An Introduction to Homotopy Type Theory

Amr Sabry

School of Informatics and Computing
Indiana University

October 31, 2013

Recursion and induction principles

Each type has a recursion and an induction principle.

```
- for natural numbers
```

$$
\begin{aligned}
&\mathrm{recN} : (C : \mathrm{Set}) \to C \to (\mathbb{N} \to C \to C) \to \mathbb{N} \to C \\
&\mathrm{recN}\ C\ c\ f\ 0 \qquad\qquad = c \\
&\mathrm{recN}\ C\ c\ f\ (\mathrm{suc}\ n) \quad = f\ n\ (\mathrm{recN}\ C\ c\ f\ n)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{indN} : \quad (C : \mathbb{N} \to \mathrm{Set}) \to \\
&\qquad\qquad C\ 0 \to ((n' : \mathbb{N}) \to C\ n' \to C\ (\mathrm{suc}\ n')) \to (n : \mathbb{N}) \to C\ n \\
&\mathrm{indN}\ C\ c\ f\ 0 \qquad\qquad = c \\
&\mathrm{indN}\ C\ c\ f\ (\mathrm{suc}\ n) \quad = f\ n\ (\mathrm{indN}\ C\ c\ f\ n)
\end{aligned}
$$

Recursion and induction principles

Examples:

$$\mathsf{double} : \mathbb{N} \to \mathbb{N}$$
$$\mathsf{double} = \mathsf{recN}\ \mathbb{N}\ 0\ (\lambda\ n\ r \to \mathsf{suc}\ (\mathsf{suc}\ r))$$

$$\mathsf{add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$
$$\mathsf{add} = \mathsf{recN}\ (\mathbb{N} \to \mathbb{N})\ (\lambda\ n \to n)\ (\lambda\ m\ g\ n \to \mathsf{suc}\ (g\ n))$$

$$\mathsf{assocAdd} : (i\ j\ k : \mathbb{N}) \to \mathsf{add}\ i\ (\mathsf{add}\ j\ k) \equiv \mathsf{add}\ (\mathsf{add}\ i\ j)\ k$$
$$\begin{array}{l} \mathsf{assocAdd} = \\ \quad \mathsf{indN} \\ \quad\quad (\lambda\ i \to (j : \mathbb{N}) \to (k : \mathbb{N}) \to \mathsf{add}\ i\ (\mathsf{add}\ j\ k) \equiv \mathsf{add}\ (\mathsf{add}\ i\ j)\ k) \\ \quad\quad (\lambda\ j\ k \to ?) \\ \quad\quad (\lambda\ i\ h\ j\ k \to ?) \end{array}$$

Propositions as types

- A proposition is a statement that is susceptible to proof
- A proposition P is modeled as a type;
- If the proposition is true, the corresponding type is inhabited, i.e., it is possible to provide evidence for P using one of the elements of the type P;
- If the proposition is false, the corresponding type is empty, i.e., it is impossible to provide evidence for P;
- Dependent functions give us $\forall$; dependent pairs give us $\exists$.

Propositions as types (ctd.)

$$\neg : \text{Set} \rightarrow \text{Set}$$
$$\neg\ A = A \rightarrow \bot$$

$$\text{taut1} : \{A\ B : \text{Set}\} \rightarrow \neg\ A \rightarrow \neg\ B \rightarrow \neg\ (A \uplus B)$$
$$\text{taut1}\ na\ nb\ (\text{inj}_1\ a) = na\ a$$
$$\text{taut1}\ na\ nb\ (\text{inj}_2\ b) = nb\ b$$

$$\text{taut2} : \{A : \text{Set}\} \rightarrow \neg\ (\neg\ (\neg\ A)) \rightarrow \neg\ A$$
$$\text{taut2}\ nnna = \lambda\ a \rightarrow nnna\ (\lambda\ na \rightarrow na\ a)$$

$$\text{taut3} : \{A : \text{Set}\} \rightarrow \neg\ (\neg\ (A \uplus \neg\ A))$$
$$\text{taut3} = \lambda\ nana \rightarrow nana\ (\text{inj}_2\ (\lambda\ a \rightarrow nana\ (\text{inj}_1\ a)))$$

Identity types

- The question of whether two elements of a type are equal is clearly a proposition
- This proposition corresponds to a type:

```
data _≡_ {A : Set} : (a b : A) → Set where
  refl : (a : A) → (a ≡ a)

i0 : 3 ≡ 3
i0 = refl 3

i1 : (1 + 2) ≡ (3 * 1)
i1 = refl 3
```

Identity types and paths

- We will interpret $x \equiv y$ as a path from x to y
- If x and y are themselves paths, then $x \equiv y$ as a path between paths, i.e., a homotopy
- We can continue this game to get paths between paths between paths between paths etc.
- What are the recursion and induction principle for these paths?

```
- recursion principle
indiscernability : {A : Set} {C : A → Set} {x y : A} →
    (p : x ≡ y) → C x → C y
indiscernability (refl x) c = c
```

K vs. J

Bad version:

$$
\begin{aligned}
&\mathsf{K} : \{A : \mathsf{Set}\}\ (C : \{x : A\} \to x \equiv x \to \mathsf{Set}) \to \\
&\quad (\forall\ x \to C\ (\mathsf{refl}\ x)) \to \\
&\quad \forall\ \{x\}\ (p : x \equiv x) \to C\ p \\
&\mathsf{K}\ C\ c\ (\mathsf{refl}\ x) = c\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{proof\text{-}irrelevance} : \{A : \mathsf{Set}\}\ \{x\ y : A\}\ (p\ q : x \equiv y) \to \quad p \equiv q \\
&\mathsf{proof\text{-}irrelevance}\ (\mathsf{refl}\ x)\ (\mathsf{refl}\ .x) = \mathsf{refl}\ (\mathsf{refl}\ x)
\end{aligned}
$$

Path induction

Good version (goes back to Leibniz)

```
-- J
pathInd : {A : Set} → (C : {x y : A} → x ≡ y → Set) →
  (c : (x : A) → C (refl x)) →
  ({x y : A} (p : x ≡ y) → C p)
pathInd C c (refl x) = c x

-- for comparison
K' : {A : Set} (C : {x : A} → x ≡ x → Set) →
  (∀ x → C (refl x)) →
  ∀ {x} (p : x ≡ x) → C p
K' C c (refl x) = c x
```

Intensionality

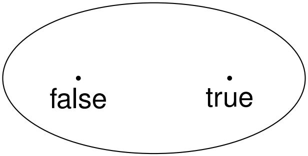
- If two terms x and y are definitionally equal, then $x \equiv y$
- The converse is not true
- This gives rise to a structure of great combinatorial complexity

Homotopy Type Theory

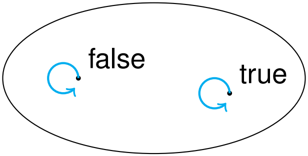
- Martin-Löf type theory
- Identity types with path induction
- Univalence
- Higher-Order Inductive Types

Types as spaces or groupoids

We are used to think of types as sets of values. So we think of the type Bool as:

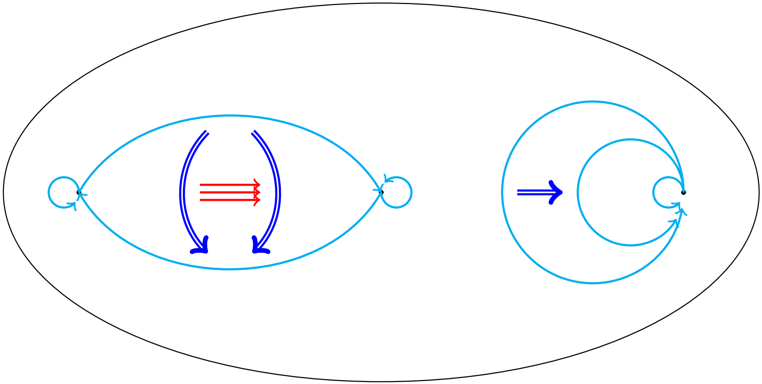


In HoTT, we should instead think about it as:



Types as spaces or groupoids

In this particular case, it makes no difference, but in general we might have something like:



Additional structure

- For every path $p : x \equiv y$, there exists a path $!p : y \equiv x$;
- For every paths $p : x \equiv y$ and $q : y \equiv z$, there exists a path $p \circ q : x \equiv z$;
- Subject to the following conditions:
  - $p \circ \mathit{refl}\ y \equiv p$
  - $p \equiv \mathit{refl}\ x \circ p$
  - $!p \circ p \equiv \mathit{refl}\ y$
  - $p \circ !p \equiv \mathit{refl}\ x$
  - $!\,(!p) \equiv p$
  - $p \circ (q \circ r) \equiv (p \circ q) \circ r$
- With similar conditions one level up and so on and so forth.

Pause

LaTeX crash ...
Switch to third talk