Inductive Types in Lean

Freek Wiedijk

Radboud University Nijmegen
Type Theory and Coq 2021
Reading List Presentations

2021-12-03

nat in Lean

```
namespace demo

inductive nat : Type
| zero : nat
| succ : nat -> nat

#print nat
#print nat.zero
#print nat.succ
#print nat.rec
```

nat in Lean

```
namespace demo

inductive nat : Type
| zero : nat
| succ : nat -> nat

#print nat
#print nat.zero
#print nat.succ
#print nat.rec
```

7:0: information: print result

```
inductive demo.nat : Type
constructors:
demo.nat.zero : nat
demo.nat.succ : nat → nat
```

nat in Lean

```
namespace demo

inductive nat : Type
| zero : nat
| succ : nat -> nat

#print nat
#print nat.zero
#print nat.succ
#print nat.rec
```

8:0: information: print result

```
constructor demo.nat.zero : nat
```

9:0: information: print result

```
constructor demo.nat.succ : nat → nat
```

nat in Lean

```
namespace demo

inductive nat : Type
| zero : nat
| succ : nat -> nat

#print nat
#print nat.zero
#print nat.succ
#print nat.rec
```

10:0: information: print result

```
protected eliminator demo.nat.rec : Π C : nat → Sort l,
C nat.zero → (Π (a : nat), C a → C (nat.succ a)) →
Π (n : nat), C n
```

add in Lean: the equation compiler

```
def add : nat -> nat -> nat
| nat.zero n := n
| (nat.succ m) n := nat.succ (add m n)

#print add
```

add in Lean: the equation compiler

```
def add : nat -> nat -> nat
| nat.zero n := n
| (nat.succ m) n := nat.succ (add m n)

#print add
```

16:0: information: print result

```
def demo.add : nat → nat → nat :=
add._main
```

add in Lean: the equation compiler

```
def add : nat -> nat -> nat
| nat.zero n := n
| (nat.succ m) n := nat.succ (add m n)

#print add
#print add._main
```

17:0: information: print result

```
def demo.add._main : nat → nat → nat :=
λ (a a_1 : nat),
  nat.brec_on a
    (λ (a : nat) (_F : nat.below (λ (a : nat), nat → nat) a) (a_1 : nat),
       (λ (a a_1 : nat) (_F : nat.below (λ (a : nat), nat → nat) a),
          nat.cases_on a (λ (_F : nat.below (λ (a : nat), nat → nat)
nat.zero), id_rhs nat a_1)
            (λ (a_1_1 : nat) (_F : nat.below (λ (a : nat), nat → nat)
(nat.succ a_1_1)),
               id_rhs nat (nat.succ ((_F.fst).fst a_1)))
            _F)
         a
         a_1
         _F)
    a_1
```

differences between Lean and Coq

- **Lean:** Haskell style definitions (`def`) defined in terms of recursors (`nat.rec`)

differences between Lean and Coq

- **Lean:** Haskell style definitions (`def`) defined in terms of recursors (`nat.rec`)
- **Coq:** recursors (`nat_rec`) defined in terms of fixpoints (`fix`/`match`)

```
Coq < Print nat_rect.
nat_rect =
fun (P : nat -> Type) (f : P 0) (f0 : forall n : nat, P n -> P (S n)) =>
fix F (n : nat) : P n :=
  match n as n0 return (P n0) with
  | 0 => f
  | S n0 => f0 n0 (F n0)
  end
     : forall P : nat -> Type,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n
```

differences between Lean and Coq

- **Lean:** Haskell style definitions (`def`) defined in terms of recursors (`nat.rec`)
- **Coq:** recursors (`nat_rec`) defined in terms of fixpoints (`fix`/`match`)

```
Coq < Print nat_rect.
nat_rect =
fun (P : nat -> Type) (f : P 0) (f0 : forall n : nat, P n -> P (S n)) =>
fix F (n : nat) : P n :=
  match n as n0 return (P n0) with
  | 0 => f
  | S n0 => f0 n0 (F n0)
  end
     : forall P : nat -> Type,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n
```

recursors in the logic: simpler, less powerful

differences between Lean and Coq

- **Lean:** Haskell style definitions (`def`) defined in terms of recursors (`nat.rec`)
- **Coq:** recursors (`nat_rec`) defined in terms of fixpoints (`fix`/`match`)

```
Coq < Print nat_rect.
nat_rect =
fun (P : nat -> Type) (f : P 0) (f0 : forall n : nat, P n -> P (S n)) =>
fix F (n : nat) : P n :=
  match n as n0 return (P n0) with
  | 0 => f
  | S n0 => f0 n0 (F n0)
  end
     : forall P : nat -> Type,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n
```

recursors in the logic: simpler, less powerful

- **Coq:** mutual/nested inductive types in the logic
- **Coq:** co-inductive types in the logic
- **Lean:** 'K-like reduction'

term syntax

$$e \to e$$

$$e ::= x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e$$

term syntax

$$e \to e$$

$$\begin{aligned} e ::= {} & x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid \\ & \mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid \\ & \mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K} \end{aligned}$$

term syntax

$$e \to e$$

$$e ::= x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid$$

$$\mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid$$

$$\mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K}$$

term syntax

$$e \to e$$

$$e ::= x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid$$
$$\mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid$$
$$\mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K}$$

$$K ::= 0 \mid (c : e) + K$$

term syntax

$$e \to e$$

$$\begin{aligned} e ::= {} & x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid \\ & \mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid \\ & \mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K} \end{aligned}$$

$$K ::= 0 \mid (c : e) + K$$

type universes:

$$\mathbb{P} = \mathsf{U}_0 \ : \ \mathsf{U}_1 \ : \ \mathsf{U}_2 \ : \ \mathsf{U}_3 \ : \ \ldots$$

term syntax

$$e \to e$$

$$\begin{aligned} e ::= {} & x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid \\ & \mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid \\ & \mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K} \end{aligned}$$

$$K ::= 0 \mid (c : e) + K$$

type universes:

$$\mathbb{P} = \mathsf{U}_0 \ : \ \mathsf{U}_1 \ : \ \mathsf{U}_2 \ : \ \mathsf{U}_3 \ : \ \ldots$$

$$* \quad : \ \square$$

term syntax

$$e \to e$$

$$e ::= x \mid \mathsf{U}_\ell \mid ee \mid \lambda x : e.\, e \mid \forall x : e.\, e \mid$$
$$\mathsf{let}\ x : e := e\ \mathsf{in}\ e \mid c_{\bar{\ell}} \mid$$
$$\mu x : e.\, K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K}$$

$$K ::= 0 \mid (c : e) + K$$

type universes:

$$\mathbb{P} = \mathsf{U}_0 : \mathsf{U}_1 : \mathsf{U}_2 : \mathsf{U}_3 : \ldots$$
$$* : \Box$$
$$\mathsf{Prop} : \mathsf{Set} : \mathsf{Type}_1 : \mathsf{Type}_2 : \ldots$$

example: $\mathbb{N}$

$$\mu x : e.\, K$$

$$\mathbb{N} := \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N) \;:\; \mathsf{U}_1$$

example: $\mathbb{N}$

$$\mu x : e.\, K$$

$$\mathbb{N} := \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N) \;:\; \mathsf{U}_1$$

$$c_{\mu x:e.K}$$

$$z_{\mathbb{N}} : \mathbb{N}$$

$$s_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$$

$$\mathsf{rec}_{\mu x:e.K}$$

$$\begin{aligned} \mathsf{rec}_{\mathbb{N}} : {} & \forall (C : \mathbb{N} \to \mathsf{U}_u). \\ & C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \to \\ & \forall n : \mathbb{N}.\, C\, n \end{aligned}$$

reduction rules for $\mathbb{N}$

$$f\, z_{\mathbb{N}} = g$$
$$f(s_{\mathbb{N}} x) = h\, x\, (f\, x)$$

reduction rules for $\mathbb{N}$

$$
\begin{aligned}
f\, z_{\mathbb{N}} &= g && : C\, z_{\mathbb{N}} \\
f\,(s_{\mathbb{N}} x) &= h\, x\, (f\, x) && : C\,(s_{\mathbb{N}} x)
\end{aligned}
$$

$$
f = \mathsf{rec}_{\mathbb{N}}\, C\, g\, h
$$

reduction rules for $\mathbb{N}$

$$\begin{aligned} f\, z_{\mathbb{N}} &= g && : C\, z_{\mathbb{N}} \\ f\,(s_{\mathbb{N}} x) &= h\, x\, (f\, x) && : C\,(s_{\mathbb{N}} x) \end{aligned}$$

$$f = \mathsf{rec}_{\mathbb{N}}\, C\, g\, h$$

$\mathsf{rec}_{\mathbb{N}}$ *computes:*

$$\begin{aligned} \mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, z_{\mathbb{N}} &\equiv g \\ \mathsf{rec}_{\mathbb{N}}\, C\, g\, h\,(s_{\mathbb{N}} x) &\equiv h\, x\,(\mathsf{rec}_{\mathbb{N}}\, C\, g\, h) \end{aligned}$$

reduction rules for $\mathbb{N}$

$$\begin{aligned} f\, z_{\mathbb{N}} &= g && : C\, z_{\mathbb{N}} \\ f\,(s_{\mathbb{N}} x) &= h\, x\, (f\, x) && : C\,(s_{\mathbb{N}} x) \end{aligned}$$

$$f = \mathsf{rec}_{\mathbb{N}}\, C\, g\, h$$

$\mathsf{rec}_{\mathbb{N}}$ *computes:*

$$\begin{aligned} \mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, z_{\mathbb{N}} &\rightsquigarrow_{\iota} g \\ \mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, (s_{\mathbb{N}} x) &\rightsquigarrow_{\iota} h\, x\, (\mathsf{rec}_{\mathbb{N}}\, C\, g\, h) \end{aligned}$$

the general pattern

$$\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_u).\, C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \\ \to \forall n : \mathbb{N}.\, C\, n$$

the general pattern

$$\mathbb{N} = \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N)$$

$$\mathsf{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \to \mathsf{U}_u).\, C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \to \forall n : \mathbb{N}.\, C\, n$$

the general pattern

$$\mathbb{N} = \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N)$$

$$\mathsf{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \to \mathsf{U}_u).\, C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \to \forall n : \mathbb{N}.\, C\, n$$

$$P = \mu (t : \forall \bar{a} :: \bar{\alpha}.\, \mathsf{U}_\ell).\, \Sigma_c (c : \forall \bar{b} :: \bar{\beta}.\, t\, \bar{p}[\bar{b}])$$

$$\mathsf{rec}_P : \forall (C : \forall \bar{a} :: \bar{\alpha}.\, P\bar{a} \to \mathsf{U}_u).\, \forall \bar{e} :: \bar{\varepsilon}.\, \forall \bar{a} :: \bar{\alpha}.\, \forall z : P\bar{a}.\, C\bar{a}z$$

$$\varepsilon_c := \forall \bar{b} :: \bar{\beta}.\, \forall \bar{v} :: \bar{\delta}.\, C\, \bar{p}[\bar{b}]\, (c\, \bar{b})$$

the general pattern

$$\mathbb{N} = \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N)$$

$$\mathsf{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \to \mathsf{U}_u).\, C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \to \forall n : \mathbb{N}.\, C\, n$$

$$P = \mu (t : \forall \bar{a} :: \bar{\alpha}.\, \mathsf{U}_\ell).\, \Sigma_c (c : \forall \bar{b} :: \bar{\beta}.\, t\, \bar{p}[\bar{b}])$$

$$\mathsf{rec}_P : \forall (C : \forall \bar{a} :: \bar{\alpha}.\, P\bar{a} \to \mathsf{U}_u).\, \forall \bar{e} :: \bar{\varepsilon}.\, \forall \bar{a} :: \bar{\alpha}.\, \forall z : P\bar{a}.\, C\bar{a}z$$

$$\varepsilon_c := \forall \bar{b} :: \bar{\beta}.\, \forall \bar{v} :: \bar{\delta}.\, C\, \bar{p}[\bar{b}]\, (c\, \bar{b})$$

$$\mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, z_{\mathbb{N}} \rightsquigarrow_\iota g$$

$$\mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, (s_{\mathbb{N}} x) \rightsquigarrow_\iota h\, x\, (\mathsf{rec}_{\mathbb{N}}\, C\, g\, h)$$

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$

$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

the general pattern

$$\mathbb{N} = \mu N : \mathsf{U}_1.\, (z : N) + (s : N \to N)$$

$$\mathsf{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \to \mathsf{U}_u).\, C\, z_{\mathbb{N}} \to (\forall a : \mathbb{N}.\, C\, a \to C(s_{\mathbb{N}}\, a)) \to \forall n : \mathbb{N}.\, C\, n$$

$$P = \mu (t : \forall \bar{a} :: \bar{\alpha}.\, \mathsf{U}_\ell).\, \Sigma_c (c : \forall \bar{b} :: \bar{\beta}.\, t\, \bar{p}[\bar{b}])$$

$$\mathsf{rec}_P : \forall (C : \forall \bar{a} :: \bar{\alpha}.\, P\bar{a} \to \mathsf{U}_u).\, \forall \bar{e} :: \bar{\varepsilon}.\, \forall \bar{a} :: \bar{\alpha}.\, \forall z : P\bar{a}.\, C\bar{a}z$$

$$\varepsilon_c := \forall \bar{b} :: \bar{\beta}.\, \forall \bar{v} :: \bar{\delta}.\, C\, \bar{p}[\bar{b}]\, (c\, \bar{b})$$

$$\mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, z_{\mathbb{N}} \rightsquigarrow_\iota g$$

$$\mathsf{rec}_{\mathbb{N}}\, C\, g\, h\, (s_{\mathbb{N}} x) \rightsquigarrow_\iota h\, x\, (\mathsf{rec}_{\mathbb{N}}\, C\, g\, h)$$

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$

$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

strict positivity

$$\Gamma \vdash e : \alpha \qquad \Gamma \vdash e \equiv e'$$

$$\Gamma \vdash \alpha \text{ type} \qquad \Gamma \vdash e \Leftrightarrow e'$$

$$\Gamma \text{ ok} \qquad e \rightsquigarrow_{\kappa} e' \qquad \kappa = \beta\zeta\delta\iota$$

$$\ell \equiv \ell'$$

$$\ell \leq \ell' + n$$

strict positivity

$$\Gamma \vdash e : \alpha \qquad \Gamma \vdash e \equiv e'$$

$$\Gamma \vdash \alpha \text{ type} \qquad \Gamma \vdash e \Leftrightarrow e'$$

$$\Gamma \text{ ok} \qquad e \rightsquigarrow_{\kappa} e' \qquad \kappa = \beta\zeta\delta\iota$$

$$\ell \equiv \ell'$$

$$\ell \leq \ell' + n$$

$$\Gamma; t : F \vdash \alpha \text{ ctor}$$

$$\Gamma; t : F \vdash K \text{ spec}$$

strict positivity

$$\Gamma \vdash e : \alpha \qquad \Gamma \vdash e \equiv e'$$
$$\Gamma \vdash \alpha \text{ type} \qquad \Gamma \vdash e \Leftrightarrow e'$$
$$\Gamma \text{ ok} \qquad e \rightsquigarrow_\kappa e' \qquad \kappa = \beta\zeta\delta\iota$$
$$\ell \equiv \ell'$$
$$\ell \leq \ell' + n$$

$$\Gamma; t : F \vdash \alpha \text{ ctor}$$
$$\Gamma; t : F \vdash K \text{ spec}$$

$$N : \mathsf{U}_1 \vdash N \text{ ctor}$$
$$N : \mathsf{U}_1 \vdash (N \to N) \text{ ctor}$$
$$N : \mathsf{U}_1 \vdash (z : N) + (s : N \to N) \text{ spec}$$

$$t = N$$
$$F = \mathsf{U}_1$$

large elimination

induction principle: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathbb{P}). \ldots$
$\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_0). \ldots$

recursor: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_\ell). \ldots$ with $\ell > 0$
$=$ 'large elimination'

large elimination

induction principle: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathbb{P}). \ldots$
$\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_0). \ldots$

recursor: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_\ell). \ldots$ with $\ell > 0$
$=$ 'large elimination'

two kinds of inductive types with large elimination:

- type lives in universe U_ℓ with $\ell > 0$
  $\mathbb{N} : \mathsf{U}_1$
- subsingleton elimination:
  - at most one constructor
  - non-recursive constructor arguments all are propositions or appear in output type

large elimination

induction principle: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathbb{P}). \ldots$
$\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_0). \ldots$

recursor: $\mathsf{rec}_{\mathbb{N}} : \forall(C : \mathbb{N} \to \mathsf{U}_\ell). \ldots$ with $\ell > 0$
$=$ 'large elimination'

two kinds of inductive types with large elimination:

- type lives in universe U_ℓ with $\ell > 0$
  $\mathbb{N} : \mathsf{U}_1$
- subsingleton elimination:
  - at most one constructor
  - non-recursive constructor arguments all are propositions or appear in output type

$$\Gamma; t : F \vdash \alpha \text{ ctor} \qquad \Gamma; t : F \vdash K \text{ spec}$$
$$\Gamma; t : F \vdash \alpha \text{ LE ctor} \qquad \Gamma; t : F \vdash K \text{ LE}$$

K-like reduction

normal reduction for constructor c:

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$

K-like reduction

normal reduction for constructor c:

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$
$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

K-like reduction

normal reduction for constructor c:

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$
$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

'K-like reduction' for subsingleton eliminators

$$\frac{P : \forall \bar{a} :: \bar{\alpha}.\, \mathbb{P}}{\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, h \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}}$$

K-like reduction

normal reduction for constructor c:

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$
$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

'K-like reduction' for subsingleton eliminators

$$\frac{P : \forall \bar{a} :: \bar{\alpha}.\, \mathbb{P}}{\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, h \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}}$$

```
inductive eq {α : Sort u} (a : α) : α → Prop
| refl [] : eq a
```

$$(\mathsf{rec}_{\mathtt{eq}}\, \alpha\, a)\, C\, x\, a\, h \rightsquigarrow_\iota x$$
$$x : Ca$$
$$h : a =_\alpha a$$

K-like reduction

normal reduction for constructor c:

$$\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, (c\, \bar{b}) \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}$$
$$v_i := \lambda \bar{x} :: \bar{\xi}_i.\, \mathsf{rec}_P\, C\, \bar{e}\, \pi_i[\bar{b}, \bar{x}]\, (u_i\, \bar{x})$$

'K-like reduction' for subsingleton eliminators

$$\frac{P : \forall \bar{a} :: \bar{\alpha}.\, \mathbb{P}}{\mathsf{rec}_P\, C\, \bar{e}\, \bar{p}[\bar{b}]\, h \rightsquigarrow_\iota e_c\, \bar{b}\, \bar{v}}$$

```
inductive eq {α : Sort u} (a : α) : α → Prop
| refl [] : eq a
```

$$(\mathsf{rec}_{\mathtt{eq}}\, \alpha\, a)\, C\, x\, a\, h \rightsquigarrow_\iota x$$
$$x : C a$$
$$h : a =_\alpha a$$

axiom K $\iff$ UIP = uniqueness of identity proofs
no homotopy type theory in Lean ≥ 3

conclusion

- inductive types in Lean simpler than in Coq
- more powerful types (mutual/nested, co-inductive)
  implemented on top of this using coding and category theory

  $\longrightarrow$ approach from Isabelle/HOL
  (talk by Márk Széles in the MFoCS seminar)