preferred by Dart users?

Another point to explore!
Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.

- **Software Engineers**: 44% agree
- **CS Students**: 12% agree
- **MTurk workers**: 51% agree
Followup Question

Agree/Disagree: Type annotations should not change the behavior of a program.

<table>
<thead>
<tr>
<th>Pop.</th>
<th>Erasure Opinion</th>
<th>Agree</th>
<th>Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.E.</td>
<td>Like</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Dislike</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>Student</td>
<td>Like</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Dislike</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>MTurk</td>
<td>Like</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Dislike</td>
<td>18</td>
<td>25</td>
</tr>
</tbody>
</table>
Question 5

```javascript
var obj0 = {
    k = 0,
    add = function(i : Number) { k = i });
var t = "hello";
obj0.add(t);
var k : String = obj0.k;
k
```

Error: line 1 expected Number got "hello"
Question 5

```
var obj0 = {
    k = 0,
    add = function(x) { x = x + k; return x; }
}
var t = "hello"
obj0.add(t);
var k;  // error
```

Error: line 1 expected "hello"
Why is the compiler complaining about line 1? The error message should be attached to **line 3**, that’s the source of the problem!
Why is the compiler complaining about line 1? The error message should be attached to **line 3**, that’s the source of the problem!
Why is the compiler complaining about line 1? The error message should be attached to line 3, that's the source of the problem!
Question 7

```javascript
1 | var x : Array(String) = ["hi", "bye"];  
2 | var y = x;  
3 | var z : Array(Number) = y;  
4 | z[0] = 42;  
5 | var a : Number = z[1];  
6 | a

Error: line 4 expected String got 42
Error: line 5 expected Number got "bye" "bye"
```

LE  LU  DE  DU
Question 7

```
1 var x : Array(String) = ["hi", "bye"];  
2 var y = x;                                
3 var z : Array;                           
4 z[0] = 42;                               
5 var a : Number;                          
6 a
```

Error: line 4 expected String got 42
Error: line 5 expected "bye"
Shallow cannot get 'stuck' if:

1. Total reduction relation for dynamic code

2. Partial reduction relation for static code (possible to get stuck)

3. Shallow checks can distinguish stuck vs. non-stuck states
Three Approaches to G.T.
How to enforce the type boundaries?
Example: Base Type

Deep       Shallow       Erasure

"hello"       ?       Int
Example: Base Type

Deep  Shallow  Erasure

"hello"  \(\rightarrow\)  Int
Example: Base Type

Deep | Shallow | Erasure

"hello" × Int
Example: Base Type

Deep  Shallow  Erasure

"hello"  "hello"  Int
Example: Higher-Order Type

Deep  Shallow  Erasure

string-trim  ?  Int->Int
Example: Higher-Order Type

Deep \hspace{1cm} Shallow \hspace{1cm} Erasure

\[ \lambda(x) \ldots \]

\textcolor{red}{\text{string-clip}} \hspace{0.5cm} \rightarrow \hspace{0.5cm} \textcolor{cyan}{\text{Int} \rightarrow \text{Int}}
Example: Higher-Order Type

Deep    Shallow    Erasure

string-trim    string-trim

Int->Int
Example: Higher-Order Type

Deep    Shallow    Erasure

string-trim  string-trim  Int->Int
Example: Inductive Type

Deep  Shallow  Erasure

(1, "A") → ? → Int×Int
Example: Inductive Type

Deep  Shallow  Erasure

↑

(1, "A")  Int×Int
Example: Inductive Type

Deep  Shallow  Erasure

(1, "A")  (1, "A")  Int × Int
Example: Inductive Type

Deep  Shallow  Erasure

(1, "A")  (1, "A")  Int×Int
## Three Approaches, Summary

<table>
<thead>
<tr>
<th></th>
<th>Deep</th>
<th>Shallow</th>
<th>Erasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>invariant</td>
<td>t</td>
<td>C(t)</td>
<td>v</td>
</tr>
<tr>
<td>base types</td>
<td>check</td>
<td>check</td>
<td>-</td>
</tr>
<tr>
<td>coinductive types</td>
<td>wrap</td>
<td>check</td>
<td>-</td>
</tr>
<tr>
<td>inductive types</td>
<td>traverse</td>
<td>check</td>
<td>-</td>
</tr>
<tr>
<td>boundaries</td>
<td>static, higher-order</td>
<td>static, selectors</td>
<td>-</td>
</tr>
</tbody>
</table>