

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

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 Professor of Philosophy Columbia University
University of Virginia Emory 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 S
gesch
oder
wiese
Ausfu
bis z
liefert

von S
gesch
oder
wiese
Ausfu
bis z
liefert

Auf Grund der obigen Prüfbarkeit gilt für innerhalb eines bestimmten endlichen Hauptsystems konzipierte Eigenschaften der Satz von 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



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Outline

Foundations of Formal Approach

- A formal system

- Classical and Intuitionistic logics

Isabelle & Coq

- First acquaintance

- Basic syntax

- Proof examples

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of Formal Approach

- A formal system

- Classical and Intuitionistic
logics

Isabelle & Coq

- First acquaintance

- Basic syntax

- Proof examples

Summary

Outline

Foundations of Formal Approach

A formal system

Classical and Intuitionistic logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Elements of a formal system

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

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Elements of a formal system

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

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

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

Definition of the formal system

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

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Definition of the formal system

The *formal system* is a quadruple $\Gamma = \langle A, V, \Omega, R \rangle$, 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

$\psi_1 \xrightarrow{\tau_1} \psi_2 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_{n-1}} \psi_n$, where $\psi_i \in A \cup \{\psi_k\}_{k=1}^{i-1}$

Definition of the formal system

The *formal system* is a quadruple $\Gamma = \langle A, V, \Omega, R \rangle$, 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

$\psi_1 \xrightarrow{\tau_1} \psi_2 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_{n-1}} \psi_n$, 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:

$$A \rightarrow (B \rightarrow A) \quad (\text{A1})$$

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \quad (\text{A2})$$

$$A \vee \neg A \quad (\text{EM})$$

Single inference rule (*Modus Ponens*)

$$\llbracket A, A \rightarrow B \rrbracket \longrightarrow B \quad (\text{MP})$$

Some provable tautologies:

$$\neg\neg(A \vee \neg A) \quad (\text{nnEM})$$

$$A \rightarrow \neg\neg A \quad (\text{DNi})$$

$$\neg\neg A \rightarrow A \quad (\text{DNe})$$

$$((A \rightarrow B) \rightarrow A) \rightarrow B \quad (\text{PL})$$

$$\neg(A \wedge B) \rightarrow \neg A \vee \neg B \quad (\text{DMdi})$$

$$\neg(A \vee B) \rightarrow \neg A \wedge \neg B \quad (\text{DMci})$$

$$\neg A \wedge \neg B \rightarrow \neg(A \vee B) \quad (\text{DMce})$$

$$\neg A \vee \neg B \rightarrow \neg(A \wedge B) \quad (\text{DMde})$$

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Intuitionistic Logic

a.k.a. Constructive Logic

Set of axioms:

$$A \rightarrow (B \rightarrow A) \quad (\text{A1})$$

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \quad (\text{A2})$$

$$\cancel{A \vee \neg A} \quad (\text{EM})$$

Single inference rule (*Modus Ponens*)

$$\llbracket A, A \rightarrow B \rrbracket \longrightarrow B \quad (\text{MP})$$

Some provable tautologies:

$$\neg\neg(A \vee \neg A) \quad (\text{nnEM})$$

$$A \rightarrow \neg\neg A \quad (\text{DNi})$$

$$\cancel{\neg\neg A \rightarrow A} \quad (\text{DNe})$$

$$\cancel{((A \rightarrow B) \rightarrow A) \rightarrow B} \quad (\text{PL})$$

$$\cancel{\neg(A \wedge B) \rightarrow \neg A \vee \neg B} \quad (\text{DMdi})$$

$$\neg(A \vee B) \rightarrow \neg A \wedge \neg B \quad (\text{DMci})$$

$$\neg A \wedge \neg B \rightarrow \neg(A \vee B) \quad (\text{DMce})$$

$$\neg A \vee \neg B \rightarrow \neg(A \wedge B) \quad (\text{DMde})$$

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Outline

Foundations of Formal Approach

A formal system

Classical and Intuitionistic logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Definition of the basic datatypes



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Definition of the basic datatypes



```
datatype bool =  
  True | False
```

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



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

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.
```

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Definition of a recursive function



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Definition of a recursive function



```
fun add ::  
  "nat  $\Rightarrow$  nat  $\Rightarrow$  nat"  
where  
  "add 0 n = n"  
| "add (Suc m) n =  
  Suc (add m n) "
```



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Definition of a recursive function



```
fun add ::  
  "nat  $\Rightarrow$  nat  $\Rightarrow$  nat"  
where  
  "add 0 n = n"  
| "add (Suc m) n =  
  Suc (add m n) "
```



```
Fixpoint add (n m: nat) : nat :=  
match n with  
| 0 => m  
| S n' => S (n' + m)  
end
```

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Simple proof



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Simple proof



```
lemma de_morgan:
  " ( $\neg P \vee \neg Q$ )  $\implies \neg (P \wedge Q)$  "
apply (rule notI)
apply (erule conjE)
apply (erule disjE)
apply
  (erule notE, assumption)+
done
```



Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle & Coq

Simple proof



```
lemma de_morgan:
  " ( $\neg P \vee \neg Q$ )  $\implies \neg(P \wedge Q)$  "
  apply (rule notI)
  apply (erule conjE)
  apply (erule disjE)
  apply
    (erule notE, assumption)+
  done
```



```
Theorem de_morgan:
  forall P Q : Prop,
    ( $\neg P \wedge \neg Q$ )  $\rightarrow \neg(P \vee Q)$  .
Proof.
  intros P Q H [Hp Hq].
  destruct H as [Hnp | Hnq].
  - apply Hnp. assumption.
  - apply Hnq. assumption.
Qed.
```

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

Isabelle: Backward proof example

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

```
fun range_sum :: "nat => nat"
  where "range_sum n = ( $\sum$  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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

The screenshot shows the Isabelle2017 Scratch.thy (modified) window. The left pane displays the source code of the theorem script, and the right pane shows the proof state.

Source Code (Left Pane):

```
theory Scratch
  imports Main
begin

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
```

Proof State (Right Pane):

```
proof (prove)
goal (1 subgoal):
  1. n * (n + 1) + 2 * (n + 1) = (n + 1) * (n + 2)
```

The status bar at the bottom indicates the file path (14.15 (387/507)) and the system information ((isabelle,isabelle,UTF-8-Isabelle)NmroUG 202/1242MB 2:18 AM).

Coq: Proof process

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

The screenshot shows the CoqIDE interface. The left pane contains a Coq script defining a function `range_sum` and proving a lemma and theorem about it. The right pane shows the current goals of the proof.

```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

Fixpoint range_sum (n: nat) : nat :=
  match n with
  | 0 => 0
  | S p => range_sum p + (S p)
  end.

Lemma range_sum_lemma: forall n: nat,
  range_sum (n + 1) = range_sum n + (n + 1).
Proof.
  intros. induction n.
  - simpl; reflexivity.
  - simpl; omega.
Qed.

Theorem SimpleArithProgressionSumFormula_Coq:
  forall n, 2 * range_sum n = n * (n + 1).
Proof.
  intros.
  induction n.
  (* goal: '2 * range_sum 0 = 0 * (0 + 1)' *)
  - simpl; reflexivity.
  (* goal: '2 * range_sum (S n) = S n * (S n + 1)' *)
  - rewrite -> Nat.mul_add_distr_l.
    rewrite -> Nat.mul_l_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_l_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed...
```

The right pane shows two subgoals:

$$\text{range_sum } (0 + 1) = \text{range_sum } 0 + (0 + 1) \quad (1/2)$$
$$\text{range_sum } (S \ n + 1) = \text{range_sum } (S \ n) + (S \ n + 1) \quad (2/2)$$

The bottom status bar indicates: Ready, proving range_sum_lemma. Line: 13 Char: 25 Coq is ready 0 / 0

Coq: Proof process

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

The screenshot shows the CoqIDE interface with a menu bar (File, Edit, View, Navigation, Try Tactics, Templates, Queries, Tools, Compile, Windows, Help) and a toolbar. The main editor displays a Coq script for proving the sum formula for an arithmetic progression. The script defines a fixpoint for the sum, a lemma for its recursive property, and a theorem for the closed-form formula. The proof of the theorem is partially completed, showing the induction step and the application of the lemma. The right-hand pane shows the current goal state, which is a subgoal for the induction step. The bottom status bar indicates the current file and the Coq engine's readiness.

```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

Fixpoint range_sum (n: nat) : nat :=
  match n with
  | 0 => 0
  | S p => range_sum p + (S p)
  end.

Lemma range_sum_lemma: forall n: nat,
  range_sum (n + 1) = range_sum n + (n + 1).
Proof.
  intros. induction n.
  - simpl; reflexivity.
  - simpl; omega.
Qed.

Theorem SimpleArithProgressionSumFormula_Coq:
  forall n, 2 * range_sum n = n * (n + 1).
Proof.
  intros.
  induction n.
  (* goal: '2 * range_sum 0 = 0 * (0 + 1)' *)
  - simpl; reflexivity.
  (* goal: '2 * range_sum (S n) = S n * (S n + 1)' *)
  - rewrite -> Nat.mul_add_distr_l.
    rewrite -> Nat.mul_l_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_l_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed...
```

1 subgoal
n : nat
2 * range_sum n = n * (n + 1) (1/1)

Messages Errors Jobs

Ready, proving SimpleArithProgressionSumFormula_Coq Line: 21 Char: 12 Coq is ready 0 / 0

Coq: Proof process

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

The screenshot shows the CoqIDE interface with a file named `*scratch*`. The left pane contains a Coq script defining a function `range_sum` and proving a theorem about the sum of an arithmetic progression. The right pane shows a subgoal and its corresponding mathematical formula. The bottom status bar indicates the current line and character position, and the Coq engine's status.

```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

Fixpoint range_sum (n: nat) : nat :=
  match n with
  | 0 => 0
  | S p => range_sum p + (S p)
  end.

Lemma range_sum_lemma: forall n: nat,
  range_sum (n + 1) = range_sum n + (n + 1).
Proof.
  intros. induction n.
  - simpl; reflexivity.
  - simpl; omega.
Qed.

Theorem SimpleArithProgressionSumFormula_Coq:
  forall n, 2 * range_sum n = n * (n + 1).
Proof.
  intros.
  induction n.
  (* goal: '2 * range_sum 0 = 0 * (0 + 1)' *)
  - simpl; reflexivity.
  (* goal: '2 * range_sum (S n) = S n * (S n + 1)' *)
  - rewrite -> Nat.mul_add_distr_l.
    rewrite -> Nat.mul_l_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_l_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed...
```

subgoal
n : nat
IHn : 2 * range_sum n = n * (n + 1)
$$2 * \text{range_sum } (S\ n) = S\ n * S\ n + S\ n * 1$$

Messages Errors Jobs

Ready, proving SimpleArithProgressionSumFormula_Coq Line: 26 Char: 38 Coq is ready 0 / 0

Coq: Proof process

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

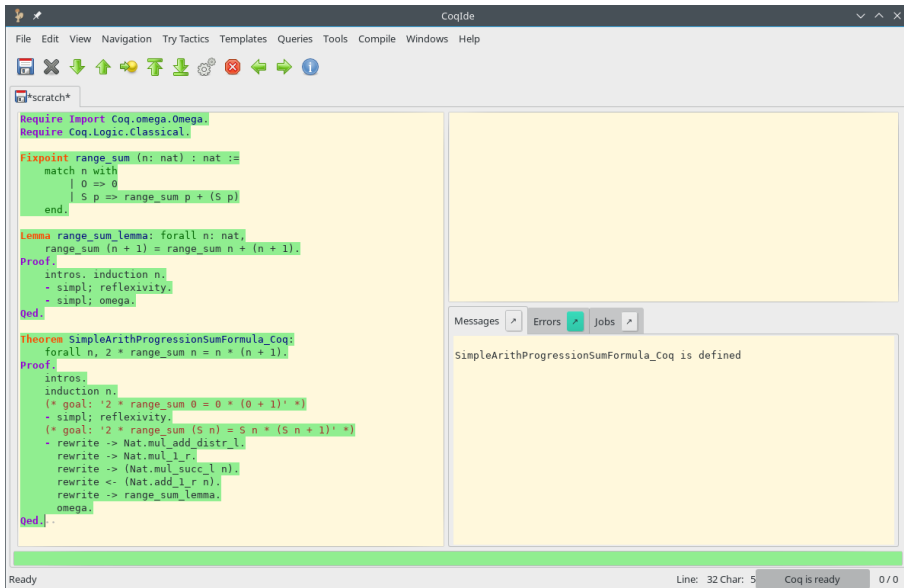
Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary



The screenshot shows the CoqIDE interface. The main editor displays a Coq script for calculating the sum of the first n natural numbers. The script defines a function `range_sum` using a fixpoint, proves a lemma `range_sum_lemma` by induction, and finally proves a theorem `SimpleArithProgressionSumFormula_Coq` by induction and rewriting. The right-hand pane shows the output of the `Qed` command, indicating that the theorem is defined. The status bar at the bottom shows 'Ready', 'Line: 32 Char: 5', 'Coq is ready', and '0 / 0'.

```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

Fixpoint range_sum (n: nat) : nat :=
  match n with
  | 0 => 0
  | S p => range_sum p + (S p)
  end.

Lemma range_sum_lemma: forall n: nat,
  range_sum (n + 1) = range_sum n + (n + 1).
Proof.
  intros. induction n.
  - simpl; reflexivity.
  - simpl; omega.
Qed.

Theorem SimpleArithProgressionSumFormula_Coq:
  forall n, 2 * range_sum n = n * (n + 1).
Proof.
  intros.
  induction n.
  (* goal: '2 * range_sum 0 = 0 * (0 + 1)' *)
  - simpl; reflexivity.
  (* goal: '2 * range_sum (S n) = S n * (S n + 1)' *)
  - rewrite -> Nat.mul_add_distr_l.
    rewrite -> Nat.mul_l_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_l_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed.
```

Messages Errors Jobs

SimpleArithProgressionSumFormula_Coq is defined

Ready Line: 32 Char: 5 Coq is ready 0 / 0

Coq: Proof process + verified code extraction

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Introduction

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Isabelle & Coq

First acquaintance

Basic syntax

Proof examples

Summary

The screenshot shows the CoqIDE interface with a menu bar (File, Edit, View, Navigation, Try Tactics, Templates, Queries, Tools, Compile, Windows, Help) and a toolbar. The editor displays a Coq script in a file named *scratch*. The script defines a function `range_sum` and proves a theorem about it. The proof is completed with `Qed.`. The interface is divided into two panes. The left pane shows the extracted function in Haskell, and the right pane shows the extracted function in OCaml. The status bar at the bottom indicates 'Ready', 'Line: 32 Char: 5', 'Coq is ready', and '0 / 0'.

```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

range_sum :: Nat -> Nat
range_sum n =
  case n of {
    0 -> 0;
    S p -> add (range_sum p) (S p) }
```

Figure 3. Extracted function in Haskell

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

let rec range_sum = function
  | 0 -> 0
  | S p -> add (range_sum p) (S p)
```

Figure 4. Extracted function in OCaml

```
(* goal: 2 * range_sum n = 0 -> 0 *)
- simpl; reflexivity.
(* goal: '2 * range_sum (S n) = S n * (S n + 1)' *)
- rewrite -> Nat.mul_add_distr_l.
  rewrite -> Nat.mul_l_r.
  rewrite -> (Nat.mul_succ_l n).
  rewrite <- (Nat.add_l_r n).
  rewrite -> range_sum_lemma.
  omega.
Qed.
```

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