Coq au vin

The Coq proof assistant and the Curry-Howard correspondence

Juan Pedro Villa-Isaza
jvillais@eafit.edu.co

Universidad EAFIT

June 8, 2011

Outline

Introduction
Coq au vin

Coq au vin
(Chicken in red wine with onions, mushrooms, and bacon)

- This popular dish may be called *coq au Chambertin*, *coq au riesling*, or *coq au* whatever wine you use for its cooking.
- It is made with either white or red wine, but the red is more characteristic.
- Serve with it a young, full-bodied red Burgundy, Beaujolais, or Côtes du Rhône.

Simone Beck, Louisette Bertholle, and Julia Child.
Mastering the Art of French Cooking.
Alfred A. Knopf, 1966.

Introduction

The Coq proof assistant

Coq
(`http://coq.inria.fr/`)

- The Coq system is a computer tool for verifying theorem proofs.
- Its underlying theory is a logical framework known as the Calculus of Inductive Constructions.
- The Coq language is extremely powerful and expressive, both for reasoning and for programming.

The Curry-Howard (propositions-as-types, formulas-as-types, proofs-as-programs) correspondence (isomorphism)

- The Curry-Howard isomorphism states an amazing correspondence between systems of formal logic and computational calculi.
- It begins with the observation that an implication $A \to B$ corresponds to a type of functions from A to B.
  - A constructive proof of an implication from A to B is a procedure that transforms proofs of A into proofs of B.
  - *An implicational formula is an intuitionistic theorem if and only if it is an inhabited type.*

Introduction

The Curry-Howard correspondence

- Provable theorems are nothing else than non-empty types.
- Virtually all proof-related concepts can be interpreted in terms of computations, and vice versa.
- The Curry-Howard isomorphism is not merely a curiosity, but a fundamental principle.
- "Programs viewed as proofs" and "*proofs viewed as programs*."

Introduction

The Curry-Howard correspondence

logic	**λ-calculus**
formula	type
propositional variable	type variable
connective	type constructor
implication	function space
conjunction	product
disjunction	disjoint sum
absurdity	empty type
proof	term
assumption	object variable
introduction	constructor
elimination	destructor
provability	inhabitation

Introduction

Coq and the Curry-Howard correspondence

- Two approaches can be followed to solve the problem of proving the formula
$$(P \implies Q) \implies ((Q \implies R) \implies (P \implies R)).$$
  1. Building a truth table (classical logic).
  2. Replacing the question "is the proposition P true?" with the question "what are the proofs of P (if any)?" (intuitionistic logic).
     - The Coq system follows this approach.
- If we consider some proof as an expression in a functional language, then the proven statement is a type (the type of proofs for this statement).

Introduction

Coq and the Curry-Howard correspondence

- Thanks to the Curry-Howard correspondence, we can use programming ideas during proof tasks and logical ideas during program design.
- The implication $P \implies Q$ becomes the arrow type $P \to Q$.

$$(P \to Q) \to (Q \to R) \to P \to R$$

- A proof of this statement is a λ-term whose type is this proposition.

```
fun (H1 : P -> Q) (H2 : Q -> R) (p : P) => H2 (H1 p)
```

- Building proofs and programs are very similar activities, but there is one important difference: *proof irrelevance*.

“Averting your face, ignite the cognac with a lighted match.”

Propositional logic

Propositions and proofs

- The coexistence of programs and proofs (and specifications and propositions) is made possible by the `Prop` sort for propositions and proofs.
- *Hypotheses* (local declarations) and *axioms* (global declarations).

$$E, \Gamma \vdash \pi : P$$

- Taking into account the *axioms* in the *environment* E and the *hypotheses* in the *context* Γ, π is a *proof* of the *proposition* P.
- *Theorems* and *lemmas* (global definitions).

Propositional logic

Goals and tactics

- Proof terms can become very complex...
- The Coq system provides a suite of tools to help in their construction.
- Working model:
  1. The user states the proposition that needs to be proved (*goal*).
  2. The user applies commands (*tactics*) to decompose the goal into simpler goals or solve it. This process ends when all subgoals are completely solved.

Propositional logic

Tactics

- The tactic `apply` *term* tries to match the current goal against the conclusion of the type of *term*...
- The tactic `absurd` *term* applies `False` elimination...
- The tactic `constructor` *num* applies to a goal such that the head of its conclusion is an inductive constant.
- The tactic `elim` chooses the appropiate destructor and applies it as the tactic `apply` would do.
- ...

Propositional logic

Example

$$(P \to Q) \to (Q \to R) \to P \to R$$

- Declaring propositional variables (`Hypothesis` and `Hypotheses`).

```
Coq < Section Propositional_logic.

Coq < Hypothesis P : Prop.
P is assumed
Coq < Hypothesis Q : Prop.
Q is assumed

Coq < Hypotheses R S : Prop.
R is assumed
S is assumed
```

Propositional logic

Example

- Activating goal-directed proofs (`Theorem`).

```
Coq < Theorem imp_trans : (P -> Q) -> (Q -> R) -> P -> R.
1 subgoal

  P : Prop
  Q : Prop
  R : Prop
  S : Prop
  ============================
   (P -> Q) -> (Q -> R) -> P -> R
```

Propositional logic

Example

- Introducing new hypotheses (`intro` and `intros`).

```
imp_trans < intro H1.
1 subgoal

  P : Prop
  Q : Prop
  R : Prop
  S : Prop
  H1 : P -> Q
  ============================
   (Q -> R) -> P -> R
```

```
imp_trans < intros H2 p.
1 subgoal

  P : Prop
  Q : Prop
  R : Prop
  S : Prop
  H1 : P -> Q
  H2 : Q -> R
  p : P
  ============================
   R
```

Propositional logic

Example

- Applying a hypothesis (`apply`).

```
imp_trans < apply H2.
1 subgoal

  P : Prop
  Q : Prop
  R : Prop
  S : Prop
  H1 : P -> Q
  H2 : Q -> R
  p : P
  ============================
   Q
```

```
imp_trans < apply H1.
1 subgoal

  P : Prop
  Q : Prop
  R : Prop
  S : Prop
  H1 : P -> Q
  H2 : Q -> R
  p : P
  ============================
   P
```

Propositional logic
Example

- Using a hypothesis (`assumption`).

```
imp_trans < assumption.
Proof completed.
```

- Building and solving the proof term (`Qed`).

```
imp_trans < Qed.
intro H1.
intros H2 p.
apply H2.
apply H1.
assumption.

imp_trans is defined
```

Propositional logic

Example

- Printing the proof term (`Print`).

```
Coq < Print imp_trans.
imp_trans =
fun (H1 : P -> Q) (H2 : Q -> R) (p : P) => H2 (H1 p)
     : (P -> Q) -> (Q -> R) -> P -> R
```

- Agda:

```
imp_trans : {P Q R : Set} -> (P -> Q) -> (Q -> R) -> P -> R
imp_trans H1 H2 p = H2 (H1 p)
```

Propositional logic

Example

- A one-shot tactic (`auto`).

```
Theorem imp_trans : (P -> Q) -> (Q -> R) -> P -> R.
Proof.
  auto.
Qed.
```

“Decorate with sprigs of parsley.”

Propositional logic

Implication (function space)

Introduction rule:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \; (\to\mathrm{I})$$

Elimination rule:

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \; (\to\mathrm{E})$$

Propositional logic

Conjunction (product)

Introduction rule:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,(\wedge\mathrm{I})$$

Elimination rules:

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,(\wedge\mathrm{E}_1)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,(\wedge\mathrm{E}_2)$$

Propositional logic

Product (conjunction)

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

proj1 =
fun (A B : Prop) (H : A /\ B) => match H with
                                 | conj H0 _ => H0
                                 end
     : forall A B : Prop, A /\ B -> A

proj2 =
fun (A B : Prop) (H : A /\ B) => match H with
                                 | conj _ H1 => H1
                                 end
     : forall A B : Prop, A /\ B -> B
```

Propositional logic

Disjunction (disjoint sum)

Introduction rules:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vee \mathrm{I}_1)$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vee \mathrm{I}_2)$$

Elimination rule:

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ (\vee \mathrm{E})$$

Propositional logic

Disjoint sum (disjunction)

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

or_ind =
fun (A B P : Prop) (f : A -> P) (f0 : B -> P) (o : A \/ B) =>
match o with
| or_introl x => f x
| or_intror x => f0 x
end
     : forall A B P : Prop, (A -> P) -> (B -> P) -> A \/ B -> P
```

Propositional logic

Absurdity (empty type)

Elimination rule:

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\,(\bot\mathrm{E})$$

Propositional logic

Empty type (absurdity)

```
Inductive False : Prop :=

False_ind = fun P : Prop => False_rect P
     : forall P : Prop, False -> P
```

Propositional logic

True

```
Inductive True : Prop :=  I : True
```

Propositional logic

Negation

```
not = fun A : Prop => A -> False
     : Prop -> Prop
```

Propositional logic

Bi-implication

```
iff = fun A B : Prop => (A -> B) /\ (B -> A)
    : Prop -> Prop -> Prop
```

Propositional logic

Example (1)

```
Theorem example1 : forall A : Prop, A -> ~ ~ A.
Proof.
  unfold not.
  intros A a H.
  apply H.
  assumption.
Qed.

example1 =
fun (A : Prop) (a : A) (H : A -> False) => H a
     : forall A : Prop, A -> ~ ~ A
```

Propositional logic

Example (2)

```
Lemma example2_1 : forall A B C : Prop,
                   A /\ (B \/ C) -> A /\ B \/ A /\ C.
Proof.
  intros A B C H1.
  elim H1.
  intros a H2.
  elim H2.
    intro b.
    constructor 1.
    constructor.
  assumption.
  assumption.
 auto.
Qed.

Lemma example2_2 : forall A B C : Prop,
                   A /\ B \/ A /\ C -> A /\ (B \/ C).
```

Propositional logic

Example (2)

```
example2_1 =
fun (A B C : Prop) (H1 : A /\ (B \/ C)) =>
and_ind
  (fun (a : A) (H2 : B \/ C) =>
   or_ind (fun b : B => or_introl (A /\ C) (conj a b))
     (fun H : C => or_intror (A /\ B) (conj a H)) H2) H1
     : forall A B C : Prop, A /\ (B \/ C) -> A /\ B \/ A /\ C
```

Propositional logic
Example (2)

```
Theorem example2 : forall A B C : Prop,
                   A /\ (B \/ C) <-> A /\ B \/ A /\ C.
Proof.
  intros A B C.
  unfold iff.
  constructor.
    exact (example2_1 A B C).
    apply example2_2.
Qed.

example2 =
fun A B C : Prop => conj (example2_1 A B C) (example2_2 A B C)
     : forall A B C : Prop, A /\ (B \/ C) <-> A /\ B \/ A /\ C
```

Predicate logic

Universal quantification

```
all =
fun (A : Type) (P : A -> Prop) => forall x : A, P x
     : forall A : Type, (A -> Prop) -> Prop
```

Predicate logic

Existential quantification

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
    ex_intro : forall x : A, P x -> ex P
```

Proof irrelevance

- When developing a program for a specification, two programs may not be considered completely equivalent.
- When considering proofs, two proofs of a proposition play exactly the same role.
- "Proof irrelevance" asserts equality of all proofs of a given formula.

```
Axiom proof_irrelevance : forall (P:Prop) (p1 p2:P), p1 = p2.
```

"The end justifies the means"?

Proof irrelevance

```
Lemma lemma7_1 : forall P Q : Prop, (P -> Q) -> (P -> P -> Q).
Proof.
  intros P Q H _.
  assumption.
Qed.

Lemma lemma7_2 : forall P Q : Prop, (P -> Q) -> (P -> P -> Q).
Proof.
  intros P Q H p1 p2.
  apply H.
  assumption.
Qed.

Lemma lemma7_12 : lemma7_1 = lemma7_2.
Proof.
  apply proof_irrelevance.
Qed.
```

Summary

- The Coq proof assistant.
- The Curry-Howard correspondence.
- Coq and the Curry-Howard correspondence.
- The computational aspects of logical systems (proofs viewed as programs).
- Proof irrelevance.

The firebird

(Coq'Art)

References

The Coq proof assistant

The Coq Development Team.
The Coq Proof Assistant.
http://coq.inria.fr/.

Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring.
The Coq Proof Assistant. A Tutorial.

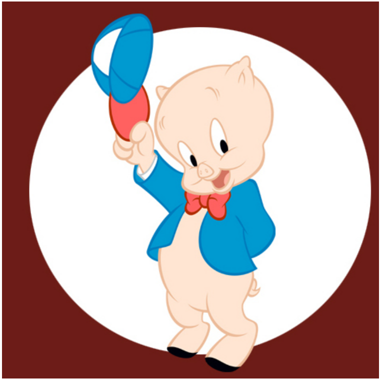
Eduardo Giménez and Pierre Castéran.
A Tutorial on [Co-]Inductive Types in Coq.

Yves Bertot and Pierre Castéran.
Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.
Springer, 2004.

References

The Curry-Howard correspondence

Morten Heine Sørensen and Paweł Urzyczyn.
Lectures on the Curry-Howard Isomorphism.
Elsevier, 2006.



(Warner Bros.)

“Th-th-th-that’s all, folks!”