Type Theory 101

Type Theory For Absolute beginners

Hi! I'm Hanneli (@hannelita)

- Computer Engineer
- Programming
- Electronics
- Math <3 <3
- Physics
- Lego
- Meetups
- Animals
- Coffee
- Pokémon
- GIFs

CODE MINER

Why 'Type Theory?'

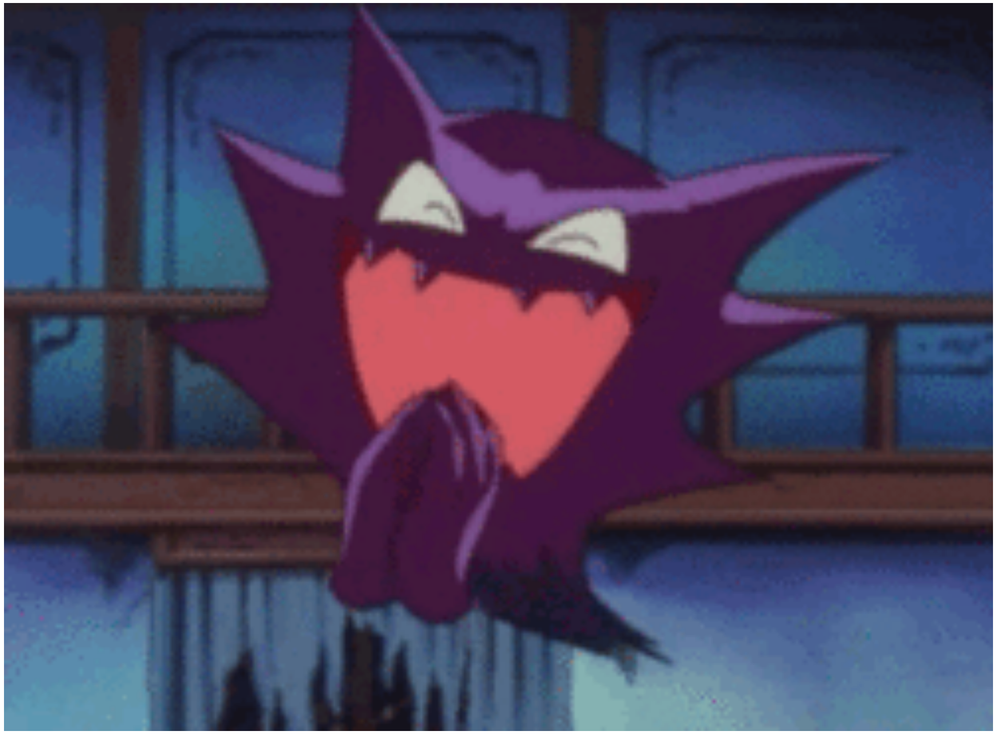
- Frameworks and architecture are important topics
- But what are the boundaries of computer science?
- We need theory to improve our practical tools.

Why 'Type Theory?'

*In mathematics, logic, and computer science, a **type theory** is any of a class of formal systems, some of which can serve as alternatives to set theory as a foundation for all mathematics. In type theory, every "term" has a "type" and operations are restricted to terms of a certain type.*
Type theory is closely related to (and in some cases overlaps with) type systems.

https://en.wikipedia.org/wiki/Type_theory

Why 'Type Theory?'



Disclaimer

Quick session

Lots of theory

And mathematics

No advanced Type Theory

GIFs :)

Goals

Understand what type theory is about

Understand how can we jump from language analysis to mathematics (it is not magic)

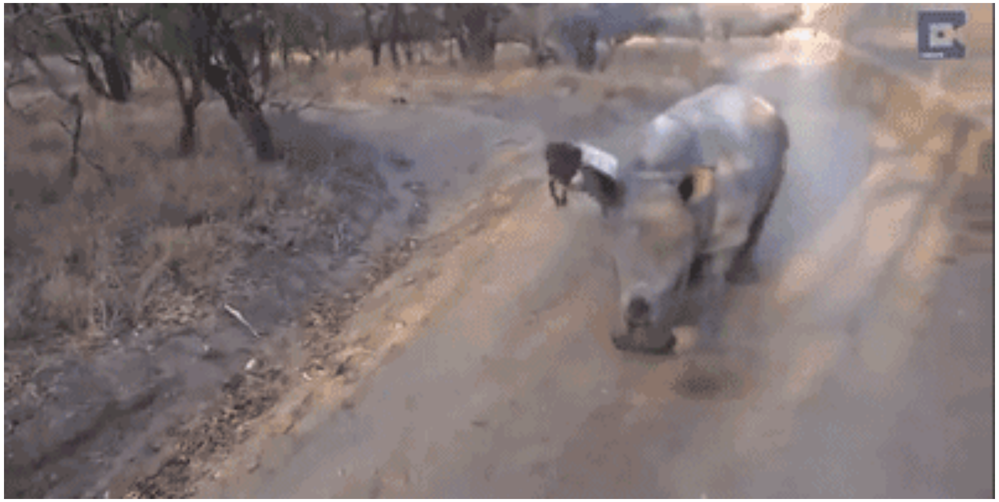
Understand some benefits of this analysis

Agenda

- **Choosing a programming language**
- Quick intro about type systems
- Sketching the possible types
- Symbolic Logic analysis
- Predicate logic
- Getting there!
- Why is this important?
- Challenges

How do you choose a programming language?

- By company
- By popularity
- By team
- By deadline to deliver a project
- By project goal
- By available tools
- That was the only language I learned at school
- Somebody told me I should use that
- I really don't know



How often do you consider the following items when choosing a language?

- Type System
- Immutability
- Avoidance to runtime errors
- Paradigm
- Verbosity
- Memory management

Wait - what is a type system?

Let's ask Wikipedia:

"In programming languages, a ***type system*** *is a collection of rules that assign a property called type to various constructs a computer program consists of,* such as variables, expressions, functions or modules*"*

Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- Sketching the possible types
- Symbolic Logic analysis
- Predicate logic
- Getting there!
- Why is this important?
- Challenges

Wait - what is a type system?

In all languages, even in Assembly, we have at least two components:

Data

Operations

Not all of the available operations make sense to all kinds of data.

If you use ***incompatible pieces*** of data for an operation, you will have a ***representation error***

Programming languages use a ***type system*** to look at a program and determine if a representation error will happen or not

What are the possible strategies that a type system can use to handle representation errors?

Strategies

- Generate a compile error
- Perform a type check before run the code
- Well defined error set

- Unpredictable runtime errors
- Try implicit conversion

- A compiler tags pieces of code and tries to infer if the behaviour will be valid or not (before the program runs)

- A compiler / interpreter generates code to keep track of the data

Strategies

- Generate a compile error
- Perform a type check before run the code
- Well defined error set

"Strong"

- Unpredictable runtime errors
- Try implicit conversion

"Weak"

- A compiler tags pieces of code and tries to infer if the behaviour will be valid or not (before the program runs)

"Static"

- A compiler / interpreter generates code to keep track of the data

"Dynamic"

* Definitions are not exact on literature

You don't have to choose only one alternative

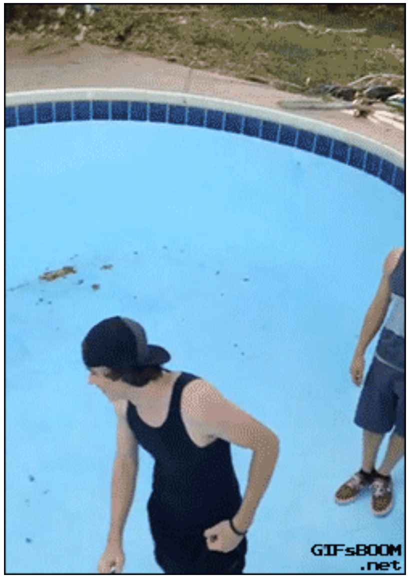
Java: static (why?)

Python: dynamic

But how can we perform the 'type check' mentioned before?

Have you ever heard someone saying "Language X has a terrible type system, it is a total mess!" Why? What does it even mean? How can we prove that?

We need some Mathematics

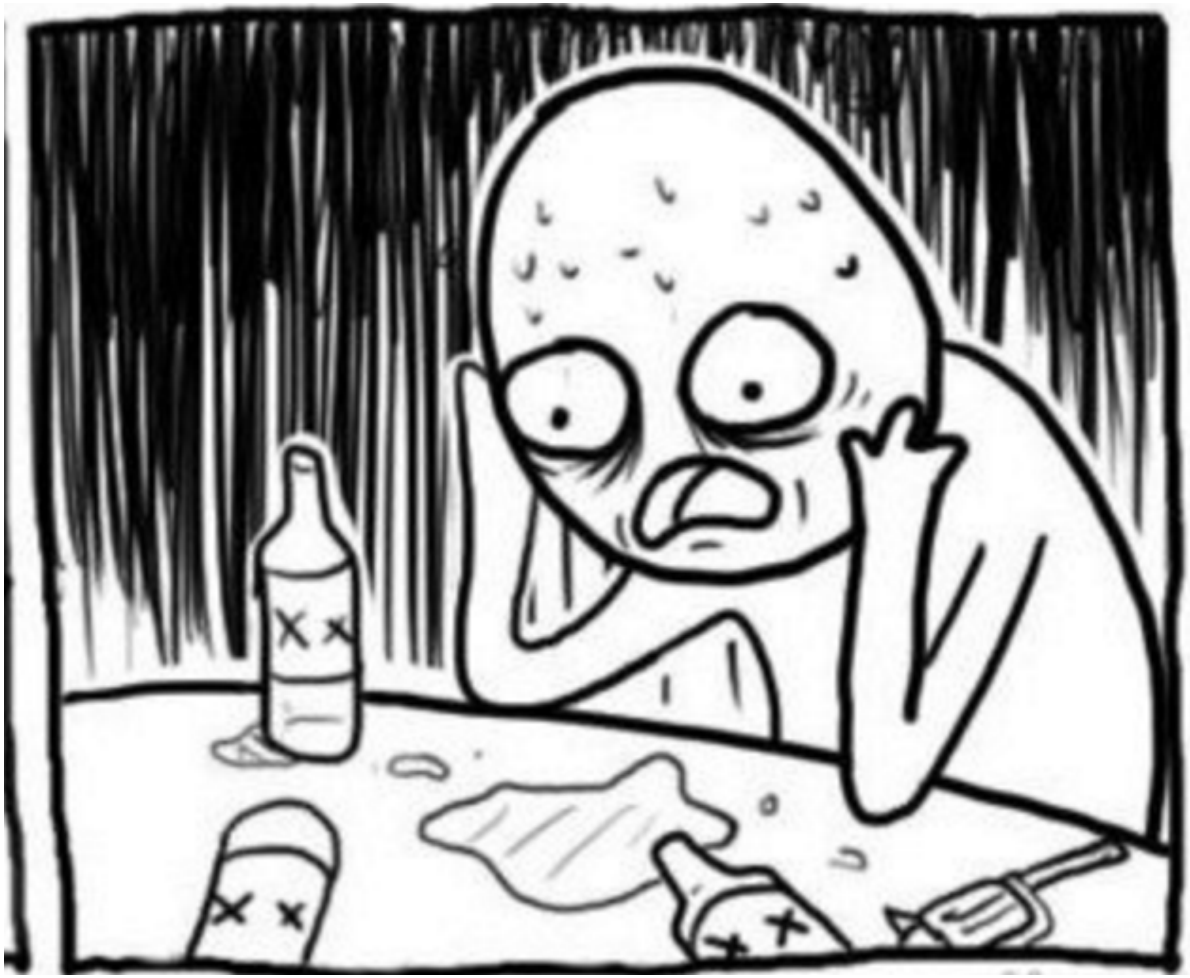


The steps to Type Theory

$$\frac{\mathsf{validType}(\Gamma, \varsigma_1) \quad \Gamma.classMap(\varsigma_1).super = \varsigma_2}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}$$

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2 \quad \Gamma \vdash \varsigma_2 \sqsubset_{class} \varsigma_3}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_3}$$

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}{\Gamma \vdash \varsigma_1 \sqsubseteq_{class} \varsigma_2} \qquad \frac{\mathsf{validType}(\Gamma, \varsigma)}{\Gamma \vdash \varsigma \sqsubseteq_{class} \varsigma}$$



Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- **Sketching the possible types**
- Symbolic Logic analysis
- Predicate logic
- Getting there!
- Why is this important?
- Challenges

#1:

Given a language, collect all the keywords and analyse the grammar for each of these works individually.

#1 - Example in Java:

extends	==> extends ClassType
implements	==> implements InterfaceTypeList
throws	==> throws ClassTypeList

#2

Make it look like Mathematics - replace text with variables :)

#2 - Example in Java:

extends ==> extends ClassType

implements ==> implements InterfaceTypeList

throws ==> throws ClassTypeList

A ==> ClassType

B ==> InterfaceType(List)

C ==> ClassType(List)

#2 - Example in Java:

(people like letters from the greek alphabet)

ζ ==> ClassType

#3

Group these results in sets and remove duplicates. These sets will reveal the types.

A very difficult task in science is grouping topics appropriately.

Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- **Sketching the possible types**
- **Symbolic Logic analysis**
- Predicate logic
- Getting there!
- Why is this important?
- Challenges

#4

Use symbolic logic to simplify your system

#4 - Example in Java

τ_r	*ResultType*	::=	*Type* | `void`
τ	*Type*	::=	*PrimitiveType* | *ReferenceType*
ρ	*ReferenceType*	::=	*ClassOrInterfaceType* | *ArrayType*
π	*PrimitiveType*	::=	**boolean** | **byte** | **short** | **int** | **long** | **char** | **float** | **double**
μ	*ClassOrInterfaceType*	::=	*ClassType* | *InterfaceType*
α	*ArrayType*	::=	*SimpleType* []
$\sigma*$	*SimpleType*	::=	*PrimitiveType* | *ClassOrInterfaceType*
ς	*ClassType*	::=	*Identifier*
ι	*InterfaceType*	::=	*Identifier*

τ_a	$\in$	*ArgumentType*	::=	*ParameterType* | Null
τ_e	$\in$	*ExpressionType*	::=	*ResultType* | Null
τ_p	$\in$	*ParameterType*	::=	*Type* | Unit
σ	$\in$	*NullOrSimpleType*	::=	*SimpleType* | Null

(::= is the definition symbol)

http://www.jot.fm/issues/issue_2007_09/article3.pdf

(Of course, you can come up with a different grouping)

#4

Every program (in Java) has its set of Classes and Variables. We call it Environment (Γ):

$$\begin{aligned} \mathit{Environment} \quad = \quad \langle \quad & \mathit{classMap}\colon \quad \mathit{ClassMap}, \\ & \mathit{interfaceMap}\colon \mathit{InterfaceMap} \quad \rangle \end{aligned}$$

#4

All mappings of a ClassType have a ClassDeclaration in Java (the same for interfaces). We will use the symbol $\xrightarrow[m]{}$

$$ClassMap = ClassType \xrightarrow[m]{} ClassDecl$$

$$InterfaceMap = InterfaceType \xrightarrow[m]{} InterfaceDecl$$

#4

Keep expanding the definitions:

$$
\begin{array}{lcl}
\mathit{Environment} & = & \langle\ \begin{array}{ll} \mathit{classMap}\colon & \mathit{ClassMap}, \\ \mathit{interfaceMap}\colon & \mathit{InterfaceMap} \end{array}\ \rangle \\[1em]
\mathit{ClassMap} & = & \mathit{ClassType} \xrightarrow[m]{} \mathit{ClassDecl} \\[1em]
\mathit{InterfaceMap} & = & \mathit{InterfaceType} \xrightarrow[m]{} \mathit{InterfaceDecl} \\[1em]
\mathit{ClassDecl} & = & \langle\ \begin{array}{ll} \mathit{modifiers}\colon & (\mathit{ModifierName})\text{-set}, \\ \mathit{super}\colon & (\mathit{ClassType})\text{-set}, \\ \mathit{interfaces}\colon & (\mathit{InterfaceType})\text{-set}, \\ \mathit{fields}\colon & \mathit{Identifier} \xrightarrow[m]{} \mathit{FieldInfo}, \\ \mathit{methods}\colon & \mathit{Sig} \xrightarrow[m]{} \mathit{MethodInfo}, \\ \mathit{constructors}\colon & \mathit{Sig} \xrightarrow[m]{} \mathit{ConstructorInfo} \end{array}\ \rangle \\[1em]
\mathit{InterfaceDecl} & = & \langle\ \begin{array}{ll} \mathit{modifiers}\colon & (\mathit{ModifierName})\text{-set}, \\ \mathit{interfaces}\colon & (\mathit{InterfaceType})\text{-set}, \\ \mathit{fields}\colon & \mathit{Identifier} \xrightarrow[m]{} \mathit{FieldInfo}, \\ \mathit{methods}\colon & \mathit{Sig} \xrightarrow[m]{} \mathit{MethodInfo} \end{array}\ \rangle \\[1em]
\mathit{Sig} & = & \mathit{Identifier} \times \mathit{ParameterType}
\end{array}
$$

http://www.jot.fm/issues/issue_2007_09/article3.pdf

Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- **Sketching the possible types**
- **Symbolic Logic analysis**
- **Predicate logic**
- Getting there!
- Why is this important?
- Challenges

#5

Use predicate logic to analyse your system. Start with true statements ('well-formed'):

#5 - Example in Java

validType = true

validType (primitive) = true

validType (π) = true

validType (environment, primitive) = true

validType (Γ, π) = true

validClass(Type) = ClassMap of the environment for that type

validClass(τ) = Γ classMap(τ)

validType(Class) = validClass(Type)

validType(ζ) = validClass(ζ)

#6 - Breathe

Free GIF!

#6 - Lambda Calculus

Understanding lambda calculus (out of scope of this presentation) will help you come out with these relations.

#6 - Bonus - Lambda Calculus

Lambda Calculus is about formal function theory. We can apply them to functional programming. We can also apply the ideas to general functions in programming.

`http://www.cs.le.ac.uk/people/amurawski/mgs2011-tlc.pdf`

#6 Lambda Calculus

With Lambda Calculus we can define a ***Type*** itself

"A type is a collection of objects having similar structure"

`http://www.nuprl.org/book/Introduction_Type_Theory.html`

#6 Lambda Calculus

Functions can transform data

```
public Integer nextInt(Integer number)
```

A: Integer

$A \rightarrow A$

#6 Lambda Calculus

Functions can transform data

```
public Integer nextInt(Integer number)
```

function: **λx.x+1**

That looks like mathematics!

#6 Lambda Calculus

λx.x+1 is in (A → A**)**

Lamba Calculus help us to build these statements, highly connected to predicate logic

#7 - Write some statements that you can prove:

"In Java, every class type that you define will be a subclass of a class"

#7 - Sketch a mathematical expression:

class => ζ

environment => Γ

class relation

(subclass or class itself) => $\sqsubset_{class}$ $\sqsubseteq_{class}$

#7 - Sketch a mathematical expression:

In an environment Γ, we can prove that a class of a certain type is a subclass of another type or the other type itself (Object)

$$\Gamma \vdash \zeta 1 \sqsubseteq_{class:} \zeta 2$$

Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- **Sketching the possible types**
- **Symbolic Logic analysis**
- **Predicate logic**
- **Getting there!**
- Why is this important?
- Challenges

#8 - We can almost read this:

$$\frac{\mathsf{validType}(\Gamma, \varsigma_1) \quad \Gamma.classMap(\varsigma_1).super = \varsigma_2}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}$$

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2 \quad \Gamma \vdash \varsigma_2 \sqsubset_{class} \varsigma_3}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_3}$$

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}{\Gamma \vdash \varsigma_1 \sqsubseteq_{class} \varsigma_2} \qquad \frac{\mathsf{validType}(\Gamma, \varsigma)}{\Gamma \vdash \varsigma \sqsubseteq_{class} \varsigma}$$

#8:

A valid class. A class type that is a subclass of another type

The super class of a type. But the super class also is a subclass of another type

$$\frac{\boxed{\mathsf{validType}(\Gamma, \varsigma_1)} \quad \boxed{\Gamma.classMap(\varsigma_1).super = \varsigma_2}}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}$$

$$\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2 \qquad \Gamma \vdash \varsigma_2 \sqsubset_{class} \varsigma_3$$

#8:

A valid class. A class type that is a subclass of another type

The super class of a type. But the super class also is a subclass of another type

$$\frac{\boxed{\mathsf{validType}(\Gamma, \varsigma_1)} \quad \boxed{\Gamma.classMap(\varsigma_1).super = \varsigma_2}}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}$$

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \boxed{\varsigma_2} \quad \Gamma \vdash \boxed{\varsigma_2 \sqsubset_{class} \varsigma_3}}{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_3}$$

#8:

General subclass chain

$$\frac{\boxed{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2 \qquad \Gamma \vdash \varsigma_2 \sqsubset_{class} \varsigma_3}}{\boxed{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_3}}$$

A subclass or the class itself

$$\frac{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}{\Gamma \vdash \zeta 1 \sqsubseteq_{class:} \zeta 2}$$

#8:

Valid type!

$$\boxed{\Gamma \vdash \varsigma_1 \sqsubset_{class} \varsigma_2}$$

$$\Gamma \vdash \varsigma_1 \sqsubseteq_{class} \varsigma_2$$

$$\frac{\text{validType}(\Gamma, \zeta)}{\Gamma \vdash \zeta 1 \sqsubseteq_{class} \zeta 2}$$

True!

Agenda

- **Choosing a programming language**
- **Quick intro about type systems**
- **Sketching the possible types**
- **Symbolic Logic analysis**
- **Predicate logic**
- **Getting there!**
- **Why is this important?**
- Challenges

Why is this so important?

- Reduce runtime errors by checking the types
- IDEs can perform a better analysis of your code based on logical statements
- Different languages have different type systems
- You have a solid point to choose a language

Sometimes it is difficult to find an equivalent type across different languages

Collect the characteristics that are important for you and compare them across the languages using the ideas of type theory

Examples:

- Is everything immutable here? (prove it)
- Is everything an object in language X? (prove it!)
- Do I have co-variance? (related to subtyping)

Agenda

- Choosing a programming language
- Quick intro about type systems
- Sketching the possible types
- Symbolic Logic analysis
- Predicate logic
- Getting there!
- Why is this important?
- **Challenges**

Challenges

- There is no single way to describe a type system
- It is hard to find equivalences between languages
- It is a lot of mathematics!
- We have lots of theory and very few time to study them

Challenges

Java type system - proposed type representations:

- http://www.jot.fm/issues/issue_2007_09/article3.pdf
- http://www.dsi.unive.it/myths/GC2004/Slides/Zucca_slides.pdf
- http://groups.csail.mit.edu/pag/pubs/ref-immutability-oopsla2004-abstract.html
- http://pubs.doc.ic.ac.uk/JavaProbablySound/JavaProbablySound.pdf

Final notes

Don't be scared of mathematics - the concepts, itself, are not so difficult!

There are several active researched focusing on Type Theory!

Even if you don't have a PhD, you can learn and use type theory concepts!

Type Theory

Most type theory studies are applied to functional languages.

But you can analyse languages that are not purely funcional as well.

References

- http://blogs.atlassian.com/2013/01/covariance-and-contravariance-in-scala/
- http://cseweb.ucsd.edu/~atl017/papers/pldi11.pdf
- STEPANOV, A. *Elements of Programming.*
- PIERCE, B. *Types and Programming Languages*
- THOMPSON, S. *Type Theory and Functional Programming* (free ebook!)
- MICHAELSON, G. *An Introduction to Functional Programming Through Lambda Calculus.*

Session at Open Source Bridge 2016

http://slides.com/hannelitavante-hannelita/type-theory-101-35#/

Session at Devoxx Belgium 2016

http://slides.com/hannelitavante-hannelita/devoxx-be-notes-type-theory#/

Special Thanks

- B.C., for the constant review and support
- Professor M. Coutinho (UNIFEI)
- JFokus Team

Thank you :)

Questions?

hannelita@gmail.com

@hannelita