MiniProver

A coq-like proof assistant

Zhenwen Li, Sirui Lu, Kewen Wu

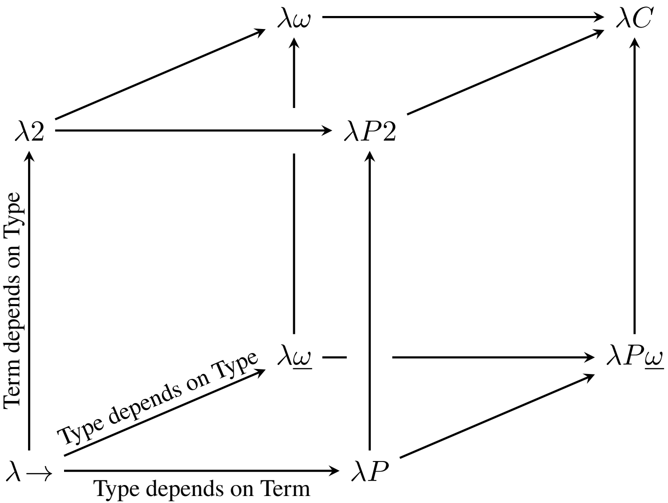
Peking University

June 14, 2018

Accomplishment

- Encode the dependent lambda calculus.
- Inductive types and induction rules.
- Fix point operator and termination checking.
- Dependent pattern matching.
- Tactic-based proving and proof object reconstruction.
- Formalize some interesting proofs in our system.

λ-cube


$\lambda\omega$
λC
$\lambda 2$
$\lambda P2$
$\lambda\underline{\omega}$
$\lambda P\underline{\omega}$
$\lambda\!\to$
λP
Term depends on Type
Type depends on Type
Type depends on Term

Example and **CIC**

- Term depends on Type
  $\lambda \mathtt{T} : \mathtt{Type}.\ \lambda \mathtt{x} : \mathtt{T}.\ \mathtt{x} \quad : \quad \Pi \mathtt{T} : \mathtt{Type}.\ \mathtt{T} \to \mathtt{T}$
- Type depends on Type
  $\lambda \mathtt{T} : \mathtt{Type}.\ \mathtt{T} \to \mathtt{T} \quad : \quad \mathtt{Type} \to \mathtt{Type}$
- Type depends on Term
  $\lambda \mathtt{n} : \mathtt{nat}.\ \mathtt{S}\ \mathtt{n} \quad : \quad \mathtt{nat} \to \mathtt{Type}$

- λC + *Definition* $\Rightarrow \lambda D$.
- λD + *Inductive Type* $\Rightarrow$ **CIC**.

Inductive Type

$$\texttt{Ind}\ [p]\ (\Gamma_{\texttt{I}} := \Gamma_{\texttt{C}})$$

- p : Number of parameters.
- $\Gamma_{\texttt{I}}$: Inductive type.
- $\Gamma_{\texttt{C}}$: Constructors of the inductive type.

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

$$\texttt{Ind}\ [1] \left([\texttt{lst} : \texttt{Type} \to \texttt{Type}] := \begin{bmatrix} \texttt{nil} : \Pi\texttt{T} : \texttt{Type},\ \texttt{lst T} \\ \texttt{cons} : \Pi\texttt{T} : \texttt{Type},\ \texttt{T} \to \texttt{lst T} \to \texttt{lst T} \end{bmatrix} \right)$$

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
    | O => m
    | S p => S (plus p m)

Fixpoint plus' (n m : nat) : nat :=
    match n with
    | O => 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 := O : nat | S : nat -> nat

fun (P : nat -> Type) (f : P O)
        (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
        | O => f
        | S n0 => f0 n0 (F n0)
:
forall P : nat -> Type, P O ->
    (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 : $\texttt{Definition}\,\texttt{x} := \texttt{u} \quad \Rightarrow \quad \texttt{x} \to \texttt{u}$
- ζ-reduction : $\texttt{let}\,\texttt{x} := \texttt{u}\,\texttt{in}\,\texttt{t} \quad \Rightarrow \quad [\texttt{x} \to \texttt{u}]\texttt{t}$
- η-expansion : $\texttt{t} : \forall \texttt{x} : \texttt{T}, \texttt{U} \quad \Rightarrow \quad \lambda \texttt{x} : \texttt{T}.\ (\texttt{t}\ \texttt{x})$

Proof Handling

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

Tactics

- Intro, Intros : $\vdash \mathtt{A} \rightarrow \mathtt{B} \Rightarrow \mathtt{A} \vdash \mathtt{B}$
- Apply : $\mathtt{A} \rightarrow \mathtt{B} \vdash \mathtt{B} \Rightarrow \mathtt{A} \rightarrow \mathtt{B} \vdash \mathtt{A}$
- Exact : Construct the term manually
- Reflexivity : $\mathtt{eq}\ \mathtt{T}\ \mathtt{x}\ \mathtt{y} \Rightarrow$ check if $\mathtt{x} =_{\beta\delta\iota\eta\zeta} \mathtt{y}$
- Split : $\vdash \mathtt{P}_1 \wedge \mathtt{P}_2 \Rightarrow \vdash \mathtt{P}_1, \mathtt{P}_2$
- 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 : $\vdash \mathtt{P_1} \vee \mathtt{P_2} \Rightarrow \vdash \mathtt{P_1}$ or $\vdash \mathtt{P_2}$
- Symmetry : $\mathtt{eq\ T\ x\ y} \vdash \mathtt{eq\ T\ y\ x}$
- Rewrite : $\mathtt{eq\ T\ x\ y} \Rightarrow \mathtt{x} \rightarrow \mathtt{y}$ or $\mathtt{y} \rightarrow \mathtt{x}$
- Unfold : Replace term with $\beta\iota$-normal form
- Exists : $\vdash \exists \mathtt{x},\ \mathtt{P\ x} \Rightarrow \mathtt{x_0} \wedge (\mathtt{P\ x_0})$

Thanks!