Type Theory and Agda

BY SEAN O'CONNOR

What is Type Theory

Everything we can talk about is a term of some type

Agda can be used to formalize Homotopy Type Theory

For example, `zero-ℕ : ℕ` denotes that the term `zero-ℕ` is of type ℕ, the type of natural numbers.

Additionally, `zero-ℤ : ℤ` denotes that the term `zero-ℤ` is of type ℤ, the type of integers.

However, `zero-ℕ` and `zero-ℤ` are treated as different terms, as each term in Homotopy Type Theory can only have one type.

"Propositions as Types"

Logical propositions are types

If P is a proposition, $p : P$ says that p is a proof of P

All propositions are types, but not all types are propositions

For example, $n == m$ is a proposition whenever $n : \mathbb{N}$ and $m : \mathbb{N}$

How functions function

In type theory, if f is a function that takes as input a term of type A and produces as output a term of type B, then $f : A \to B$

$g : P \to Q$ says that g is a function that converts proofs of P into proofs of Q

Specifically, if $p : P$, then $g\ p : Q$

The notation $\lambda\ x \to ?$ can be used to easily inline functions. As an example, $\lambda\ n \to n + \text{one-}\mathbb{N}$ is a function of type $\mathbb{N} \to \mathbb{N}$ that adds 1

Negation in Type Theory

Not helpful to say a term isn't of some type

We define $\varnothing : \mathsf{Type}$ such that there are explicitly no terms of type $\varnothing$

Furthermore, we define $\neg A = A \to \varnothing$ for any type A

If we had both $a : A$ and $y : \neg A$, then $y\ a : \varnothing$

But there are no terms of type $\varnothing$, so we have a contradiction

Constructive Logic

The LEM is not assumed to be true. That is, it is not assumed that there exists a term f such that, for any type A, we have that $f\ A : A \vee (\neg A)$

Likewise, double negation elimination is also not assumed to be true, as it posits that for any type A, there exists a function of type $(\neg (\neg A)) \rightarrow A$

However, we can derive triple negation elimination as a theorem, which states that for any type A, there exists a function of type $(\neg (\neg (\neg A))) \rightarrow (\neg A)$

```
data ∅ : Type where

¬ : (A : Type) → Type
¬ A = A → ∅

triple-¬ : {A : Type}  → (¬ (¬ (¬ A))) → (¬ A)
triple-¬ x = ?
```

```
data ∅ : Type where

¬ : (A : Type) → Type
¬ A = A → ∅

triple-¬ : {A : Type}  → (¬ (¬ (¬ A))) → (¬ A)
triple-¬ x = ?
```

```
triple-¬ x = { }0
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ A
————————————————————————————————————————
x : ¬ (¬ (¬ A))
```

```
data ∅ : Type where

¬ : (A : Type) → Type
¬ A = A → ∅

triple-¬ : {A : Type}  → (¬ (¬ (¬ A))) → (¬ A)
triple-¬ x = ?
```

```
triple-¬ x = {λ a → ? }0
```

```
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ A
————————————————————————————————————————
x : ¬ (¬ (¬ A))
```

```
data ∅ : Type where

¬ : (A : Type) → Type
¬ A = A → ∅

triple-¬ : {A : Type}  → (¬ (¬ (¬ A))) → (¬ A)
triple-¬ x = ?
```

```
triple-¬ x = λ a → { }1
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → { }1
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → {x ? }1
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → { }1
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x { }2
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ (¬ A)
————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x { }2
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ (¬ A)
————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x {λ y → ? }2
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ (¬ A)
————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x { }2
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ¬ (¬ A)
————————————————————————————————————————————
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x λ y → { }3
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————————————————
y : ¬ A
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x λ y → { }3
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————
y : ¬ A
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x λ y → {y a }3
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————
y : ¬ A
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x λ y → {y a }3
∏U\**-  Presentation.agda   Top L1     (Agda)
Goal: ∅
————————————————————————————————
y : ¬ A
a : A
x : ¬ (¬ (¬ A))
```

```
triple-¬ x = λ a → x λ y → y a
```

Acknowledgements

David Jaz Myers, my advisor

The entire JHU Directed Reading Program

Thank you for listening!