Martin Löf Type Theory, Homotopy Type Theory and Agda

Natalie Ravenhill

Type Theory

Timeline

- 1936/1940 - (Un)typed Lambda Calculus
  M ::= c | x | λ(x:τ).M | M M
  τ ::= t | $\tau \rightarrow \tau$
- 1972 - Martin Löf Type Theory
- late 00s - Homotopy Type Theory

Type Theory

Timeline

- 1936/1940 - (Un)typed Lambda Calculus
  M ::= c | x | λ(x:τ).M | M M
  τ ::= t | $\tau \to \tau$
- 1972 - Martin Löf Type Theory
- late 00s - Homotopy Type Theory

Agda

Agda is:

- A dependently typed programming language
- A theorem prover

Example Agda Code

```
-first define natural numbers:
data Nat : Set where
  zero : Nat
  succ : Nat → Nat

-then we can use this type to
-inductively define vectors
data Vec (A : Set) : Nat → Set where
  [] : Vec A zero
  _::_ : {n : Nat} → A → Vec A n → Vec A (succ n)
```

Curry Howard Correspondence

Logic	Type Theory
True	$\mathbb{1}$
False	$\mathbb{0}$
$\neg A$	$A \to \mathbb{0}$
$A \Rightarrow B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$\exists(x : A).P(x)$	$\Sigma_{(x:A)} P(x)$
$\forall(x : A).P(x)$	$\Pi_{(x:A)}.P(x)$

Product Type = Conjunction

```
U = Set

-dependent pair type
data Σ {A : U} (B : A → U) : U where
_,_ : (a : A) → (b : B a) → Σ B

-product type
_×_ : U → U → U
A × B = Σ (λ (a : A) → B)
```

Coproduct Type = Disjunction

$$\mathsf{U} = \mathsf{Set}$$

$$\begin{aligned}
&\mathsf{data}\ _+_\ (A\ B : \mathsf{U}) : \mathsf{U}\ \mathsf{where} \\
&\mathsf{inl} : A \to A + B \\
&\mathsf{inr} : B \to A + B
\end{aligned}$$

Negation

Negation is a function: $\mathsf{U} = \mathsf{Set}$

```
-false is empty type
data 𝟘 : U where

¬ : U → U
¬ A = A → 𝟘
```

Example of a proof

To prove the statement $\neg A \wedge \neg B \to \neg(A \vee B)$

```
open import Product
open import Coproduct
open import Negation

deMorgan : {A B : Set} → (¬ A) × (¬ B) → ¬(A + B)
deMorgan {A} {B} (f , g) = h
where
  h : A + B → 𝟘
  h (inl a) = f a
  h (inr b) = g b
```

$\exists$ = Dependent Pair Type

```
U = Set

-dependent pair type
data Σ {A : U} (B : A → U) : U where
_,_ : (a : A) → (b : B a) → Σ B
```

$\forall$ = Dependent Function Type

$$\mathsf{U} = \mathsf{Set}$$

$$\Pi : \{A : \mathsf{U}\}\ (B : A \to \mathsf{U}) \to \mathsf{U}$$
$$\Pi\ \ \{A\}\ B = \ (a : A) \to B\ a$$

Identity Type

Given two elements a , b of a type A, if they are equal to each other then we form an element of the type of proofs of equality, $\equiv$.

```
U = Set

infixr 0 _≡_
data _≡_ {A : U} : A → A → U where
  refl : (x : A) → x ≡ x
```

Same but with paths

Given two points a , b of a space A, if there is a path between them then we form an element of the type of paths between a and b , ≡.

```
U = Set

infixr 0 _≡_
data _≡_ {A : U} : A → A → U where
  refl : (x : A) → x ≡ x
```

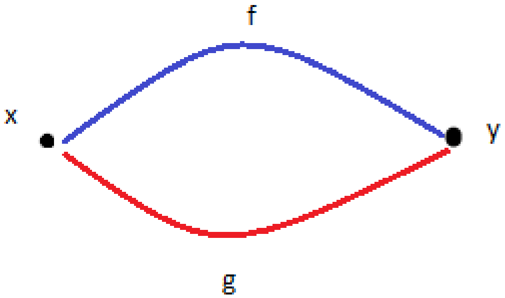
Type Theory

Timeline

- 1936/1940 - (Un)typed Lambda Calculus
  $M ::= c \mid x \mid \lambda(x{:}\tau).M \mid M\ M$
  $\tau ::= t \mid \tau \to \tau$
- 1972 - Martin Löf Type Theory
- late 00s - Homotopy Type Theory

Homotopy

A homotopy between a pair of continuous maps $f : X \to Y$ and $g : X \to Y$ is a continuous map $H : X \times [0,1] \to Y$ such that $H\ (x\ ,0) = f(x)$ and $H\ (x\ ,\ 1) = g(x)$



Homotopy Type Theory

Type Theory	HoTT
a : A	point
A	space
$a \equiv a$	path space
$a \equiv_{a \equiv a} a$	Homotopy

Properties of Equality

Equality	Homotopy
reflexivity	constant path
symmetry	inversion of paths
transitivity	concatenation

Induction

```
U = Set

data ℕ : U where
  zero : ℕ
  succ : ℕ → ℕ

ℕ-ind : {C : ℕ → U} → C zero → ((n : ℕ) → C n → C (succ n))
      → ((n : ℕ) → C n)
ℕ-ind {C} c₀ cs zero = c₀
ℕ-ind {C} c₀ cs (succ n) = cs n (ℕ-ind {C} c₀ cs n)
```

Induction on Equality

```
open import Equality

≡-Ind : (A : U) (C : (x y : A) → x ≡ y → U)
    → ((x : A) → C x x (refl x))
    → (x y : A) → (p : x ≡ y) → C x y p
≡-Ind A C c = f
where
    f : (x y : A) (p : x ≡ y) → C x y p
    f x .x (refl .x) = c x
```

Symmetry

```
open import Equality
open import Induction

symInd : (A : U) (x y : A) → x ≡ y → y ≡ x
symInd A = ≡-Ind A D d
  where
    D : (x y : A) → (x ≡ y) → U
    D x y _ = y ≡ x
    d : (x : A) → D x x (refl x)
    d = λ x → refl x
```

Transitivity

```
open import Equality
open import Induction

transitivity : (A : U) (x y z : A) → x ≡ y → y ≡ z → x ≡ z
transitivity A x y z p q = f x y p z q
where
  D : (x y : A) → x ≡ y → U
  D x y p = (z : A) → (q : y ≡ z) → x ≡ z
  c : (x : A) → D x x (refl x)
  c x z q = q
  E : (x z : A) (q : x ≡ z) → U
  E x z q = x ≡ z
  e : (x : A) → E x x (refl x)
  e x = refl x
  d : (x z : A) (q : x ≡ z) → E x z q
  d = ≡-Ind A E e
  f : (x y : A) (p : x ≡ y) → D x y p
  f = ≡-Ind A D d
```

Conclusion

We have discussed:

- Martin Löf Type Theory
- Curry Howard Correspondence
- Homotopy Type Theory
- Proofs on Equalities/Paths

But there is much more to Homotopy Type Theory!

For more information see:

`http://homotopytypetheory.org/`