

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Introduction

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 Funktionentheorie¹⁾.

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 als „Disjunktionsprinzip“ auftretenden) Satzes vom ausgeschlossenen Dritten, daß entweder p oder q wenigstens 6 Elemente enthält.

Ebenso: Wenn man in der elementaren Arithmetik bewiesen hat, daß, wenn keine der ganzen positiven Zahlen a, a_1, a_2, \dots, a_n durch die Primzahl c teilbar ist,

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Introduction

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

PRINCIPLES OF MATHEMATICAL LOGIC



Über die Bedeutung des Satzes vom ausgeschlossenen Dritten in der Mathematik, insbesondere in der Funktionentheorie¹⁾.

Von L. E. J. Brouwer in Amsterdam.

§ 1.

von S
gesch
oder
wiese
Ausfe
bis z
liefert
funge
bzw.
demz
mitur

licher
Dritte
oder
mente
Unm



wenigstens 11 Elemente enthält, so folgt hieraus auf Grund des (in diesem Falle als „Disjunktionsprinzip“ auftretenden) Satzes vom ausgeschlossenen Dritten, daß entweder p oder q wenigstens 6 Elemente enthält.

Ebenso: Wenn man in der elementaren Arithmetik bewiesen hat, daß, wenn keine der ganzen positiven Zahlen a_1, a_2, \dots, a_n durch die Primzahl c teilbar ist,

Outline

Foundations of Formal Approach

A formal system

Classical and Intuitionistic logics

Two Theorem Provers

Isabelle

Coq

Comparison of the theorem provers

Comparison

Proof examples

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

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 provable from Φ — $\Phi \vdash \phi$
- ▶ A tautology \top — $\vdash \phi$
- ▶ A contradiction \perp — $\vdash \neg \phi$

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Definition of the formal system

A *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

A *formal proof* of the formula ϕ is a finite sequence of judgements

$$\psi_1 \xrightarrow{\tau_1} \psi_2 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_n} \psi_n$$

where each ψ_i is either an axiom ϕ_{A_i} , or a formula inferred from the set of previously derived formulas according to the rules of inference.

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

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

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Intuitionistic Logic

a.k.a. Constructive Logic

Set of axioms:

$$A \rightarrow (B \rightarrow A) \quad (A1)$$

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \quad (A2)$$

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

Single inference rule (*Modus Ponens*)

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

Some provable tautologies:

$$\neg\neg(A \vee \neg A) \quad (nnEM)$$

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

$$\neg\neg A \rightarrow A \quad (DNe)$$

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

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

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

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

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Isabelle: first acquaintance

- ▶ a generic proof assistant
- ▶ based on classical higher-order logic
- ▶ created in 1986 by
 - ▶ Larry Paulson @ University of Cambridge, and
 - ▶ Tobias Nipkow @ Technische Universität München
- ▶ uses powerful functional language HOL
- ▶ the proof system core *Isabelle* is extended by various theories: Isabelle/HOL, Isabelle/ZF, Isabelle/CCL, etc.

Example 1: Definition of basic datatypes

```
datatype bool =  
  True | False  
  
datatype nat =  
  zero ("0") | Suc nat
```

Example 2: Definition of addition over nat

```
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

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Coq: first acquaintance

- ▶ a formal proof management system
- ▶ based intuitionistic logic (Calculus of Inductive Constructions)
- ▶ created at INRIA (Paris, France) in 1984
- ▶ uses powerful functional language Gallina
- ▶ has large collection of formalised theories
- ▶ widely used in software verification (proof code extraction)

Example 3: Definition of basic datatypes

```
Inductive False : Prop := .

Inductive True : Prop := I : True.

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

Example 4: Definition of addition over nat

```
Fixpoint add (n m: nat) : nat :=
  match n with
  | O => m
  | S n' => S (n' + m)
  end
where "n + m" :=
  (add n m) : nat_scope.
```

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Comparison

Major similarities:

- ▶ both work in a similar way of *verifying* the proof or *assisting* in creation of the new one
- ▶ $premises \xrightarrow{tactics} goals$ (forward proof)
- ▶ $goals \xrightarrow{tactics} premises$ (backward proof)
- ▶ both have large amount of libraries with formalised theories
- ▶ both dispose the set of highly automated tactics
- ▶ both are being actively developed these days

Major differences:

- ▶ based on different logics \Rightarrow
 - ★ unprovable statements and invalid proofs in Coq
 - ★ sometimes more complex proof in Coq
 - ★ *constructive* proof in Coq

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system
Classical and Intuitionistic
logics

Two Theorem
Provers

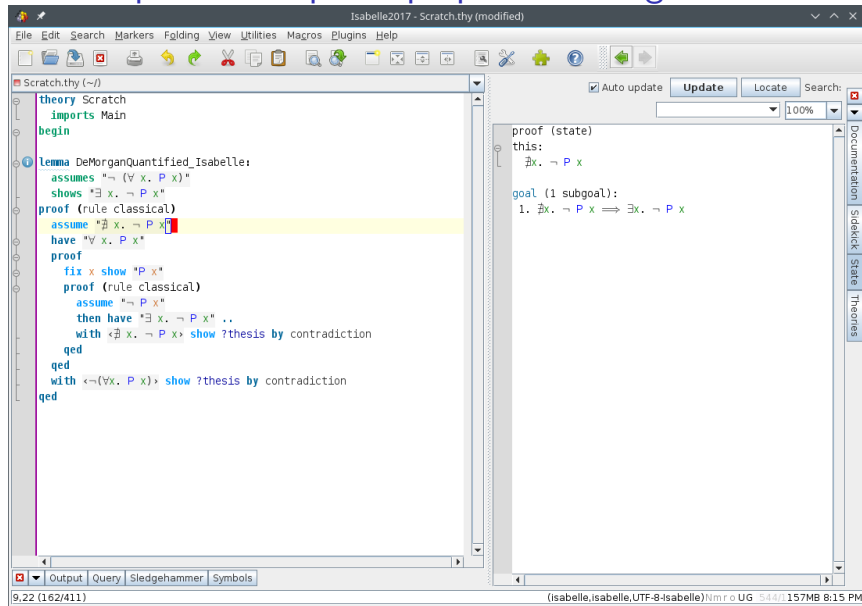
Isabelle
Coq

Comparison of the
theorem provers

Comparison
Proof examples

Summary

Isabelle: proof example in propositional logic



The screenshot shows the Isabelle2017 IDE with a file named 'Scratch.thy (modified)'. The main editor displays a theory 'Scratch' with a lemma 'DeMorganQuantified_Isabelle'. The lemma states that if $\neg (\forall x. P x)$, then $\exists x. \neg P x$. The proof is structured as follows:

```
theory Scratch
  imports Main
begin

lemma DeMorganQuantified_Isabelle:
  assumes "¬ (∀ x. P x)"
  shows "∃ x. ¬ P x"
proof (rule classical)
  assume "¬ ∃ x. ¬ P x"
  have "∀ x. P x"
  proof
    fix x show "P x"
    proof (rule classical)
      assume "¬ P x"
      then have "∃ x. ¬ P x" ..
      with <¬ ∃ x. ¬ P x> show ?thesis by contradiction
    qed
  qed
  with <¬ (∀ x. P x)> show ?thesis by contradiction
qed
```

The right-hand pane shows the current state of the proof, with the goal $\exists x. \neg P x$ and a subgoal $\exists x. \neg P x \Rightarrow \exists x. \neg P x$. The bottom status bar indicates the version 9.22 (162/411) and the time 8:15 PM.

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system
Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle
Coq

Comparison of the
theorem provers

Comparison
Proof examples

Summary

Isabelle: proof example over nat

The screenshot shows the Isabelle2017 Scratch.thy (modified) editor. The left pane displays the source code of a theory named 'Scratch'. The right pane shows the proof state for the goal (1 subgoal): $1. n * (n + 1) + 2 * (n + 1) = (n + 1) * (n + 2)$.

Source Code (Scratch.thy):

```
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 is 14,15 (387/507) bytes and the system is Isabelle, Isabelle, UTF-8-Isabelle, Nm r o UG, 202/1242MB, 2:18 AM.

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system
Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle
Coq

Comparison of the
theorem provers

Comparison
Proof examples

Summary

Coq: proof example in propositional logic

The screenshot shows the CoqIDE window with a file named `*scratch*`. The main editor contains the following Coq code:

```
Theorem DeMorganPropositional_Coq:
  forall P Q : Prop, ~(P /\ Q) <-> ~P /\ ~Q.
Proof.
  (* 'tauto' automatically proves the equation *)
  intros P Q. unfold iff.
  split.
  - intros H_not_or. unfold not. constructor.
    + intro H_P. apply H_not_or. left. apply H_P.
    + intro H_Q. apply H_not_or. right. apply H_Q.
  - intros H_and_not_H_or.
    destruct H_and_not as [H_not_P H_not_Q].
    destruct H_or as [H_P | H_Q].
    + apply H_not_P. assumption.
    + apply H_not_Q. assumption.
Qed.
```

The right-hand pane shows the current subgoal:

```
|1 subgoal
P, Q : Prop
H_not_or : ~ (P /\ Q)
H_Q : Q
----- (1/1)
P /\ Q
```

At the bottom, the status bar indicates "Ready, proving DeMorganPropositional_Coq", "Line: 9 Char: 33", and "Coq is ready 0 / 0".

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Coq: proof example over nat

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 with various icons. The main editor displays a Coq script for proving a formula about the sum of an arithmetic progression. The script defines a fixpoint `range_sum`, a lemma `range_sum_lemma`, and a theorem `SimpleArithProgressionSumFormula_Coq`. The proof of the theorem is in progress, with the current goal being `2 * range_sum (S n) = S n * (S n + 1)`. The right-hand pane shows the current goal and the induction hypothesis. The bottom status bar indicates 'Ready, proving SimpleArithProgressionSumFormula_Coq' and 'Line: 26 Char: 38 Coq is ready 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_1_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_1_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed...
```

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

Messages Errors Jobs

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

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system

Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle

Coq

Comparison of the
theorem provers

Comparison

Proof examples

Summary

Coq: proof example over nat + verified code extraction

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system
Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle
Coq

Comparison of the
theorem provers

Comparison
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 window displays a Coq script in a file named `*scratch*`. The script defines a function `range_sum` using a `Fixpoint` and proves a lemma `range_sum` using `induction` and `rewrite`. The script is highlighted in green. Below the script, two extracted versions of the function are shown in yellow boxes. The left box, labeled **Figure 3. Extracted function in Haskell**, shows the Haskell code for `range_sum`. The right box, labeled **Figure 4. Extracted function in OCaml**, shows the OCaml code for `range_sum`. The status bar at the bottom indicates 'Ready, proving SimpleArithProgressionSumFormula_Coq', 'Line: 26 Char: 38', '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 =>
    end.

Lemma range_sum
  range_sum (n
Proof.
  intros. indu
  - simpl; ref
  - simpl; ome
Qed.

Theorem SimpleAr
  forall n, 2
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_1_r.
    rewrite -> (Nat.mul_succ_l n).
    rewrite <- (Nat.add_1_r n).
    rewrite -> range_sum_lemma.
    omega.
Qed...
```

Figure 3. Extracted function in Haskell

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

Figure 4. Extracted function in OCaml

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

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


Summary

- ▶ Two widespread theorem provers were considered: Isabelle and Coq
- ▶ The **key difference** between them lie in differences between logical theories they based on
- ▶ Nonetheless, they both may be used to solve applied problems, such as software testing and verification

Comparison of two
theorem provers:
Isabelle & Coq

A. Yushkovskiy,
S. Tripakis

Foundations of
Formal Approach

A formal system
Classical and Intuitionistic
logics

Two Theorem
Provers

Isabelle
Coq

Comparison of the
theorem provers

Comparison
Proof examples

Summary