

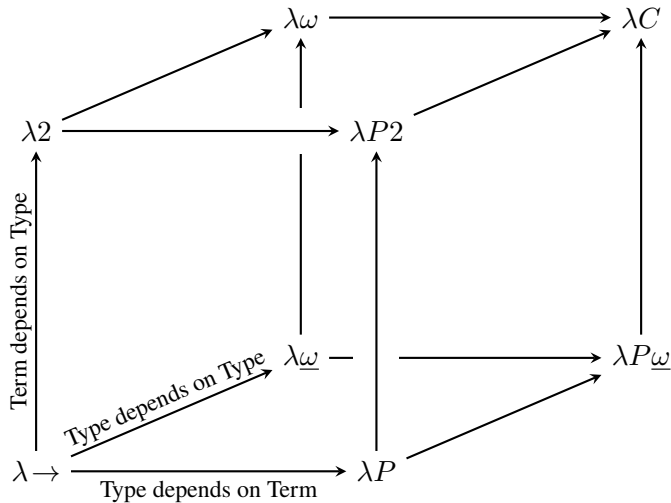
MiniProver

A coq-like proof assistant

Zhenwen Li, Sirui Lu, Kewen Wu

Peking University

June 17, 2018

λ -cube

Example and CIC

- Term depends on Type

$$\lambda T : \text{Type}. \lambda x : T. x \quad : \quad \Pi T : \text{Type}. T \rightarrow T$$

- Type depends on Type

$$\lambda T : \text{Type}. T \rightarrow T \quad : \quad \text{Type} \rightarrow \text{Type}$$

- Type depends on Term

$$\lambda n : \text{nat}. S \, n \quad : \quad \text{nat} \rightarrow \text{Type}$$

- $\lambda C + \text{Definition} \Rightarrow \lambda D.$

- $\lambda D + \text{Inductive Type} \Rightarrow \mathbf{CIC}.$

Inductive Type

$$\text{Ind } [p] \ (\Gamma_I := \Gamma_C)$$

- p : Number of parameters.
- Γ_I : Inductive type.
- Γ_C : Constructors of the inductive type.

Inductive `lst (T : Type) : Type :=`
 | `nil : lst T`
 | `cons : T -> lst T -> lst T`

$$\text{Ind } [1] \left([\text{lst} : \text{Type} \rightarrow \text{Type}] := \left[\begin{array}{l} \text{nil} : \Pi T : \text{Type}, \text{lst } T \\ \text{cons} : \Pi T : \text{Type}, T \rightarrow \text{lst } T \rightarrow \text{lst } T \end{array} \right] \right)$$

Match

`match t as x_0 in namelist return returntype with [equation]`

- t : A term of inductive type.
- *namelist* : Arguments of the inductive type.
- *equation* : Different constructors of the inductive type.

```
Inductive eq (T : Type) (x : T) : T -> Type :=  
  | eq_refl : eq T x x
```

```
fun (n : nat) (m : nat) (e : eq nat n m) =>  
  match e in (eq _ _ y) return (eq nat y n) with  
  | eq_refl _ _ => ...
```

Termination Check

```
Fixpoint plus (n m : nat) : nat :=  
  match n with  
  | 0 => m  
  | S p => S (plus p m)
```

```
Fixpoint plus' (n m : nat) : nat :=  
  match n with  
  | 0 => m  
  | S p => S (plus m p)
```

Criterion

Descending on at least one variable.

Positivity Check

Inductive `ill` : **Type** :=
| `malf` : (`ill` -> `ill`) -> `ill`

Definition `extract` (`t` : `ill`) : `ill` :=
 `match` `t` `with`
 | `malf` `f` => `f t`

`extract (malf extract)` *(* not terminating *)*

Criterion

Types of constructors satisfy **positivity condition** for name of inductive type.

Build Term and Type for Inductive Definition

Intuition

View mathematical induction as building a term for inductive definition.

Inductive **nat** : **Type** := 0 : **nat** | S : **nat** -> **nat**

```

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 in (nat) return (P n0) with
    | 0 => f
    | S n0 => f0 n0 (F n0)

:
forall P : nat -> Type, P 0 ->
  (forall n : nat, P n -> P (S n)) ->
    forall n : nat, P n
  
```

Evaluation (Conversion)

- β -reduction : Reduce *application*
- ι -reduction : Reduce *match*
- δ -reduction : Definition $x := u \Rightarrow x \rightarrow u$
- ζ -reduction : $\text{let } x := u \text{ in } t \Rightarrow [x \rightarrow u]t$
- η -expansion : $t : \forall x : T, U \Rightarrow \lambda x : T. (t \ x)$

Proof Handling

- Build proof object.
- Navigate : Undo.
- Switch mode : Proof, Qed, Abort.
- Request info : Print.

Tactics

- Intro, Intros : $\vdash A \rightarrow B \Rightarrow A \vdash B$
- Apply : $A \rightarrow B \vdash B \Rightarrow A \rightarrow B \vdash A$
- Exact : Construct the term manually
- Reflexivity : $\text{eq } T \ x \ y \Rightarrow \text{check if } x =_{\beta\delta\iota\eta\zeta} y$
- Split : $\vdash P_1 \wedge P_2 \Rightarrow \vdash P_1, P_2$
- Induction : Classified sub-proofs with induction hypothesis
- Equivalence : Build equivalence relations inside inductive term

Tactics

- **Simpl** : Simplify into human-readable goal
- **Destruct** : Classified sub-proofs without induction hypothesis
- **Left, Right** : $\vdash P_1 \vee P_2 \Rightarrow \vdash P_1$ or $\vdash P_2$
- **Symmetry** : $\text{eq } T \ x \ y \vdash \text{eq } T \ y \ x$
- **Rewrite** : $\text{eq } T \ x \ y \Rightarrow x \rightarrow y$ or $y \rightarrow x$
- **Unfold** : Replace term with $\beta\iota$ -normal form
- **Exists** : $\vdash \exists x, P \ x \Rightarrow x_0 \wedge (P \ x_0)$

Thanks!