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

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.

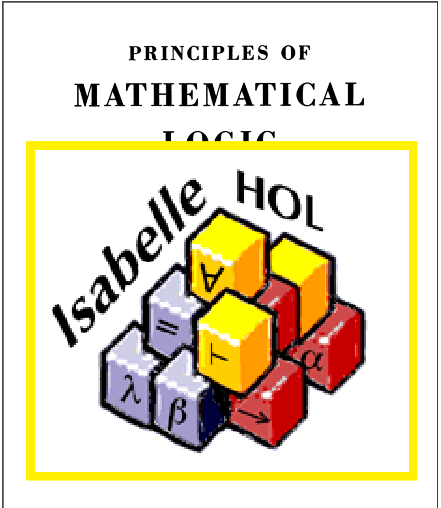
Innerhalb eines bestimmten endlichen „Hauptsystems" können Eigenschaften von Systemen, d. h. Abbildbarkeiten von Systemen auf andere Systeme mit vorgeschriebenen Elementkorrespondenzen, immer *geprüft* (d. h. entweder bewiesen oder ad absurdum geführt) werden; die durch die betreffende Eigenschaft angewiesene 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 als „Disjunktionsprinzip" auftretenden) Satzes vom ausgeschlossenen Dritten, daß entweder p oder q wenigstens 6 Elemente enthält.

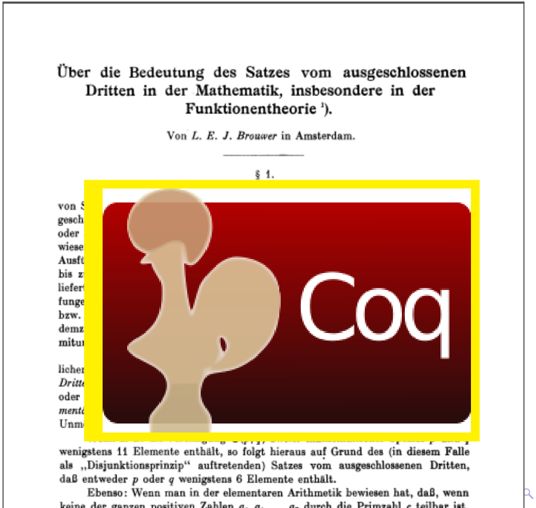
Ebenso: Wenn man in der elementaren Arithmetik bewiesen hat, daß, wenn

Introduction



PRINCIPLES OF

MATHEMATICAL



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

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

§ 1.

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

Outline

Elements of a Formal System

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

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

- 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 $\bot$ — $\vdash \neg\phi$

Definition of the formal system

A *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

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 the rules of inference.

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:

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

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

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

Single inference rule (*Modus Ponens*)

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

Some provable tautologies:

$\neg\neg(A \lor \neg A)$	(nnEM)	$\cancel{\neg(A \land B) \to \neg A \lor \neg B}$	(DMdi)
$\cancel{A \to \neg\neg A}$	(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)
$\cancel{((A \to B) \to A) \to B}$	(PL)	$\neg A \lor \neg B \to \neg(A \land B)$	(DMde)

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

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

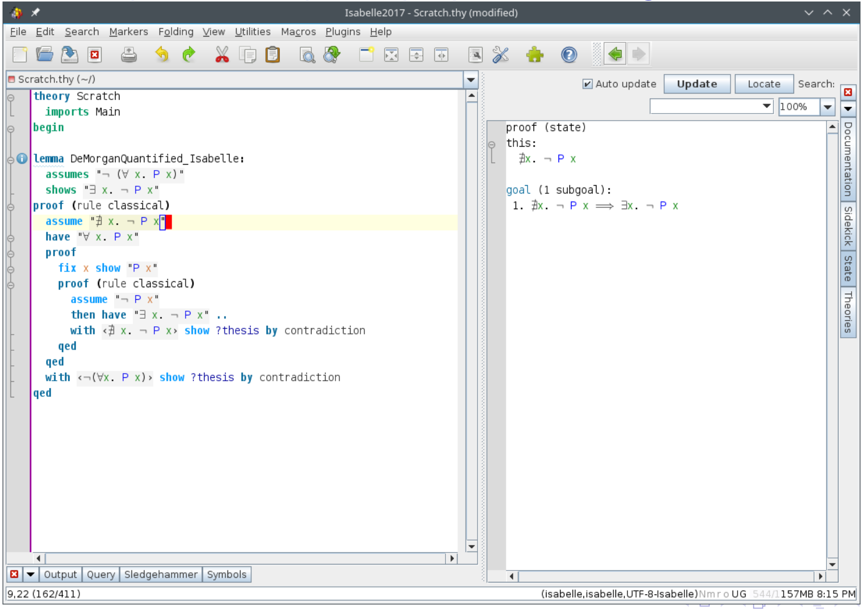
Major similarities:

- 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

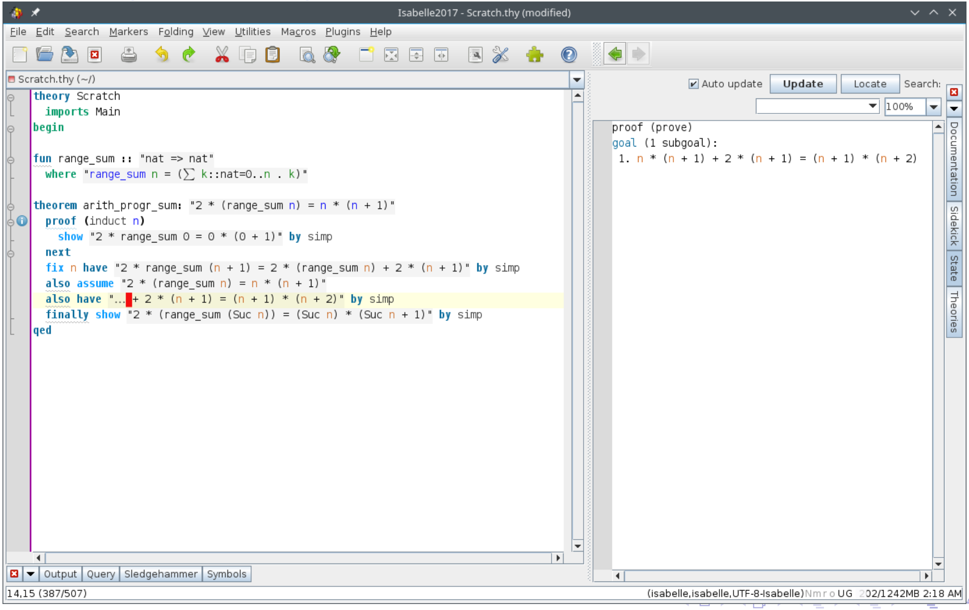
Major differences:

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

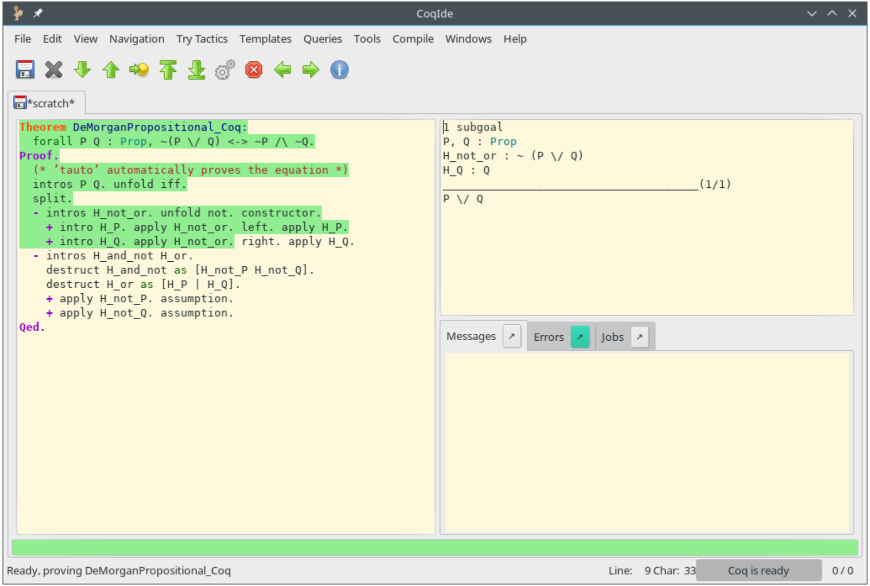
Isabelle: proof example in propositional logic



Isabelle: proof example over nat



Coq: proof example in propositional logic



Coq: proof example over `nat`

CoqIde

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

scratch

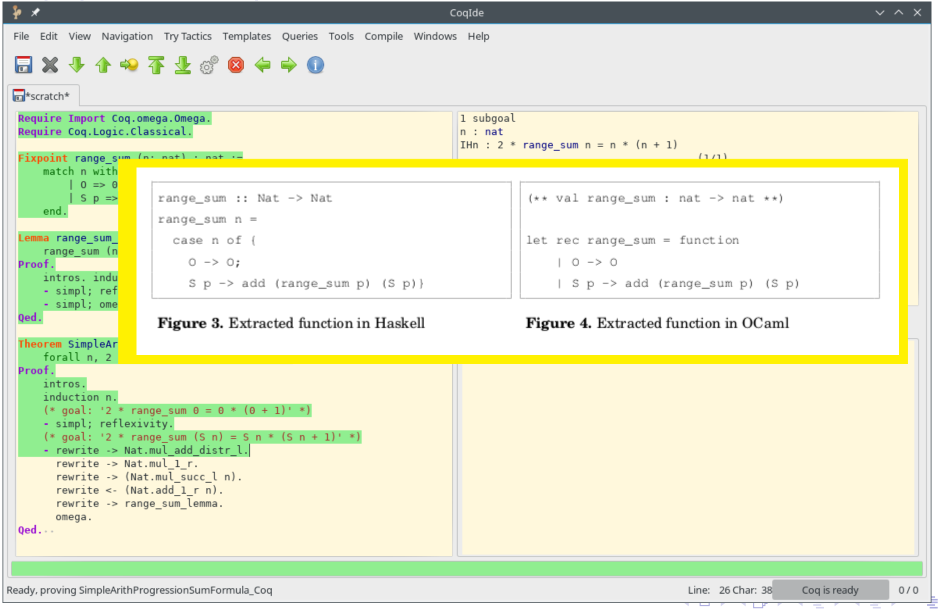
```
Require Import Coq.omega.Omega.
Require Coq.Logic.Classical.

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)
______________________________________(1/1)
2 * 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 example over `nat` + 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

A. Yushkovskiy, S. Tripakis

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