MPRI 2-7-2: Proof Assistants

Bruno Barras, Matthieu Sozeau

Jan 5, 2017

Recap: Calculus of Constructions (CC)

Features:

- Pure Type Systems with 2 sorts (**Prop**: **Type**) or ($* : \Box$)
- Curry-Howard: propositions as types / proofs as terms
- Dependent types
- Polymorphism (impredicativity of $*$)

Expressivity:

- Propositional and predicate (higher-order) logic (OK)
- Datatypes (limited, see last week's TP...)

Datatypes

Most useful datatypes can be encoded in Peano Arithmetic:

- Natural numbers (obviously), rational numbers, ...
- Lists
- Finitely branching trees, ...

Datatypes

Most useful datatypes can be encoded in Peano Arithmetic:

- Natural numbers (obviously), rational numbers, ...
- Lists
- Finitely branching trees, ...

... in theory, but awkward in practice!

Datatypes

Most useful datatypes can be encoded in Peano Arithmetic:

- Natural numbers (obviously), rational numbers, ...
- Lists
- Finitely branching trees, ...

... in theory, but awkward in practice!

$\Rightarrow$ Calculus of Inductive Constructions:
Calculus of Constructions + (co)Inductive Types (Coquand, Paulin 1989)

Plan

Inductive sets/types

Simple Inductive Types

Inductive Types with Parameters

Inductive sets

Induction is a very general principle that has many instances in mathematics.

Examples of inductive sets:

- Natural numbers ($\Rightarrow$ mathematical induction)
- Sets/Subsets defined by inference rules
- Generalization to well-founded trees (structural induction)

Natural numbers in Peano Arithmetic

Peano Arithmetic (PA)

- 0 is a natural number;
- if n is a natural number, then $S(n)$ is a natural number;
- equational theory: add, mult, discrimination, injectivity;
- induction scheme:
  $P(0)$ and $\forall n.\, P(n) \Rightarrow P(S(n))$ implies $\forall n. P(n)$

Inference rules in PA

Defines subsets of $\mathbb{N}$:

- even numbers $2\mathbb{N}$

$$\frac{}{0 \in 2\mathbb{N}} \qquad \frac{n \in 2\mathbb{N}}{S(S(n)) \in 2\mathbb{N}}$$

Minimality: any set closed by the above rules is larger than $2\mathbb{N}$:
$P(0)$ and $\forall n.\, P(n) \Rightarrow P(S(S(n)))$ implies $\forall n \in 2\mathbb{N}.P(n)$

Inference rules: beyond mere arithmetic

The previous schemes suffices to modelize inference rules:

- Syntax of (lists of) λ-terms (AST) as a subset of $\mathbb{N}$.
- Typing rules are inference rules that define a subset D of judgments that are derivable

$$\frac{(\Gamma, M, \tau \to \tau') \in D \qquad (\Gamma, N, \tau) \in D}{(\Gamma,\ M\,N,\ \tau') \in D}$$

Inference rules in set theory

In set theory, inference rules can be used to define collections
$\Rightarrow$ Inductive set

Example: natural numbers

$$\frac{}{0 \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{S(n) \in \mathbb{N}}$$

Collections X with closure condition:

$$0 \in X \wedge \forall n.\, n \in X \Rightarrow S(n) \in X$$

- Under a monotonicity condition (not detailed here), the collections with the above closure condition are closed by arbitrary intersection
- Under further conditions, the intersection of all collections with the above closure condition is a set that we call $\mathbb{N}$.

Natural numbers as an inductive set

Properties of $\mathbb{N}$:

- $\mathbb{N}$ is closed, so it satisfies the expected introduction rules
- The minimality of $\mathbb{N}$ is expressed by the schematic rule

$$\forall P.P(0) \wedge (\forall n \in \mathbb{N}.P(n) \Rightarrow P(S(n))) \Rightarrow \forall n \in \mathbb{N}.P(n)$$

$\Rightarrow$ $\mathbb{N}$ satisfies the Peano axioms.

Inductive sets as fixpoints

Another viewpoint:

- $\mathbb{N}$ is the smallest fixpoint of

$$F(X) = \{0\} \cup \{S(n) \mid n \in X\}$$

- $F(P) \subseteq P$ is the property of closure by rules
- The minimality property

$$\forall P. F(P) \subseteq P \Rightarrow \mathbb{N} \subseteq P$$

rephrases the induction schema

Inference rules: beyond arithmetic

Infinitely branching trees cannot be defined in PA

But can be defined as an inductive set:

$$\frac{}{\textbf{Leaf} \in \mathcal{T}} \qquad \frac{x \in \mathcal{L} \quad f \in \mathbb{N} \to \mathcal{T}}{\textbf{Node}(x, f) \in \mathcal{T}}$$

Inference rules: gone too far...

Consider the rule

$$\frac{x \in \mathcal{P}(V)}{\mathbf{C}(x) \in V}$$

The rules satisfy the monotonicity condition, there exists a smallest collection closed by the rule.

But V is not a set: it is the collection of well-founded sets.

Type of Natural Numbers

Martin-Löf scheme (form/intro/elim/comp):

- 1 formation rule:

$$\frac{}{\vdash \mathbb{N} : \textbf{Type}}$$

- 2 introduction rules:

$$\frac{}{\vdash 0 : \mathbb{N}} \qquad \frac{\vdash n : \mathbb{N}}{\vdash S(n) : \mathbb{N}}$$

- 1 elimination rule ($P : \mathbb{N} \to \textbf{Type}$ as a subset of $\mathbb{N}$)

$$\frac{\begin{array}{c}\vdash P : \mathbb{N} \to \textbf{Type} \qquad \vdash n : \mathbb{N} \\ \vdash f_0 : P(0) \qquad \vdash f_S : \Pi n{:}\mathbb{N}.\, P(n) \to P(S(n))\end{array}}{\vdash Rec(f_0, f_S, n) : P(n)}$$

- 2 computation rules

$$Rec(f_0, f_S, 0) = f_0 \qquad Rec(f_0, f_S, S(n)) = f_S(n, Rec(f_0, f_S, n))$$

Dependent vs non-dependent elimination

The induction scheme:

$$\frac{\vdash P : \mathbb{N} \to \textbf{Type} \quad \vdash n : \mathbb{N} \quad \vdash f_0 : P(0) \quad \vdash f_S : \Pi n{:}\mathbb{N}.\, P(n) \to P(S(n))}{\vdash Rec(f_0, f_S, n) : P(n)}$$

If we drop the dependent types (P is a constant type):

$$\frac{\vdash P : \textbf{Type} \quad \vdash n : \mathbb{N} \quad \vdash f_0 : P \quad \vdash f_S : \mathbb{N} \to P \to P}{\vdash Rec(f_0, f_S, n) : P}$$

$\Rightarrow$ This is the recursor of Gödel's T!

Conclusions:

- Induction scheme and recursor is another instance of the Curry-Howard isomorphism
- The recursor of Gödel's T is a non-dependent specialization of the induction scheme

Inductive types in Coq

Coq provides the user with a general mechanism:

- Inductive type specified by the introduction rules (called constructors)
- A dependent induction/recursion scheme is derived systematically (called eliminator)
- Computation rules derived systematically (ι-reduction)

Comparison with Martin-Löf's inductive types:

- Coq checks the definition preserves consistency (but not complete!)
  $\Rightarrow$ Strictly positive inductive definitions
- Coq allows impredicative inductive definitions (defined later...)
- Coq uses style of Pure Type Systems

Natural numbers in Coq

Declaration of the natural numbers:

```
Inductive nat : Type :=
| O : nat | S : nat -> nat.
```

which defines

- a type $\Gamma \vdash \texttt{nat} : \textbf{Type}$
- a set of introduction rules for this type : constructors

$$\Gamma \vdash O : \texttt{nat} \qquad \frac{\Gamma \vdash n : \texttt{nat}}{\Gamma \vdash S\ n : \texttt{nat}}$$

Recursive inductive types: Natural numbers example

which defines also

- an elimination rule (pattern-matching operator with a result depending on the object which is eliminated)

$$\frac{\Gamma \vdash t : \mathtt{nat} \qquad \Gamma, x{:}\mathtt{nat} \vdash A(x) : s \qquad \Gamma \vdash t_1 : A(O) \qquad \Gamma, n{:}\mathtt{nat} \vdash t_2 : A(S\ n)}{\Gamma \vdash (\mathtt{match}\ t\ \mathtt{as}\ x\ \mathtt{return}\ A(x)\ \mathtt{with}\ O \Rightarrow t_1 \mid S\ n \Rightarrow t_2\ \mathtt{end}) : A(t)}$$

- reduction rules preserve typing (ι-reduction)

$$(\mathtt{match}\ O\ \mathtt{as}\ x\ \mathtt{return}\ A(x)\ \mathtt{with}\ O \Rightarrow t_1 \mid S\ n \Rightarrow t_2\ \mathtt{end}) \rightarrow_\iota t_1$$

$$(\mathtt{match}\ S\ m\ \mathtt{as}\ x\ \mathtt{return}\ A(x)\ \mathtt{with}\ O \Rightarrow t_1 \mid S\ n \Rightarrow t_2\ \mathtt{end}) \rightarrow_\iota t_2[m/n]$$

Recursive inductive types

Example of natural numbers

- We obtain case analysis and construction by cases : the term

$$
\begin{array}{l}
\lambda P : \texttt{nat} \to s. \\
\lambda H_O : P(O). \\
\lambda H_S : \forall m : \texttt{nat}.\, P(S\ m). \\
\lambda n : \texttt{nat}. \\
\quad \texttt{match}\ n\ \texttt{as}\ y\ \texttt{return}\ P(y)\ \texttt{with} \\
\quad \mid O \ \texttt{=>}\ H_O \\
\quad \mid S\ m\ \texttt{=>}\ H_S\ m \\
\quad \texttt{end}
\end{array}
$$

- is a proof of

$$\forall P : \texttt{nat} \to s.\, P(O) \to (\forall m : \texttt{nat}.\, P(S\ m)) \to \forall n : \texttt{nat}.\, P(n)$$

How to derive the standard recursion scheme ?

Fixpoint operator : application

From case analysis to recursor on natural numbers

case-analysis

$\lambda P : \mathtt{nat} \to s,$
$\lambda H_O : P(O),$
$\lambda H_S : \forall m : \mathtt{nat}, P(S\ m),$
$\lambda n : \mathtt{nat},$
$\mathtt{match}\ n\ \mathtt{return}\ P(n)\ \mathtt{with}$
$O \Rightarrow H_O \mid S\ m \Rightarrow H_S\ m$
$\mathtt{end}$

recursor

$\lambda P : \mathtt{nat} \to s,$
$\lambda H_O : P(O),$
$\lambda H_S : \forall m : \mathtt{nat}, P(m) \to P(S\ m),$
$\mathtt{fix}\ f\ (n : \mathtt{nat})\ :\ P(n)\ :=$
$\mathtt{match}\ n\ \mathtt{return}\ P(n)\ \mathtt{with}$
$O \Rightarrow H_O \mid S\ m \Rightarrow H_S\ m\ (f\ m)$
$\mathtt{end}$

has type

$\forall P : \mathtt{nat} \to s,$
$P(O) \to$
$(\forall m : \mathtt{nat}, P(S\ m)) \to$
$\forall n : \mathtt{nat}, P(n)$

has type

$\forall P : \mathtt{nat} \to s,$
$P(O) \to$
$(\forall m : \mathtt{nat}, P(m) \to P(S\ m)) \to$
$\forall n : \mathtt{nat}, P(n)$

Fixpoint operator : well-foundness

Requirement of the Calculus of Inductive Constructions :

- the argument of the fixpoint has type an inductive definition
- recursive calls are on arguments which are *structurally* smaller

Example of recursor on natural numbers

$$
\begin{array}{l}
\lambda P : \texttt{nat} \to s, \\
\lambda H_O : P(O), \\
\lambda H_S : \forall m : \texttt{nat}, P(m) \to P(S\ m), \\
\texttt{fix}\ f\ (n : \texttt{nat})\ :\ P(n)\ := \\
\quad \texttt{match}\ n\ \texttt{as}\ y\ \texttt{return}\ P(y)\ \texttt{with} \\
\qquad O \Rightarrow H_O \mid S\ m \Rightarrow H_S\ m\ (f\ m) \\
\quad \texttt{end}
\end{array}
$$

is correct with respect to CCI : recursive call on m which is structurally smaller than n in the inductive `nat`.

Inductive types with parameters

Example of lists

```
Inductive list (A:Type) : Type :=
| nil : list A
| cons : A -> list A -> list A.
```

which defines

- a family of types $\dfrac{}{\Gamma \vdash \mathit{list} : \mathbf{Type} \to \mathbf{Type}}$
- a set of introduction rules for the types in this family

$$\frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma \vdash \mathtt{nil}_A : \mathit{list}\ A} \qquad \frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma \vdash a : A \quad \Gamma \vdash l : \mathit{list}\ A}{\Gamma \vdash \mathtt{cons}_A\ a\ l : \mathit{list}\ A}$$

Inductive types with parameters

Example of lists : elimination

- An elimination rule (pattern-matching operator with a result depending on the object which is eliminated)

$$\frac{\Gamma \vdash l : list\ A \quad \Gamma, x : list\ A \vdash C(x) : s \qquad \Gamma \vdash t_1 : C(\texttt{nil}) \quad \Gamma, a : A, l : list\ A \vdash t_2 : C(\texttt{cons}_A\ a\ l)}{\Gamma \vdash \left(\begin{array}{l}\texttt{match}\ l\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with} \\ \quad \texttt{nil} \Rightarrow t_1 \mid \texttt{cons}\ a\ l \Rightarrow t_2 \\ \texttt{end}\end{array}\right) : C(l)}$$

- reduction rules which preserve typing (ι-reduction)

$$\left(\begin{array}{l}\texttt{match}\ \texttt{nil}_A\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with} \\ \quad \texttt{nil} \Rightarrow t_1 \mid \texttt{cons}\ a\ l \Rightarrow t_2 \\ \texttt{end}\end{array}\right) \rightarrow_\iota t_1$$

$$\left(\begin{array}{l}\texttt{match}\ \texttt{cons}_A\ a'\ l'\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with} \\ \quad \texttt{nil}\ p \Rightarrow t_1 \mid \texttt{cons}\ a\ l \Rightarrow t_2 \\ \texttt{end}\end{array}\right)$$

$$\rightarrow_\iota t_2[a', l'/a, l]$$

Infinitely branching trees in Coq

Declaration of the infinitely branching trees:

```
Inductive tree (A:Type) : Type :=
| Leaf : tree A
| Node : A -> (nat -> tree A) -> tree A.
tree is defined
tree_rect is defined
tree_ind is defined
tree_rec is defined

tree_rect =
fun (A : Type) (P : tree A->Type) (f : P (Leaf A))
 (f0 : forall (a : A) (t : nat -> tree A),
      (forall n:nat, P (t n)) -> P (Node A a t)) =>
fix F (t : tree A) : P t :=
  match t as t0 return (P t0) with
  | Leaf => f
  | Node y t0 => f0 y t0 (fun n : nat => F (t0 n))
  end
```

Logical connectives

Disjunction example

```
Inductive or (A:Prop) (B:Prop) : Prop :=
| or_introl : A -> or A B
| or_intror : B -> or A B.
```

- General elimination rule

$$\frac{\Gamma \vdash t : or\,A\,B \quad \Gamma, x : or\,A\,B \vdash C(x) : \textbf{Prop} \qquad \Gamma, p : A \vdash t_1 : C\,(or\_introl\,p) \quad \Gamma, q : B \vdash t_2 : C\,(or\_intror\,q))}{\Gamma \vdash \left(\begin{array}{l} \texttt{match}\ t\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with} \\ \quad or\_introl\,p \Rightarrow t_1 \mid or\_intror\,q \Rightarrow t_2 \\ \texttt{end} \end{array}\right) : C(t)}$$

More logical connectives

The other logical connectives:

```
Inductive and (A:Prop) (B:Prop) : Prop :=
| conj : A -> B -> and A B.
Inductive True : Prop := I.
Inductive False : Prop := .
Inductive ex (A:Type)(P:A->Prop) : Prop :=
| ex_intro : forall (x:A), P x -> ex A P.
```

Exercise: guess the type of the generated eliminator.