形式化方法导引

第 6 章 案例分析

6.3 – Coq: A Prover based on Higher-order Logic

6.3.1 Introduction & 6.3.2 Basics

黄文超
https://faculty.ustc.edu.cn/huangwenchao

代码链接

1. Introduction

What is Coq proof assistant?

- an environment for developing mathematical facts
  - defining objects (integers, sets, trees, functions, programs ...)
  - making statements (using basic predicates and logical connectives)
  - writing proofs

Main functions of Coq?

- compiler
  - automatically checks the correctness of definitions
  - automatically checks the correctness of proofs
- environment
  - advanced notations; proof search; modular developments
  - program extraction towards languages like Ocaml and Haskell

1. Introduction

Impressive examples in different areas?

- pure mathematics
  - the Fundamental theorem of Algebra
    - every polynomial has a root in complex field
  - Feit-Thompson theorem on finite groups
  - the four-color theorem …
- formalizing programming environments
  - JavaCard platform ~ the Gemalto and Trusted Logic companies
    - the highest level of certification (common criteria EAL 7)
  - CompCert: a certified optimizing compiler for C
  - Certicrypt: an environment of formal proofs for computational cryptography
  - Ynot library: for proving imperative programs using separation logic

1. Introduction

Related systems?

- similar to HOL systems
  - a family of interactive theorem provers based on Church's higher-order logic
    - including Isabelle/HOL, HOL4, HOL-light, PVS
- Difference from HOL systems
  - Coq is based on an intuitionistic type theory
    - functions are programs that can be computed

Websites or Books?

- Official website: https://coq.inria.fr
- Coq'art book
- Software Foundations
- Coq in a Hurry

1. Introduction

Coq architecture （v8.14.1,v8.15.1）

- Two-levels architecture
  - small kernel based on a language with few primitive constructions
    - functions, (co)-inductive definitions, product types, sorts
    - a limited number of rules for type-checking and computation
  - rich environment
    - to help designing theories and proofs offering mechanisms
      - like user extensible notations, tactics for proof automation, libraries
    - can be used and extended safely
      - ultimately any definition and proof is checked by a safe kernel

```
5=2+3
```

```
@eq Z (Zpos (xI (xO xH))) (Zplus (Zpos (xO xH)) (Zpos (xI xH)))
```

1. Introduction

Program verification in Coq

- One can express the property “the program p is correct” as a mathematical statement, and prove it is correct
- One can develop a specific program analyzer (model-checking, abstract interpretation,. . .) in Coq, prove it correct and use it
- One can
    - represent the program p by a Coq term t
    - represent the specification by a type T
    - such that t : T (which is automatically checked) implies p is correct
    - It works well for functional (possibly monadic) programs
- One can use an external tool to generate proof obligations and then use Coq to solve obligations.

2. Basics - 2.1 Basic Terms

A Coq object in the environment has a *name* and a *type*

Command： **Check** *term*

- takes a *term* as an argument
- checks it is well-formed
- displays its *type*

```
Check nat.
```

```
nat
      : Set
```

The object nat is a predefined type for natural numbers

- its type is a special constant Set called a sort.

2. Basics - 2.1 Basic Terms

A Coq object in the environment has a *name* and a *type*

```
Check 0.
```

```
0
     : nat
```

The constant 0 has type nat

```
Check S.
```

```
S
     : nat -> nat
```

The object S is the successor function

- it has type nat → nat

```
Check plus.
```

```
Nat.add
     : nat -> nat -> nat
```

The binary function plus has type nat → nat → nat

- which should be read as nat → nat → nat

2. Basics - 2.1 Basic Terms

A function f can be applied to a term t using the notation $f\ t$.

- The term $f\ t_1\ t_2$ stands for $(f\ t_1)\ t_2$
- The natural number 10
  - a notation for the successor function applied 10 times to 0
- The usual infix notation $t_1 + t_2$ can be used instead of plus $t_1\ t_2$

```
Check (3+2).
```

```
3 + 2
     : nat
```

2. Basics - 2.1 Basic Terms

In Coq, logical propositions are also seen as terms.

- The type of propositions is the sort Prop

Paper notations ~ Coq input

$\bot$	$\top$	$t = u$	$t \neq u$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \to Q$	$P \Leftrightarrow Q$
`False`	`True`	`t=u`	`t<>u`	`~P`	`P /\ Q`	`P \/ Q`	`P -> Q`	`P <-> Q`

$\forall x, P$	`forall x, P`	`forall x:T, P`	`forall T (x y:T) (z:nat),P`
$\exists x, P$	`exists x, P`	`exists x:T, P`	*no multiple bindings*

2. Basics - 2.1 Basic Terms

The command **Check** verifies a proposition is well-formed

- but does not say if it is true or not

```
Check (1+2=3).
```

```
1 + 2 = 3
     : Prop
```

```
Check (forall x:nat, exists y, x=y+y).
```

```
forall x : nat, exists y : nat, x = y + y
     : Prop
```

2. Basics - 2.2 Logical rules and tactics

问: How to produce a proof to establish a proposition is true?

Backward reasoning with **tactics**

- A tactic transforms a goal into a set of subgoals
  - solving these subgoals is sufficient to solve the original goal

问: How to introduce a new goal?

Command: the following commands with prop (prop : Prop)

- **Lemma** id : prop
- **Theorem** id : prop
- **Goal** prop.

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
```

2. Basics - 2.2 Logical rules and tactics

问: The format of a proof?

Curry-Howard isomorphism (柯里-霍华德同构)

- A proof of a proposition A is represented by a term of type A

问: Result of Curry-Howard isomorphism?

There is only one form of judgment $\Gamma \vdash p : A$

- The environment Γ is a list of names associated with types $x : T$
- When A is a type of objects (e.g., $x : \text{nat} \vdash x : \text{nat}$)
  - p is well-formed in the environment Γ and has type A
- When A is a proposition (e.g., $x : \text{nat},\ h : x = 1 \vdash \ldots : x \neq 0$)
  - A is provable under the assumption of Γ and p is a witness of that proof

2. Basics - 2.2 Logical rules and tactics

问: Rules for a proof?

- axiom rule
- introduction rules
- elimination rules

Axiom rule

- The goal to be proven is directly a hypothesis
- The logical rule and tactics:

$\dfrac{h : A \in \Gamma}{\Gamma \vdash h : A}$	$\texttt{exact } h \text{ or } \texttt{assumption}$

2. Basics - 2.2 Logical rules and tactics

Introduction rules for some connectives (e.g., $\to$, $\vee$, $\wedge$ . . .)

- give a mean to prove a proposition formed with that connective
  - if we can prove simpler propositions

e.g.

$\to$	$\dfrac{\Gamma, h : A \vdash ? : B}{\Gamma \vdash ? : A \to B}$	`intro` h

回顾第3章:

$$\dfrac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \to \psi} \to\mathrm{i}$$

- A tactic will work with a still unresolved goal
  - that we indicate using ? in place of the proof-term

2. Basics - 2.2 Logical rules and tactics

Elimination rules for some connectives (e.g., $\rightarrow$, $\vee$, $\wedge$ . . .)

- explains how we can use a proof of a proposition with that connective

e.g.

$\dfrac{\Gamma \vdash h : A \rightarrow B \qquad \Gamma \vdash ? : A}{\Gamma \vdash ? : B}$	`apply` h

回顾第3章

$$\frac{\phi \quad \phi \longrightarrow \psi}{\psi} \rightarrow e$$

例: Lemma ex1: $\forall A\ B\ C : Prop, (A \to B \to C) \to (A \to B) \to A \to C$

$\dfrac{h : A \in \Gamma}{\Gamma \vdash h : A}$	exact h or assumption	$\to$	$\dfrac{\Gamma, h : A \vdash ? : B}{\Gamma \vdash ? : A \to B}$	intro h

$\dfrac{\Gamma \vdash h : A \to B \qquad \Gamma \vdash ? : A}{\Gamma \vdash ? : B}$	apply h

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intro h1.
intro h2.
intro h3.
intro h4.
intro h5.
intro h6.
apply h4.
assumption.
apply h5.
assumption.
Qed.
```

2. Basics - 2.2 Logical rules and tactics

	introduction		elimination	
$\bot$			$\dfrac{\Gamma\vdash ?:\mathtt{False}}{\Gamma\vdash ?:C}$	`exfalso`
$\neg$	$\dfrac{\Gamma,h:A\vdash ?:\mathtt{False}}{\Gamma\vdash ?:\neg A}$	`intro` h	$\dfrac{\Gamma\vdash h:\neg A \qquad \Gamma\vdash ?:A}{\Gamma\vdash ?:C}$	`destruct` h
$\to$	$\dfrac{\Gamma,h:A\vdash ?:B}{\Gamma\vdash ?:A\to B}$	`intro` h	$\dfrac{\Gamma\vdash h:A\to B \qquad \Gamma\vdash ?:A}{\Gamma\vdash ?:B}$	`apply` h
$\forall$	$\dfrac{\Gamma,y:A\vdash ?:B[x\leftarrow y]}{\Gamma\vdash ?:\forall x:A,B}$	`intro` y	$\dfrac{\Gamma\vdash h:\forall x:A,B \qquad \Gamma\vdash t:A}{\Gamma\vdash ?:B[x\leftarrow t]}$	`apply` h `with` $(x$ `:=` $t)$
$\wedge$	$\dfrac{\Gamma\vdash ?:A \quad \Gamma\vdash ?:B}{\Gamma\vdash ?:A\wedge B}$	`split`	$\dfrac{\Gamma\vdash h:A\wedge B \quad \Gamma,l:A,m:B\vdash ?:C}{\Gamma\vdash ?:C}$	`destruct` h `as` (l,m)
$\vee$	$\dfrac{\Gamma\vdash ?:A}{\Gamma\vdash ?:A\vee B}$ $\dfrac{\Gamma\vdash ?:B}{\Gamma\vdash ?:A\vee B}$	`left` `right`	$\dfrac{\Gamma\vdash h:A\vee B \quad \Gamma,l:A\vdash ?:C \quad \Gamma,l:B\vdash ?:C}{\Gamma\vdash ?:C}$	`destruct` h `as` $[l\mid l]$
$\exists$	$\dfrac{\Gamma\vdash t:A \quad \Gamma\vdash ?:B[x\leftarrow t]}{\Gamma\vdash ?:\exists x:A,B}$	`exists` t	$\dfrac{\Gamma\vdash h:\exists x:A,B \quad \Gamma,x:A,l:B\vdash ?:C}{\Gamma\vdash ?:C}$	`destruct` h `as` (x,l)

2. Basics - 2.2 Logical rules and tactics

	introduction		elimination	
$=$	$\dfrac{t \equiv u}{\Gamma \vdash ? : t = u}$	`reflexivity`	$\dfrac{\Gamma \vdash h : t = u \qquad \Gamma \vdash ? : C[x \leftarrow u]}{\Gamma \vdash ? : C[x \leftarrow t]}$	`rewrite` h

- Two terms t and u are convertible (written $t \equiv u$) when they represent the same value after computation.
- The elimination rule allows to replace a term by an equal in any context.

$\dfrac{\Gamma \vdash ? : u = t}{\Gamma \vdash ? : t = u}$	`symmetry`
$\dfrac{\Gamma \vdash ? : t = v \qquad \Gamma \vdash ? : v = u}{\Gamma \vdash ? : t = u}$	`transitivity` v
$\dfrac{\Gamma \vdash ? : f = g \qquad \Gamma \vdash ? : t_1 = u_1 \ldots \Gamma \vdash ? : t_n = u_n}{\Gamma \vdash ? : f\ t_1 \ldots t_n = g\ u_1 \ldots u_n}$	`f_equal`

2. Basics - 2.2 Logical rules and tactics

回顾: How to produce a proof to establish a proposition is true?

Backward reasoning with **tactics**

- A tactic transforms a goal into a set of subgoals
  - solving these subgoals is sufficient to solve the original goal

It is often useful to do **forward** reasoning

- by adding new facts in the goal to be proven

$\dfrac{\Gamma \vdash ? : B \qquad \Gamma, h : B \vdash ? : A}{\Gamma \vdash ? : A}$	`assert` $(h : B)$

2. Basics - 2.2 Logical rules and tactics

It would be painful to apply only atomic rules

- Tactics usually combine in one step several introductions or elimination rules.
  - The tactic **intros** does multiple introductions and infer names when none are given.
  - The tactic **apply** takes as an argument a proof h of a proposition

$$\forall x_1 \ldots x_n, A_1 \to \ldots A_p \to B$$

    - It tries to find terms t_i such that the current goal is equivalent to $B[x_i \leftarrow t_i]_{i=1\ldots n}$
    - and generates subgoals corresponding to $A_j[x_i \leftarrow t_i]_{i=1\ldots n}$

2. Basics - 2.2 Logical rules and tactics

It would be painful to apply only atomic rules

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intros.
apply H.
assumption.
apply H0.
assumption.
Qed.
```

2. Basics - 2.2 Logical rules and tactics

It would be painful to apply only atomic rules

Some tactics are doing proof search to help solve a goal:

contradiction	solves the goal when False, or A and ¬A appear in the hypotheses
tauto	solves propositional tautologies
trivial	tries very simple lemmas to solve the goal
auto	searches in a database of lemmas to solve the goal
intuition	removes the propositional structure of the goal then auto
omega	solves goals in linear arithmetic

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
auto.
Qed.
```

2. Basics - 2.2 Logical rules and tactics

Finishing proofs

- Commands: **Theorem** and **Lemma**
  - given a name name and a property A
  - enter the interactive proof mode
    - in which tactics are used to transform the goal
  - after some effort there will be no remaining subgoals
    - the proof of A is finished
- Actually, Coq is doing one more check before accepting the proof
  - extracts a term p and the trusted kernel has to check that
    - $\Gamma \vdash p : A$ is a valid judgment
    - by elementary rules for type-checking p
  - Commands: **Qed** and **Save**

2. Basics - 2.2 Logical rules and tactics

If a proof is not finished:

- Using the command **Admitted** instead of Qed
  - gives the possibility to finish the proof
  - introducing the original goal as an axiom.

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
Admited.
```

- It is convenient to postpone a proof but it is also potentially dangerous
- Safety in Coq is only guaranteed if there are no axioms left in the proof.
  - The command **Print Assumptions** name can be used to display all axioms used in the theorem name.

```
Print Assumptions ex1.
```

2. Basics - 2.2 Logical rules and tactics

Command: **Definitions**

- A new definition is introduced by:

$$\textbf{Definition}\ \mathit{name}\ \mathit{args} : \ \mathit{type}\ := \ \mathit{term}.$$

- The identifier *name*: an abbreviation for the term *term*.
- The type *type*: optional as well as the arguments
- The arguments: a list of identifiers possibly associated with types

```
Definition square (x:nat) : nat := x * x.
```

- A Coq definition name can be unfolded in a goal by using the tactic：
  - **unfold** name (in the conclusion)
  - **unfold** name in H (in hypothesis H).

2. Basics - 2.2 Logical rules and tactics

Command: **Section** *name*

- It is often convenient to introduce a local context of variables and properties, which are shared between several definitions.
- Then objects can be introduced using the syntax

$$\textbf{Variable}\ \mathit{name} : \mathit{type} \qquad \text{or} \qquad \textbf{Hypothesis}\ \mathit{name} : \mathit{prop}$$

- command **Variables**： introduce variables with the same type
- command **End**： end the section

```
Section test.
Variable A : Type.
Variables x y : A.
Definition double : A * A := (x,x).
Definition triple : A * A * A := (x,y,x).
End test.
```

2. Basics - 2.2 Logical rules and tactics

Command: **Section** *name*

```
Section test.
Variable A : Type.
Variables x y : A.
Definition double : A * A := (x,x).
Definition triple : A * A * A := (x,y,x).
End test.
```

- After ending the section, the objects A, x and y are not accessible anymore
  - one can observe the new types of double and triple.

```
Print double.
```

```
double =
fun (A : Type) (x : A) => (x, x)
     : forall A : Type, A -> A * A
```

2. Basics - 2.3 Libraries in Coq

The Coq environment is organized in a modular way

- Some libraries are already loaded when starting the system

```
Print Libraries.
```

```
Loaded library files:
  Coq.Init.Notations
  Coq.Init.Ltac
  Coq.Init.Logic
  Coq.Init.Datatypes
  Coq.Init.Logic_Type
  Coq.Init.Specif
  Coq.Init.Decimal
  Coq.Init.Hexadecimal
  Coq.Init.Number
  Coq.Init.Nat
  Coq.Init.Byte
  Coq.Init.Numeral
  …
```

2. Basics - 2.3 Libraries in Coq

Searching the environment

- **Search** *name*: display all declarations id : type in the environment such that *name* appears in type.

```
Search plus.
```

- **Search** [$name_1 \cdot \cdot \cdot name_n$]: find objects with types mentioning all the names $name_i$

```
Search [plus 0].
```

- **Search** *pattern*: find objects with types mentioning an instance of the pattern
    - which is a term possibly using the special symbol "_" to represent an arbitrary term.

```
Search (~_ <->_).
```

2. Basics - 2.3 Libraries in Coq

Other Commands w.r.t libraries

- **Check** *term*: checks if term can be typed and displays its type.
- **Print** *name*: prints the definition of name together with its type.
- **About** *id*: displays the type of the object id
  - (plus other informations like qualified name or implicit arguments).

```
About pair.
```

```
pair : forall {A B : Type}, A -> B -> A * B

pair is template universe polymorphic
on prod.u0 prod.u1
Arguments pair {A B}%type_scope _ _
Expands to: Constructor Coq.Init.Datatypes.pair
```

https://faculty.ustc.edu.cn/huangwenchao/zh_CN/zdylm/680196/list/index.htm

2. Basics - 2.3 Libraries in Coq

Load New Libraries

- Command: **Require Import** *name*
  - checks if module name is already present in the environment
    - If not, and if a file *name.vo* occurs in the load-path
      - then it is loaded and opened (its contents is revealed)

```
Require Import Arith.
```

- Command: **Print Libraries**
  - display the set of loaded modules
- Command: **Print LoadPath**
  - display the load-path

2. Basics - 2.4 Examples

- Define an absolute value function on **mathematical** integers
- prove the result is positive

- Mathematical integers in Coq are defined as a type Z
  - Their representation is based on a binary representation of positive numbers (type positive)

```
Require Import ZArith.
Open Scope Z_scope.
```

- Find a function: Z.leb

```
Search (Z->Z->bool).
```

```
Z.leb: Z -> Z -> bool
```

- Search properties for Z.leb

```
Search Z.leb.
```

```
Zle_cases: forall n m : Z, if n <=? m then n <= m else n > m
```

2. Basics - 2.4 Examples

- Define an absolute value function on **mathematical** integers
- prove the result is positive

- Ready to Go

```
Require Import ZArith.
Open Scope Z_scope.
Definition abs (n:Z) : Z := if Z.leb 0 n then n else -n.
Lemma abs_pos : forall n, 0 <= abs n.
intro n.
unfold abs.
assert (if Z.leb 0 n then 0 <= n else 0 > n).
apply Zle_cases.
destruct (Z.leb 0 n);auto with zarith.
Qed.
```

2. Basics - 2.4 Examples

- Define an absolute value function on **mathematical** integers
- prove the result is positive

- Ready to Go (without proof search)

```
Require Import ZArith.
Open Scope Z_scope.
Definition abs (n:Z) : Z := if Z.leb 0 n then n else -n.
Lemma abs_pos : forall n, 0 <= abs n.
intro n.
unfold abs.
assert (if Z.leb 0 n then 0 <= n else 0 > n).
apply Zle_cases.
destruct (Z.leb 0 n).
apply H.
Search [Z.lt Z.gt]. assert (n<0). apply Z.gt_lt. assumption.
assert (n<=0).
Search [Z.lt Z.le]. apply Z.lt_le_incl. assumption.
Search [Z.le Z.opp]. apply Z.opp_nonneg_nonpos. assumption.
Qed.
```

2. Basics - 2.5 Intuitionistic Logic v.s Classical Logic

$A \vee \neg A$ is **not** an axiom in intuitionistic logic

- Coq implements an intuitionistic logic
- Actually, both $A \vee B$ and $\exists x : A, B$ have a strong constructive meaning.
  - from a proof of $\exists x : A, B$
    - one can compute t such that $B[x \leftarrow t]$ is provable
  - from a proof of $A \vee B$
    - one can compute a boolean b and proofs of $b = \text{true} \rightarrow A$ and $b = \text{false} \rightarrow B$
- It is also possible to use classical versions of logical connectives
  - negative formulas are classical, e.g., $\neg\neg A \rightarrow A$
  - a library Classical introduces the excluded middle as an axiom

```
Require Import Classical.
```

作业

- 实验小作业，使用Coq证明如下命题（不允许使用搜索策略，不允许使用Classical库），附上代码和文档（文档中列出每个证明步骤的输出截图）

```
Lemma ex1: forall A, ~~~ A -> ~ A.
```

```
Lemma ex2: forall A B, A \/ B -> ~ (~ A /\ ~ B).
```

```
Lemma ex3: forall T (P:T -> Prop),
(~ exists x, P x) -> forall x, ~ P x.
```