

**JET  
BRAINS**



# Postmodern type systems

Tesla Ice Zhang

# About Me

---

I am Tesla Ice Zhang. I work with programming languages.

Here's my profile: [Tesla \(Yinsen\) Ice Zhang \(psu.edu\)](#)

# Dependent types

---

Let's first dive into DT.

## Popular type systems

---

- Assembly has no types.
- C, Java 4, C# 1, etc. have simple types.
- Java 5, C# 2, Kotlin, etc. have fancier types.
- C++ templates are even fancier.
- Swift, Haskell, etc. have some deductions.

## Mixing types and values

---

- Lambda cube: an abstraction over the mixture of types and values.
- CIC and MLTT – where types and values are mixed altogether.

In some (old) research PLs, we can mix values into types (similar to constexpr in C++, but in this case, the entire language is constexpr).

This allows us to type more values.

## Functions

---

- The `printf` function. Can we check its arguments' types at compile time?

We first curry `printf : (string, any[]) -> ()`.

`printf : string -> (any[] -> ())`

That is to say,

```
printf "xyr" : any[] -> ()  
printf "age %i" : any[] -> ()  
printf "job %s" : any[] -> ()  
printf "at (%f, %f)" : any[] -> ()
```



This is what we have:

```
printf "xyr" : any[] -> ()  
printf "age %i" : any[] -> ()  
printf "job %s" : any[] -> ()  
printf "at (%f, %f)" : any[] -> ()
```

This is what we want:

```
printf "xyr" : () -> ()  
printf "age %i" : (int) -> ()  
printf "job %s" : (string) -> ()  
printf "at (%f, %f)" : (float, float) -> ()
```

To do this, we need to change printf's type into something else. What should we replace the `any[]` with?

```
printf : string -> (? -> ())
```

Observe: it depends on the first argument. So, let's invent this new syntax, which gives a name to the first argument, so we can talk about its **value** elsewhere in the type signature:

```
printf : (s : string) -> (? -> ())
```

Essentially, the `?` should be a type calculated from `s`, so we replace it with a function. The

```
printf : (s : string) -> (? -> ())
```

Becomes:

```
printf : (s : string) -> (Fmt(s) -> ())
```

Observe `Fmt` – it should be a function, but what type does it have?

```
printf : (s : string) -> (Fmt(s) -> ())
```

It returns a type! What is the type of types?

## Dependent Types

---

- What we've just seen, is a dependent type system.
- It has functions returning types (in other words, type expressions with values inside), the type of types, etc.

# Modeling stuffs

---

How can we exploit the power of DT?

**What are types,  
precisely?**

---

- We can see types as sets, and their instances as the elements of sets



Types	Sets
Nothing, !	$\emptyset$
Types that talks about values	Families of sets
Functions	Maps of values
Classes, Records, Tuples	Products of sets
Subtypes	Subsets
Equality of values	Equality of elements
...	...

**Sets = types, *relations* = ?**

---

- Fun fact: we also encode relations and logical propositions as types

Types	Propositions
Nothing, !	$\perp$
Types that talks about values	$\forall x, f(x). y$
Functions	$\forall x. y$
Classes, Records, Tuples	$\wedge$
Subtypes	$\vee$
The MLTT Id type	Equality of terms
...	...

Example: 2 is not a rational (proof omitted).

theorem : (m : Nat) -> (n : Nat)  
-> 2 \* m \* m = n \* n -> m = 0

Corresponding proposition:

$$\forall m, n \in \mathbb{N} \rightarrow 2 \times m^2 = n^2 \rightarrow m = 0$$

## What else can we do?

---

What if the sets are no longer discrete, but instead continuous?

Can we talk about continuous (preserving topology) functions?

**What else can we encode  
with types?**

---

- We can also encode topological spaces as types, and we interpret paths the same way as an ‘equality relation’
- This allows us to talk about spaces and continuous functions in type systems

Types	Spaces
Elements	Points
Equality of values	Paths on points
Functions	Continuous maps of values
...	...

## Fundamental groups

---

The definition of fundamental group can be encoded as a type (that talks about another type):

```
Loop : (A : Type) -> (a : A) -> Type
Loop A a => a = a
```

Then we can prove it to be a group (easy!).



**The theorem:**  $\pi_1(\mathbb{S}^1) \cong \mathbb{Z}$

---

We can prove a very basic fact, that the fundamental group of circle is isomorphic to the integer additive group, using **a type system!**

# Klein-Bottles

---

Klein bottles are just torus with the surface twisted:

```
data Torus : Type where
  point : Torus
  line1 : point ≡ point
  line2 : point ≡ point
  square : PathP (λ i → line1 i ≡ line1 i) line2 line2
```

```
data KleinBottle : Type where
  point : KleinBottle
  line1 : point ≡ point
  line2 : point ≡ point
  square : PathP (λ i → line1 (~ i) ≡ line1 i) line2 line2
```

# Hopf fibrations

---

Hopf fibrations of spheres:

```
rotIsEquiv : (a : S1) → isEquiv (a · _)
```

```
HopfS2 : S2 → Type0
```

```
HopfS2 base = S1
```

```
HopfS2 (surf i j) = Glue S1 (λ { (i = i0) → _ , idEquiv S1  
; (i = i1) → _ , idEquiv S1  
; (j = i0) → _ , idEquiv S1  
; (j = i1) → _ , _ , rotIsEquiv (loop i) } )
```

# Why types?

---

So – what's the point of all of these?

And how complicated it is?

\_\_\_\_\_

- we carry out a similar calculation, which turns out to be trivial. Thus we obtain the following theorem.
- Theorem 1.1.** Let  $p = 2$ . Then there is a module isomorphism
- $$P(\mathcal{A}^*) \otimes P(\mathcal{A}) \cong P(\mathcal{A}^*) \otimes P(\mathcal{A}_2 \oplus \mathcal{A}_3),$$
- and the reduced coproduct can be given as follows:
- $$\begin{aligned} \hat{\psi}(x_2) = & x_2 \otimes 1 + x_2 \otimes x_1 + x_2 \otimes x_2 + x_2 \otimes x_3 + x_2 \otimes x_4 + x_2 \otimes x_5 \\ & + x_2 \otimes x_6 + x_2 \otimes x_7 + x_2 \otimes x_8 + x_2 \otimes x_9 + x_2 \otimes x_{10} + x_2 \otimes x_{11} \\ & + x_2 \otimes x_{12} + x_2 \otimes x_{13} + x_2 \otimes x_{14} + x_2 \otimes x_{15} + x_2 \otimes x_{16} + x_2 \otimes x_{17} \\ & + x_2 \otimes x_{18} + x_2 \otimes x_{19} + x_2 \otimes x_{20} + x_2 \otimes x_{21} + x_2 \otimes x_{22} + x_2 \otimes x_{23} \\ & + x_2 \otimes x_{24} + x_2 \otimes x_{25} + x_2 \otimes x_{26} + x_2 \otimes x_{27} + x_2 \otimes x_{28} + x_2 \otimes x_{29} \\ & + x_2 \otimes x_{30} + x_2 \otimes x_{31} + x_2 \otimes x_{32} + x_2 \otimes x_{33} + x_2 \otimes x_{34} + x_2 \otimes x_{35} \\ & + x_2 \otimes x_{36} + x_2 \otimes x_{37} + x_2 \otimes x_{38} + x_2 \otimes x_{39} + x_2 \otimes x_{40} + x_2 \otimes x_{41} \\ & + x_2 \otimes x_{42} + x_2 \otimes x_{43} + x_2 \otimes x_{44} + x_2 \otimes x_{45} + x_2 \otimes x_{46} + x_2 \otimes x_{47} \\ & + x_2 \otimes x_{48} + x_2 \otimes x_{49} + x_2 \otimes x_{50} + x_2 \otimes x_{51} + x_2 \otimes x_{52} + x_2 \otimes x_{53} \\ & + x_2 \otimes x_{54} + x_2 \otimes x_{55} + x_2 \otimes x_{56} + x_2 \otimes x_{57} + x_2 \otimes x_{58} + x_2 \otimes x_{59} \\ & + x_2 \otimes x_{60} + x_2 \otimes x_{61} + x_2 \otimes x_{62} + x_2 \otimes x_{63} + x_2 \otimes x_{64} + x_2 \otimes x_{65} \\ & + x_2 \otimes x_{66} + x_2 \otimes x_{67} + x_2 \otimes x_{68} + x_2 \otimes x_{69} + x_2 \otimes x_{70} + x_2 \otimes x_{71} \\ & + x_2 \otimes x_{72} + x_2 \otimes x_{73} + x_2 \otimes x_{74} + x_2 \otimes x_{75} + x_2 \otimes x_{76} + x_2 \otimes x_{77} \\ & + x_2 \otimes x_{78} + x_2 \otimes x_{79} + x_2 \otimes x_{80} + x_2 \otimes x_{81} + x_2 \otimes x_{82} + x_2 \otimes x_{83} \\ & + x_2 \otimes x_{84} + x_2 \otimes x_{85} + x_2 \otimes x_{86} + x_2 \otimes x_{87} + x_2 \otimes x_{88} + x_2 \otimes x_{89} \\ & + x_2 \otimes x_{90} + x_2 \otimes x_{91} + x_2 \otimes x_{92} + x_2 \otimes x_{93} + x_2 \otimes x_{94} + x_2 \otimes x_{95} \\ & + x_2 \otimes x_{96} + x_2 \otimes x_{97} + x_2 \otimes x_{98} + x_2 \otimes x_{99} + x_2 \otimes x_{100} + x_2 \otimes x_{101} \\ & + x_2 \otimes x_{102} + x_2 \otimes x_{103} + x_2 \otimes x_{104} + x_2 \otimes x_{105} + x_2 \otimes x_{106} + x_2 \otimes x_{107} \\ & + x_2 \otimes x_{108} + x_2 \otimes x_{109} + x_2 \otimes x_{110} + x_2 \otimes x_{111} + x_2 \otimes x_{112} + x_2 \otimes x_{113} \\ & + x_2 \otimes x_{114} + x_2 \otimes x_{115} + x_2 \otimes x_{116} + x_2 \otimes x_{117} + x_2 \otimes x_{118} + x_2 \otimes x_{119} \\ & + x_2 \otimes x_{120} + x_2 \otimes x_{121} + x_2 \otimes x_{122} + x_2 \otimes x_{123} + x_2 \otimes x_{124} + x_2 \otimes x_{125} \\ & + x_2 \otimes x_{126} + x_2 \otimes x_{127} + x_2 \otimes x_{128} + x_2 \otimes x_{129} + x_2 \otimes x_{130} + x_2 \otimes x_{131} \\ & + x_2 \otimes x_{132} + x_2 \otimes x_{133} + x_2 \otimes x_{134} + x_2 \otimes x_{135} + x_2 \otimes x_{136} + x_2 \otimes x_{137} \\ & + x_2 \otimes x_{138} + x_2 \otimes x_{139} + x_2 \otimes x_{140} + x_2 \otimes x_{141} + x_2 \otimes x_{142} + x_2 \otimes x_{143} \\ & + x_2 \otimes x_{144} + x_2 \otimes x_{145} + x_2 \otimes x_{146} + x_2 \otimes x_{147} + x_2 \otimes x_{148} + x_2 \otimes x_{149} \\ & + x_2 \otimes x_{150} + x_2 \otimes x_{151} + x_2 \otimes x_{152} + x_2 \otimes x_{153} + x_2 \otimes x_{154} + x_2 \otimes x_{155} \\ & + x_2 \otimes x_{156} + x_2 \otimes x_{157} + x_2 \otimes x_{158} + x_2 \otimes x_{159} + x_2 \otimes x_{160} + x_2 \otimes x_{161} \\ & + x_2 \otimes x_{162} + x_2 \otimes x_{163} + x_2 \otimes x_{164} + x_2 \otimes x_{165} + x_2 \otimes x_{166} + x_2 \otimes x_{167} \\ & + x_2 \otimes x_{168} + x_2 \otimes x_{169} + x_2 \otimes x_{170} + x_2 \otimes x_{171} + x_2 \otimes x_{172} + x_2 \otimes x_{173} \\ & + x_2 \otimes x_{174} + x_2 \otimes x_{175} + x_2 \otimes x_{176} + x_2 \otimes x_{177} + x_2 \otimes x_{178} + x_2 \otimes x_{179} \\ & + x_2 \otimes x_{180} + x_2 \otimes x_{181} + x_2 \otimes x_{182} + x_2 \otimes x_{183} + x_2 \otimes x_{184} + x_2 \otimes x_{185} \\ & + x_2 \otimes x_{186} + x_2 \otimes x_{187} + x_2 \otimes x_{188} + x_2 \otimes x_{189} + x_2 \otimes x_{190} + x_2 \otimes x_{191} \\ & + x_2 \otimes x_{192} + x_2 \otimes x_{193} + x_2 \otimes x_{194} + x_2 \otimes x_{195} + x_2 \otimes x_{196} + x_2 \otimes x_{197} \\ & + x_2 \otimes x_{198} + x_2 \otimes x_{199} + x_2 \otimes x_{200} + x_2 \otimes x_{201} + x_2 \otimes x_{202} + x_2 \otimes x_{203} \\ & + x_2 \otimes x_{204} + x_2 \otimes x_{205} + x_2 \otimes x_{206} + x_2 \otimes x_{207} + x_2 \otimes x_{208} + x_2 \otimes x_{209} \\ & + x_2 \otimes x_{210} + x_2 \otimes x_{211} + x_2 \otimes x_{212} + x_2 \otimes x_{213} + x_2 \otimes x_{214} + x_2 \otimes x_{215} \\ & + x_2 \otimes x_{216} + x_2 \otimes x_{217} + x_2 \otimes x_{218} + x_2 \otimes x_{219} + x_2 \otimes x_{220} + x_2 \otimes x_{221} \\ & + x_2 \otimes x_{222} + x_2 \otimes x_{223} + x_2 \otimes x_{224} + x_2 \otimes x_{225} + x_2 \otimes x_{226} + x_2 \otimes x_{227} \\ & + x_2 \otimes x_{228} + x_2 \otimes x_{229} + x_2 \otimes x_{230} + x_2 \otimes x_{231} + x_2 \otimes x_{232} + x_2 \otimes x_{233} \\ & + x_2 \otimes x_{234} + x_2 \otimes x_{235} + x_2 \otimes x_{236} + x_2 \otimes x_{237} + x_2 \otimes x_{238} + x_2 \otimes x_{239} \\ & + x_2 \otimes x_{240} + x_2 \otimes x_{241} + x_2 \otimes x_{242} + x_2 \otimes x_{243} + x_2 \otimes x_{244} + x_2 \otimes x_{245} \\ & + x_2 \otimes x_{246} + x_2 \otimes x_{247} + x_2 \otimes x_{248} + x_2 \otimes x_{249} + x_2 \otimes x_{250} + x_2 \otimes x_{251} \\ & + x_2 \otimes x_{252} + x_2 \otimes x_{253} + x_2 \otimes x_{254} + x_2 \otimes x_{255} + x_2 \otimes x_{256} + x_2 \otimes x_{257} \\ & + x_2 \otimes x_{258} + x_2 \otimes x_{259} + x_2 \otimes x_{260} + x_2 \otimes x_{261} + x_2 \otimes x_{262} + x_2 \otimes x_{263} \\ & + x_2 \otimes x_{264} + x_2 \otimes x_{265} + x_2 \otimes x_{266} + x_2 \otimes x_{267} + x_2 \otimes x_{268} + x_2 \otimes x_{269} \\ & + x_2 \otimes x_{270} + x_2 \otimes x_{271} + x_2 \otimes x_{272} + x_2 \otimes x_{273} + x_2 \otimes x_{274} + x_2 \otimes x_{275} \\ & + x_2 \otimes x_{276} + x_2 \otimes x_{277} + x_2 \otimes x_{278} + x_2 \otimes x_{279} + x_2 \otimes x_{280} + x_2 \otimes x_{281} \\ & + x_2 \otimes x_{282} + x_2 \otimes x_{283} + x_2 \otimes x_{284} + x_2 \otimes x_{285} + x_2 \otimes x_{286} + x_2 \otimes x_{287} \\ & + x_2 \otimes x_{288} + x_2 \otimes x_{289} + x_2 \otimes x_{290} + x_2 \otimes x_{291} + x_2 \otimes x_{292} + x_2 \otimes x_{293} \\ & + x_2 \otimes x_{294} + x_2 \otimes x_{295} + x_2 \otimes x_{296$$

## Why are we encoding things into types?

---

- How do we teach students ‘theorem proving’? What’s a valid proof and what’s not?
- The theory of types answers this *very clearly*: a proof is an instance of the type corresponds to the theorem.

## Why are we encoding things into types?

---

- How do we study a mathematical concept? By asking the person who invented it? By asking a person who understand it?
- How do we even determine the prerequisite of a concept? Several math books have their chapter 0/1 talking about basic set theory. Is that necessary?
- If we write the idea using a programming language, then we can just 'go to definition' in the IDE!

## Why are we encoding spaces into types?

---

- It brings better extensionality to the type system like bisimulation and function's extensionality
- It provides a canonical way to represent quotients without breaking subject reduction (as in Lean) or introducing setoid hell (as in Coq)
- It allows us to transport proofs from isomorphic types (by univalence)
- It provides a low-level language to reason over continuous spaces/functions
- It opens the door towards  $\infty$ -categories in types



## What can it do to a normal programmer?

---

- We can help real-world programming with more expressive type systems (if we open an unsafe mode)
- We can make sure some contracts are satisfied at compile time, and erase the checks or assertions at compile time
- Rust's lifetime, generalized

# My secret, evil plan

---

Not going to be a part of the slides 😊

**Thank you  
for your attention**

---