

# Reliable Numerical Design for ML via PL

Ben Greenman

Ganesh Gopalakrishnan

Taylor Allred

Xinyi Li

(Prof) Ben Greenman

**Interests:** PL, Gradual Typing, Human Factors



(Prof) Ganesh Gopalakrishnan

**Interests:** SW Correctness, Formal Methods



Taylor Allred

**Interests:** PL, Julia Runtime Verification



Xinyi Li

**Interests:** FP Exceptions, debugging NNs



# Reliable Numerical Design for ML via PL

But first — a long introduction to PL

# Programming Languages

---

# Programming Languages

---

Elegant Abstractions + Efficient Implementation

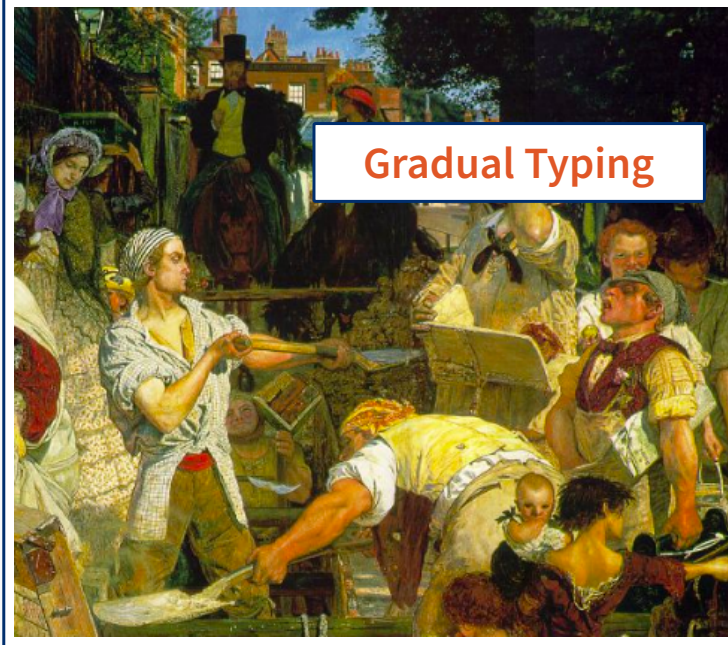
# Programming Languages

Elegant Abstractions + Efficient Implementation



# Programming Languages

Elegant Abstractions + Efficient Implementation



**Gradual Typing**



## Gradual Typing

Typed

||

Untyped

Should your language be typed or untyped?

## Gradual Typing

Typed

||

Untyped

Should your language be typed or untyped?

**Why not both?**

Example: TypeScript



## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

**Good!**

( Very useful in BIG programs )

## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

Untyped client

```
mag(["hello", "world"])
```

## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

Untyped client

```
mag(["hello", "world"])
```

**Now what?!**



## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

```
mag([15, -4])
```

Untyped client

```
mag(["hello", "world"])
```

**Now what?!**

```
sqrt("hello"**2 + "world"**2)
```

## Example: TypeScript



Typed function

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

Untyped client

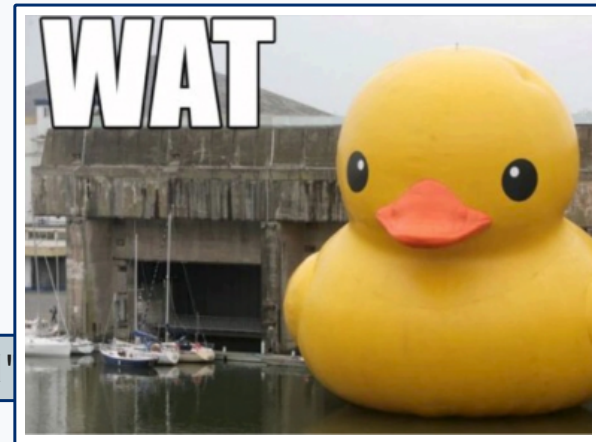
```
mag([15, -4])
```

Untyped client

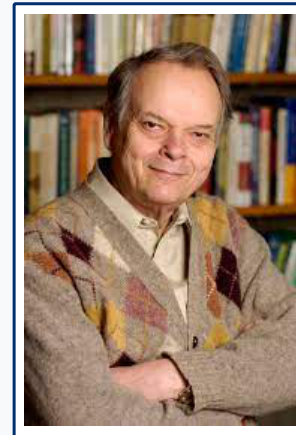
```
mag(["hello", "world"])
```

**Now what?!**

```
sqrt("hello"**2 + "world"
```



"Type structure is a syntactic discipline  
for **enforcing** levels of abstraction"



"Type structure is a syntactic discipline  
for **enforcing** levels of abstraction"

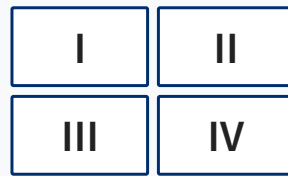
Not in TypeScript!





RQ. How to bring **sound gradual types** into practice?

RQ. How to bring **sound gradual types** into practice?





Build a Language




I

## Build a Language

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

```
mag(["hello", "world"])
```

# I Build a Language



```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);  
mag(["hello", "world"])
```

# I Build a Language

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);  
mag(["hello", "world"])
```



Typed Racket

**Enforce types** at boundaries  
with higher-order contracts

# I Build a Language

```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);  
mag(["hello", "world"])
```



Typed Racket

**Enforce types** at boundaries  
with higher-order contracts

Researchers need to pursue ideals. Nobody else can!

## I Build a Language

But ... research can fail



Build a Language

# I Build a Language

Design, Test, Deploy, .... Major Problem!



# I Build a Language

Design, Test, Deploy, .... Major Problem!



12000x slowdown

warning on use trie functions in #lang racket?



**johnbclements**  
to Racket Users

This program constructs a trie containing exactly two keys; each appears to be  $n^2$  in the length of the key, so doubling it to 256

**Huge cost** at boundaries!





## Measure Performance



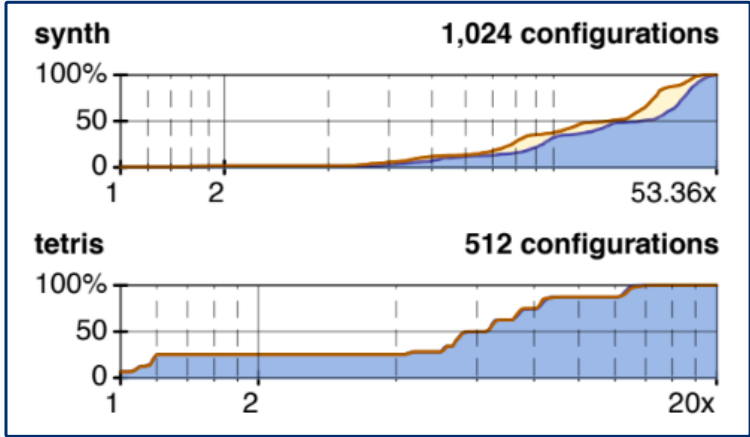
## Measure Performance

RQ. **Which boundaries** are slow  
and **what can we do** about it?

## II

# Measure Performance

RQ. **Which boundaries** are slow  
and **what can we do** about it?





## Measure Performance

II

## Measure Performance

**Is Sound Gradual Typing Dead?**

POPL '16

II

## Measure Performance

**Is Sound Gradual Typing Dead?**

POPL '16

**Sound Gradual Typing is Nominally Alive and Well**

OOPSLA '17

II

## Measure Performance

**Is Sound Gradual Typing Dead?**

POPL '16

**Sound Gradual Typing is Nominally Alive and Well**

OOPSLA '17

**Transient Typechecks are (Almost) Free**

ECOOP '19

II

## Measure Performance

**Is Sound Gradual Typing Dead?**

POPL '16

**Sound Gradual Typing is Nominally Alive and Well**

OOPSLA '17

**Transient Typechecks are (Almost) Free**

ECOOP '19

**Sound Gradual Typing: Only Mostly Dead**

OOPSLA '17





## Revisit the Design


## III Revisit the Design

More than one way to have *sound* gradual types

### III

## Revisit the Design

More than one way to have *sound* gradual types




```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

```
mag(["hello", "world"])
```

### III

## Revisit the Design

More than one way to have *sound* gradual types



```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```


```
mag(["hello", "world"])
```

Q. Any ideas?

## III

## Revisit the Design

More than one way to have *sound* gradual types



```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

```
mag(["hello", "world"])
```


Q. Any ideas?

- Deep checks at boundaries
- Shallow checks within typed code
- Type Tags on values

## III

## Revisit the Design

More than one way to have *sound* gradual types



```
function mag(xy: number[])  
  return sqrt(xy[0]**2 + xy[1]**2);
```

```
mag(["hello", "world"])
```

Q. Any ideas?

- Deep checks at boundaries
- Shallow checks within typed code
- Type Tags on values

**Tradeoff** between guarantees and performance



## Revisit the Design

III

## Revisit the Design



Typed Racket v8.6

**Deep or Shallow** types at boundaries





?!

Researchers need to pursue ideals. Nobody else can!

?!

Researchers need to pursue ideals. Nobody else can!

**Must find important questions**

IV

Get Feedback

# IV

## Get Feedback

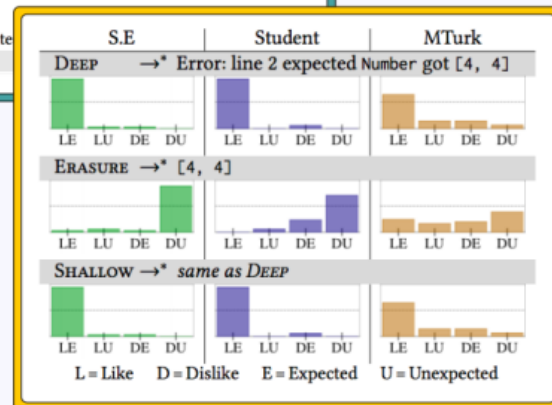


Typed Racket Survey

### Question 1

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

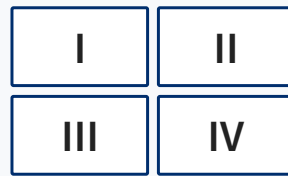
Error: line 2 expected  
[4, 4]





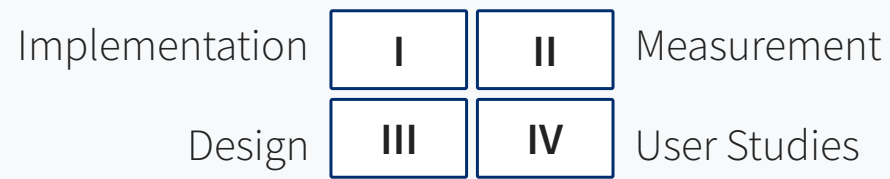
RQ. How to bring **sound gradual types** into practice?

RQ. How to bring **sound gradual types** into practice?





RQ. How to bring **sound gradual types** into practice?



Reliable Numerical Design for ML via PL

Problem: silent failures in numeric code  
( float != Real )

Problem: silent failures in numeric code  
( `float != Real` )

Approach: dynamic analysis

Problem: silent failures in numeric code  
( `float != Real` )

Approach: dynamic analysis

---

RQ. **What types** do we need  
to gradually harden numeric code?

