Overview of the Coq Proof Assistant

Nicolas Magaud
School of Computer Science and Engineering
The University of New South Wales

Guest lecture
Theorem Proving

Outline

- Some Theoretical Background
  - Constructive Logic
  - Curry-Howard Isomorphism
- The Coq Proof Assistant
  - Specification Language: Inductive Definitions
  - Proof Development
  - Practical Use and Demos

Constructive Logic

- Also known as Intuitionistic Logic.
- Does not take the excluded middle rule $A \vee \neg A$ into account !
- Pierce law: $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$
- A proof (of existence) of $\{f \mid P(f)\}$ actually provides an executable function f.
- Application: extraction of programs from proofs

$$\forall a : \mathsf{nat}, \forall b : \mathsf{nat}, \exists q : \mathsf{nat}, r : \mathsf{nat} \ \mid \ a = q * b + r \wedge 0 \leq r < b$$

From this proof, we can compute q and r from a and b.

Natural Deduction

- Propositional Logic (implication fragment)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow_E$$

- Rules for the other Connectives

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{E1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{E2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{I1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{I2} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee_E$$

$$\frac{\Gamma, A \vdash \mathsf{False}}{\Gamma \vdash \neg A} \neg_I \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \mathsf{False}} \neg_E \qquad \frac{\Gamma \vdash \mathsf{False}}{\Gamma \vdash A} \mathsf{False}_E$$

Semantics - Interpretation of a Logic (I)

- Tarski semantics
  - Boolean interpretation of the logic

A	B	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$\neg A \equiv A \Rightarrow False$
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	1	0

Semantics - Interpretation of a Logic (II)

- Heyting-Kolmogorov semantics
  - A proof of $A \Rightarrow B$ is a function
    which for any proof of A yields a proof of B.
  - A proof of $A \wedge B$ is a pair
    featuring a proof of A and a proof of B.
  - A proof of $A \vee B$ is a pair (i, p)
    with ($i = 0$ and p a proof of A) or ($i = 1$ and a proof of B).
  - A proof of $\forall x.A$ is a function
    which for any object t builds a proof of $A[t/x]$.
- It looks like computing and λ-calculus, doesn't it ?

Curry-Howard Isomorphism

- A formula (statement) in the logic is represented as a type in the λ-calculus.
- A proof of a formula A is a term of type A.

logic	λ-calculus
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$	$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}$
$\dfrac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash t : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash (t\ a) : B}$
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash a, b : A \times B}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \textit{fst}\ t : A}$

Curry-Howard (II)

- Dependent types : from $A \rightarrow B$ to $\forall x : A.(B\ x)$
- More Curry-Howard:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ x \notin \Gamma \qquad \frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}$$

$$\frac{\Gamma \vdash \forall x.B}{\Gamma \vdash B[t/x]} \qquad \frac{\Gamma \vdash M : \Pi x : A.B \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B[N/x]}$$

- λ-cube: classification of λ-calculi
- Calculus of Constructions (CC): the most expressive calculus in the λ-cube (polymorphism, dependent types and higher-order)
- Calculus of Inductive Constructions: CC plus Inductive Definitions and Recursion Operators (fixpoint and pattern matching)

Outline

- Some Theoretical Background
  - Constructive Logic
  - Curry-Howard Isomorphism
- The Coq Proof Assistant
  - Specification Language: Inductive Definitions
  - Proof Development
  - Practical Use and Demos

The Coq Proof Assistant

- Main Features
  - Interactive Theorem Proving
  - Powerful Specification Language
    (includes dependent types and inductive definitions)
  - Tactic Language to Build Proofs
  - Type-checking Algorithm to Check Proofs
- More concrete stuff
  - 3 sorts to classify types: Prop,Set,Type
  - Inductive definitions are primitive
  - Elimination mechanisms on such definitions

Examples of Applications of Dependent Types

- Lists and Vectors

$$\text{append} : \forall n : \text{nat}.(\text{list } n) \rightarrow \forall m : \text{nat}.(\text{list } m) \rightarrow (\text{list } n + m)$$

- Integer Square Root

$$\forall n : \text{int}.\ 0 \leq n \rightarrow$$
$$\exists s, r : \text{int}.\ 0 \leq s \ \wedge \ 0 \leq r \ \wedge \ n = s^2 + r \ \wedge \ s^2 \leq n < (s+1)^2$$

- printf (single expression)

$$\text{printf} : \forall t : \mathit{type}.\ t \rightarrow \text{unit}$$

An Inductive Definition

- `Inductive nat : Set := O : nat | S : nat -> nat.`
- A mean to Reason about it

  $\forall P : \mathsf{nat} \rightarrow \mathsf{Prop}, P\;0 \rightarrow (\forall n : \mathsf{nat}, P\;n \rightarrow P\;(\mathsf{S}\;n)) \rightarrow \forall n : \mathsf{nat}, P\;n$

- What about Computing ?

  We need something like Gödel recursion operator in System T:

$$R_a : a \rightarrow (\mathsf{nat} \rightarrow a \rightarrow a) \rightarrow nat \rightarrow a$$

equipped with the following rules:

$$R_a\;v0\;vr\;\mathsf{0} \rightarrow\;\; v0$$

$$R_a\;v0\;vr\;(\mathsf{S}\;p)\;\;\rightarrow\;\; vr\;p\;(R_a\;v0\;vr\;p)$$

This is achieved using Pattern Matching and Structural Recursion.

Logic Connectives as Inductive Definitions (I)


```
Inductive True: Prop := I: True.
Inductive False: Prop :=.

  False_ind : forall P:Prop, False -> P

Inductive and (A : Prop) (B : Prop) : Prop :=
  conj : A -> B -> A /\ B

  and_ind : forall A B P : Prop, (A -> B -> P) -> A /\ B -> P

Inductive or (A : Prop) (B : Prop) : Prop :=
  or_introl : A -> A \/ B | or_intror : B -> A \/ B

  or_ind : forall A B P : Prop, (A -> P) -> (B -> P) -> A \/ B -> P
```

Logic Connectives as Inductive Definitions (II) $\boxed{14}$

- Inductive Constructors $\equiv$ Introduction Rules
- Induction principles (_ind) $\equiv$ Elimination Rules
- Example: how to prove $\forall A, B$: Prop, $A \vee B \rightarrow B \vee A$?
  coming soon...

Proof Development

- Backward Reasoning
- Tactic Based Theorem Proving
- Each tactic application refines the proof term.
- Alternatively one can give a proof term directly.
- Sometimes proofs can be performed automatically.
- Eventually a proof term is produced and type-checked.
- Demo (or_commute.v)

$$\forall A, B : \mathsf{Prop}, A \vee B \rightarrow B \vee A$$

Equality as an Inductive Type

- No equality as a primitive notion in Coq
- Propositional Equality: Leibnitz' equality

```
Inductive eq (A : Type) (x : A) : A -> Prop :=
          refl_equal : x = x
```

eq_ind : $\forall A :$ Type$, x : A, P : A \rightarrow$ Prop$, P\,x \rightarrow \forall y : A, x = y \rightarrow P\,y$

- Terms can also be definitionaly equal ($\beta\delta\iota$-convertible)
- No Extensionality Property (related to extraction matters)

$$\forall f, g : A \rightarrow B, \forall x : A, f\ x = g\ x \rightarrow f = g$$

- Rewriting relies on the substitution principle eq_ind.

Functions Definitions

- Defining (Structural Recursive) Functions
  - Functions have to be total.
  - Definition by Pattern Matching and Guarded Fixpoint
  - Allows to define all primitive recursive functions
    (and more ... e.g. Ackermann)
- Example

```
Fixpoint plus (n m:nat) struct n : nat :=
  match n with | O => m
               | S p => S (plus p m)
  end.
```

- Computational Behaviour (ι-reduction)

plus O $m \xrightarrow{\iota} m$ plus (S p) $m \xrightarrow{\iota}$ (S (plus p m))

Inductive definitions and Induction

- Inductive datatypes e.g. trees (see demo later)
- Inductive predicates

```
Inductive le (n : nat) : nat -> Prop :=
  | le_n : n <= n
  | le_S : forall m : nat, n <= m -> n <= S m
```

le is a parametric inductive type representing a relation.
As an inductive type, it also comes with a induction principle:

$$\forall n : \mathsf{nat}, P : \mathsf{nat} \to \mathsf{Prop},$$
$$P\ n \to (\forall m : \mathsf{nat}, n \leq m \to P\ m \to P\ (\mathsf{S}\ m)) \to$$
$$\forall n0 : nat, n \leq n0 \to P\ n0$$

- Dependent Types

Proofs: some examples

- Inductive Reasoning of bacic types and on a relation (tree.v)
- Induction, Inversion Principles and Case Analysis (coins.v)
- Sometimes induction is not enough: Functional Induction (mod2.v)
- A taste of Dependent Types (dep.v)

Related Tools and Challenges

- Coq has a large standard library including Integers, Reals, Sets.
- Extraction
  - Fully certified programs can be extracted from proofs.
  - from CCInd to $F\omega$
  - Actually from Coq to ML or Haskell
  - Hoare logic and correctness proofs of imperative programs (see http://why.lri.fr)
- Challenges:
  - More Automation (try and formalize the sum example)
  - Friendlier Handling of Dependent Types and Dependently-typed Functions

Further Reading and Exercices

- Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions by Yves Bertot and Pierre Castran
- http://pauillac.inria.fr/coq (Coq Manual, Standard Library)
- Exercices
  - http://www.labri.fr/Perso/˜ casteran/CoqArt/
  - ftp://ftp-sop.inria.fr/lemme/Laurent.Thery/CoqExamples/