Comparison of two theorem provers: Isabelle & Coq

A. Yushkovskiy   S. Tripakis

Department of Computer Science
School of Science
Aalto University

CS-E4000: Seminar in Computer Science
autumn 2017

Introduction

Historical note

PRINCIPLES OF

MATHEMATICAL

LOGIC

BY

D. HILBERT AND W. ACKERMANN

TRANSLATED FROM THE GERMAN BY

LEWIS M. HAMMOND · GEORGE G. LECKIE · F. STEINHARDT

Professor of Philosophy, University of Virginia · Professor of Philosophy, Emory University · Columbia University

EDITED AND WITH NOTES BY

ROBERT E. LUCE

Assistant Professor of Mathematics

Rutgers University

PUBLISHED AND DISTRIBUTED IN THE PUBLIC INTEREST BY AUTHORITY OF THE ATTORNEY GENERAL UNDER LICENSE NUMBER A-1428

Über die Bedeutung des Satzes vom ausgeschlossenen Dritten in der Mathematik, insbesondere in der Funktionentheorie [1]).

Von *L. E. J. Brouwer* in Amsterdam.

§ 1.

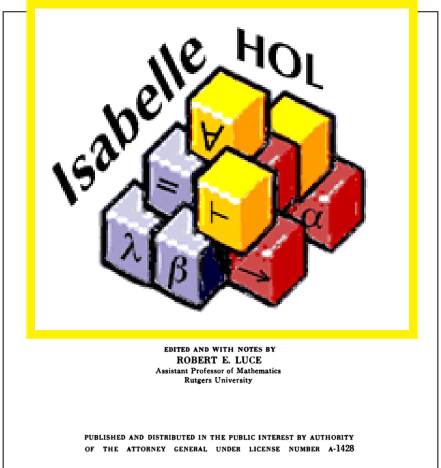
Innerhalb eines bestimmten endlichen „Hauptsystems" können Eigenschaften von Systemen, d. h. Abbildbarkeiten von Systemen auf andere Systeme mit vorgeschriebenen Elementkorrespondenzen, immer *geprüft* (d. h. entweder bewiesen oder ad absurdum geführt) werden; die durch die betreffende Eigenschaft angewiesene Abbildung besitzt nämlich auf jeden Fall nur eine endliche Anzahl von Ausführungsmöglichkeiten, von denen jede für sich unternommen und entweder bis zur Beendigung oder bis zur Hemmung fortgesetzt werden kann. (Hierbei liefert das Prinzip der mathematischen Induktion oft das Mittel, derartige Prüfungen ohne individuelle Betrachtung jedes an der Abbildung beteiligten Elementes bzw. jeder für die Abbildung bestehenden Ausführungsmöglichkeit durchzuführen; demzufolge kann die Prüfung auch für Systeme mit sehr großer Elementenzahl mitunter verhältnismäßig schnell verlaufen.)

Auf Grund der obigen Prüfbarkeit gilt für innerhalb eines bestimmten endlichen Hauptsystems konzipierte Eigenschaften der *Satz vom ausgeschlossenen Dritten*, d. h. das Prinzip, daß jede Eigenschaft für jedes System entweder richtig oder unmöglich ist, und insbesondere der *Satz von der Reziprozität der Komplementärspezies*, d. h. das Prinzip, daß für jedes System aus der Unmöglichkeit der Unmöglichkeit einer Eigenschaft die Richtigkeit dieser Eigenschaft folgt.

Wenn z. B. die Vereinigung $\mathfrak{S}(p, q)$ zweier mathematischer Spezies p und q wenigstens 11 Elemente enthält, so folgt hieraus auf Grund des (in diesem Falle

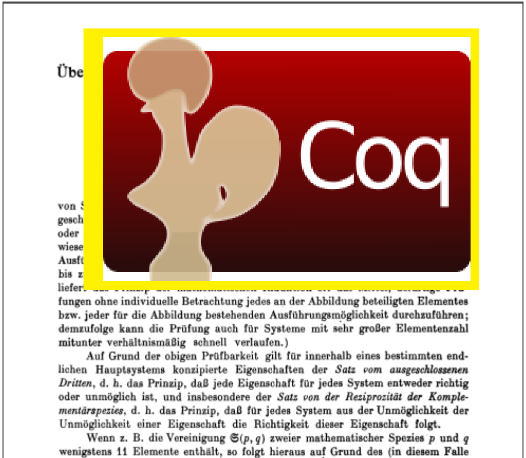
Introduction

Historical note



EDITED AND WITH NOTES BY
ROBERT E. LUCE
Assistant Professor of Mathematics
Rutgers University

PUBLISHED AND DISTRIBUTED IN THE PUBLIC INTEREST BY AUTHORITY OF THE ATTORNEY GENERAL UNDER LICENSE NUMBER A-1428



Übe

von
gesch
oder
wiese
Ausf
bis z
liefer
fungen ohne individuelle Betrachtung jedes an der Abbildung beteiligten Elementes bzw. jeder für die Abbildung bestehenden Ausführungsmöglichkeit durchzuführen; demzufolge kann die Prüfung auch für Systeme mit sehr großer Elementenzahl mitunter verhältnismäßig schnell verlaufen.)

Auf Grund der obigen Prüfbarkeit gilt für innerhalb eines bestimmten endlichen Hauptsystems konzipierte Eigenschaften der *Satz vom ausgeschlossenen Dritten*, d. h. das Prinzip, daß jede Eigenschaft für jedes System entweder richtig oder unmöglich ist, und insbesondere der *Satz von der Reziprozität der Komplementärspezies*, d. h. das Prinzip, daß für jedes System aus der Unmöglichkeit der Unmöglichkeit einer Eigenschaft die Richtigkeit dieser Eigenschaft folgt.

Wenn z. B. die Vereinigung $\mathfrak{S}(p, q)$ zweier mathematischer Spezies p und q wenigstens 11 Elemente enthält, so folgt hieraus auf Grund des (in diesem Falle

Outline

Foundations of Formal Approach
- A formal system
- Classical and Intuitionistic logics

Isabelle & Coq
- First acquaintance
- Basic syntax
- Proof examples

Outline

Foundations of Formal Approach
- A formal system
- Classical and Intuitionistic logics

Isabelle & Coq
- First acquaintance
- Basic syntax
- Proof examples

Elements of a formal system

- A formula (judgement, statement) $\phi \in \Phi$:

$$\begin{aligned} \phi := {} & p \mid q \mid \ldots \\ & \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \rightarrow \phi_2 \\ & \mid \mathit{true} \mid \mathit{false} \\ & \mid \ldots \end{aligned}$$

Elements of a formal system

- A formula (judgement, statement) $\phi \in \Phi$:

$$\begin{aligned}\phi := {} & p \mid q \mid \ldots \\ & \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \neg\phi_1 \mid \phi_1 \to \phi_2 \\ & \mid \mathit{true} \mid \mathit{false} \\ & \mid \ldots\end{aligned}$$

- Propositional variables: – $p, q, \ldots \in V$
- An axiom – $\phi_A \in A$
- An inference rule τ – a transition function $\tau : \Phi \to \Phi$
- A formula ϕ is provable from Φ – $\Phi \vdash \phi$
- A tautology $\top$ – $\vdash \phi$
- A contradiction $\bot$ – $\vdash \neg\phi$

Definition of the formal system

The *formal system* is a quadruple $\Gamma = < A, V, \Omega, R >$, where

- A – set of axioms
- V – set of propositional variables
- Ω – set of logical operators
- R – set of inference rules

Definition of the formal system

The *formal system* is a quadruple $\Gamma = < A, V, \Omega, R >$, where

- A – set of axioms
- V – set of propositional variables
- Ω – set of logical operators
- R – set of inference rules

The *proof* of the formula ϕ is a finite sequence of judgements $\psi_1 \xrightarrow{\tau_1} \psi_2 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_{n-1}} \psi_n$, where $\psi_i \in A \cup \{\psi_k\}_{k=1}^{i-1}$

Definition of the formal system

The *formal system* is a quadruple $\Gamma = < A, V, \Omega, R >$, where

- A – set of axioms
- V – set of propositional variables
- Ω – set of logical operators
- R – set of inference rules

The *proof* of the formula ϕ is a finite sequence of judgements $\psi_1 \xrightarrow{\tau_1} \psi_2 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_{n-1}} \psi_n$, where $\psi_i \in A \cup \{\psi_k\}_{k=1}^{i-1}$

Styles of proof writing:

- forward proof: $premises \xrightarrow{\tau} goals$
- backward proof: $goals \xleftarrow{\tau} premises$

Classical Logic

example: The Hilbert System

Set of axioms:

$$A \to (B \to A) \tag{A1}$$

$$(A \to (B \to C)) \to ((A \to B) \to (A \to C)) \tag{A2}$$

$$A \lor \neg A \tag{EM}$$

Single inference rule (*Modus Ponens*)

$$[\![A, A \to B]\!] \longrightarrow B \tag{MP}$$

Some provable tautologies:

$\neg\neg(A \lor \neg A)$	(nnEM)	$\neg(A \land B) \to \neg A \lor \neg B$	(DMdi)
$A \to \neg\neg A$	(DNi)	$\neg(A \lor B) \to \neg A \land \neg B$	(DMci)
$\neg\neg A \to A$	(DNe)	$\neg A \land \neg B \to \neg(A \lor B)$	(DMce)
$((A \to B) \to A) \to B$	(PL)	$\neg A \lor \neg B \to \neg(A \land B)$	(DMde)

Intuitionistic Logic

a.k.a. Constructive Logic

Set of axioms:

$$A \to (B \to A) \tag{A1}$$

$$(A \to (B \to C)) \to ((A \to B) \to (A \to C)) \tag{A2}$$

$$\cancel{A \vee \neg A} \tag{EM}$$

Single inference rule (*Modus Ponens*)

$$[\![A, A \to B]\!] \longrightarrow B \tag{MP}$$

Some provable tautologies:

$\neg\neg(A \vee \neg A)$	(nnEM)	$\cancel{\neg(A \wedge B) \to \neg A \vee \neg B}$	(DMdi)
$A \to \neg\neg A$	(DNi)	$\neg(A \vee B) \to \neg A \wedge \neg B$	(DMci)
$\cancel{\neg\neg A \to A}$	(DNe)	$\neg A \wedge \neg B \to \neg(A \vee B)$	(DMce)
$\cancel{((A \to B) \to A) \to B}$	(PL)	$\neg A \vee \neg B \to \neg(A \wedge B)$	(DMde)

Outline

Isabelle & Coq

First acquaintance

- based on **classical** higher-order logic
- created in 1986
  at University of Cambridge and Technische Universität München
- highly automated
- uses functional language `HOL`
- has large collection of formalised theories

- based on **intuitionistic** logic (Calculus of Inductive Constructions)
- created in 1984
  at INRIA (Paris, France)
- highly automated
- uses functional language `Gallina`
- has large collection of formalised theories

Isabelle & Coq

Definition of the basic datatypes

Isabelle & Coq

Definition of the basic datatypes

```
datatype bool =
  True | False

datatype nat =
  zero ("0") | Suc nat
```

Isabelle & Coq

Definition of the basic datatypes

```
datatype bool =
  True | False

datatype nat =
  zero ("0") | Suc nat
```

```
Inductive False : Prop := .

Inductive True : Prop := I : True.

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

Isabelle & Coq

Definition of a recursive function

Isabelle & Coq

Definition of a recursive function

```
fun add ::
  "nat ⇒ nat ⇒ nat"
where
  "add 0 n = n"
| "add (Suc m) n =
       Suc(add m n)"
```

Isabelle & Coq

Definition of a recursive function

```
fun add ::
  "nat ⇒ nat ⇒ nat"
where
  "add 0 n = n"
| "add (Suc m) n =
       Suc(add m n)"
```

```
Fixpoint add (n m: nat) : nat :=
match n with
  | O => m
  | S n′ => S (n′ + m)
end
```

Isabelle & Coq

Simple proof

A. Yushkovskiy,
S. Tripakis

Isabelle & Coq

Simple proof

```
lemma de_morgan:
  "(¬P ∨ ¬Q) ⟹ ¬(P ∧ Q)"
apply (rule notI)
apply (erule conjE)
apply (erule disjE)
apply
  (erule notE, assumption)+
done
```

Isabelle & Coq

Simple proof

```
lemma de_morgan:
  "(¬P ∨ ¬Q) ⟹ ¬(P ∧ Q)"
apply (rule notI)
apply (erule conjE)
apply (erule disjE)
apply
  (erule notE, assumption)+
done
```

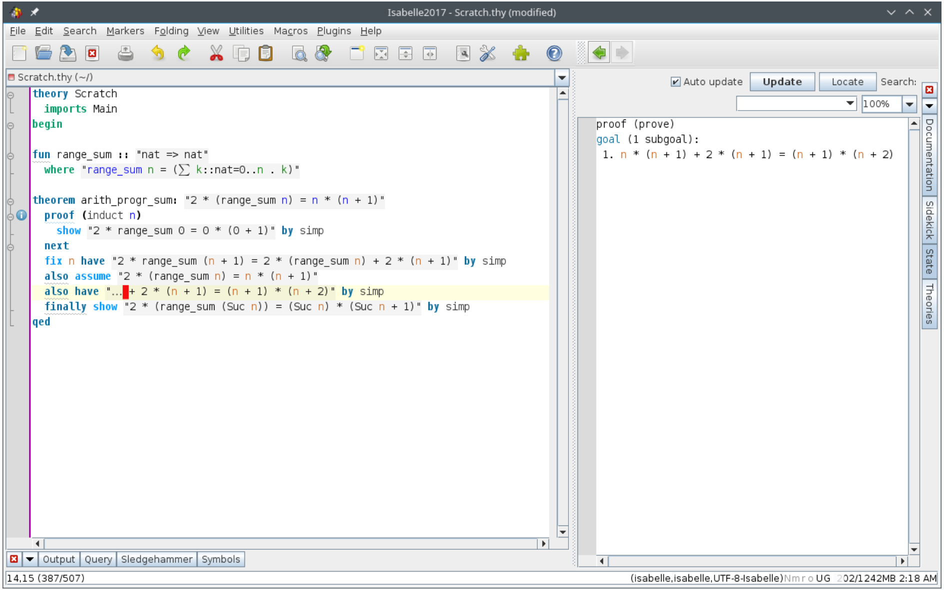
```
Theorem de_morgan:
  forall P Q : Prop,
    (¬P \/ ¬Q) -> ¬(P /\ Q).
Proof.
  intros P Q H [Hp Hq].
  destruct H as [Hnp | Hnq].
  - apply Hnp. assumption.
  - apply Hnq. assumption.
Qed.
```

Isabelle: Backward proof example

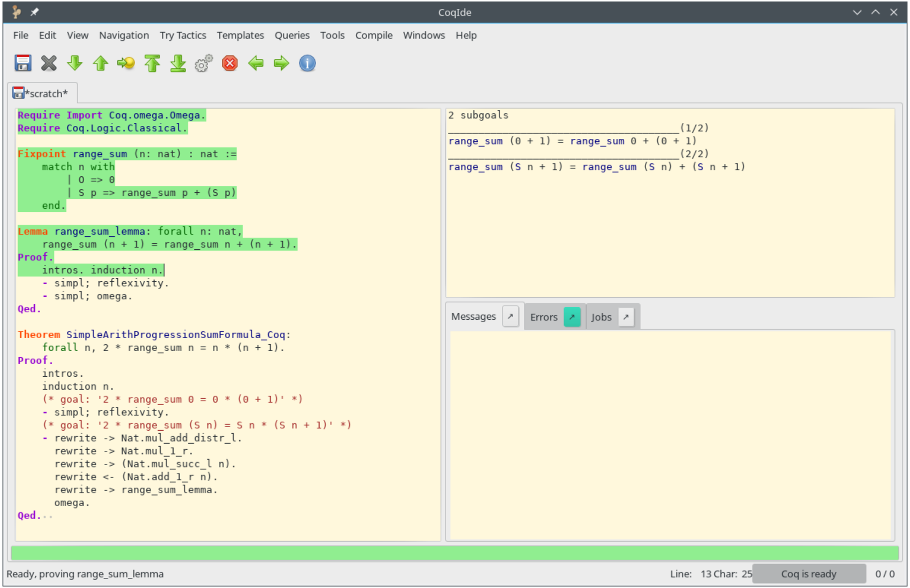
```
fun range_sum :: "nat => nat"
  where "range_sum n = (∑ k::nat=0..n . k)"

theorem arith_progr_sum: "2 * (range_sum n) = n * (n + 1)"
  proof (induct n)
    show "2 * range_sum 0 = 0 * (0 + 1)" by simp
  next
  fix n have "2 * range_sum (n + 1) = 2 * (range_sum n) + 2 * (n + 1)" by simp
  also assume "2 * (range_sum n) = n * (n + 1)"
  also have "... + 2 * (n + 1) = (n + 1) * (n + 2)" by simp
  finally show "2 * (range_sum (Suc n)) = (Suc n) * (Suc n + 1)" by simp
qed
```

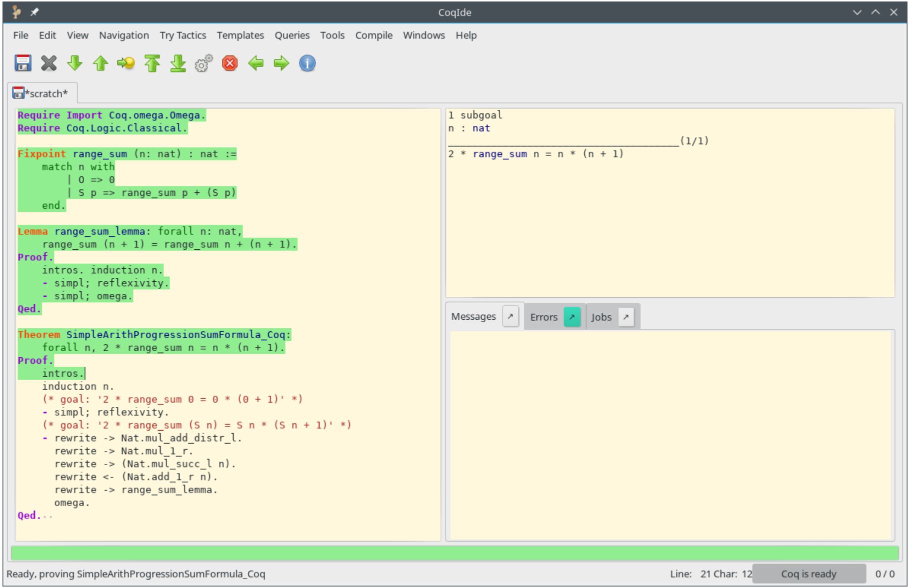
Isabelle: Proof process



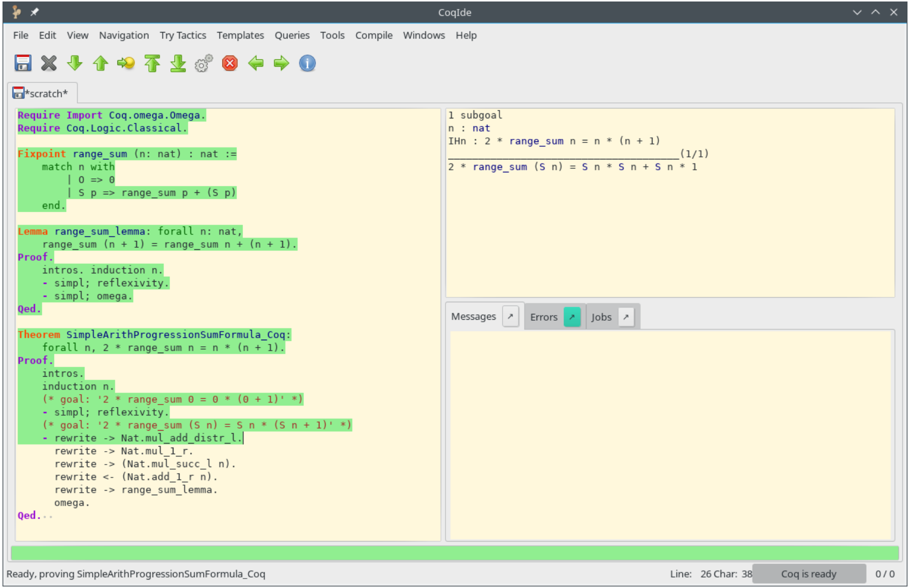
Coq: Proof process



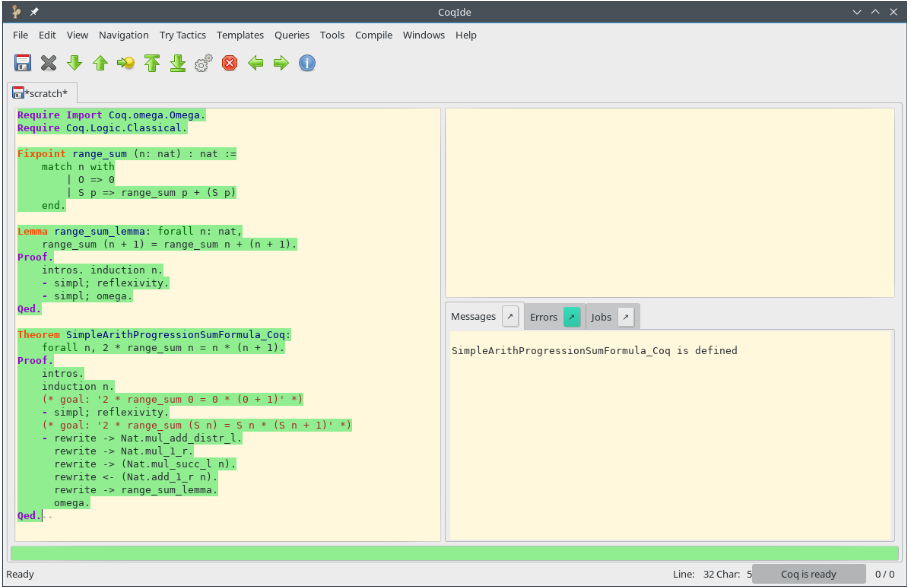
Coq: Proof process



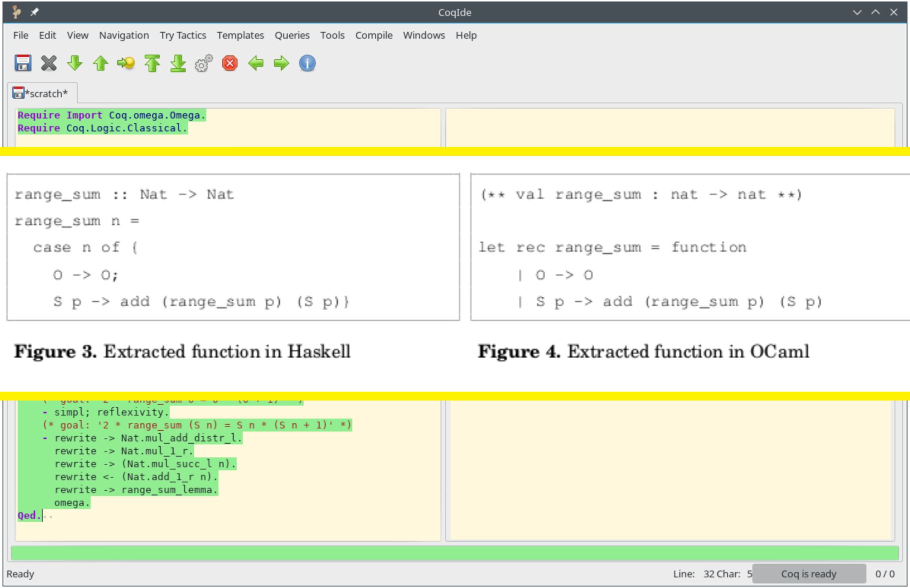
Coq: Proof process



Coq: Proof process



Coq: Proof process + verified code extraction




```
range_sum :: Nat -> Nat
range_sum n =
  case n of {
    O -> O;
    S p -> add (range_sum p) (S p)}
```

Figure 3. Extracted function in Haskell

```
(** val range_sum : nat -> nat **)

let rec range_sum = function
    | O -> O
    | S p -> add (range_sum p) (S p)
```

Figure 4. Extracted function in OCaml

Summary

- Two widespread theorem provers were considered: Isabelle and Coq
- The tools are based on different logics $\Rightarrow$
  - unprovable statements and invalid classical proofs in Coq
  - sometimes more complex proof in Coq
  - ⋆ *constructive* proof in Coq
- Nonetheless, they both may be used to solve applied problems, such as software testing and verification