# Comparison of theorem provers

**Artem Yushkovskiy**
artem.yushkovskiy@aalto.fi

**Tutor**: Stavros Tripakis

## Abstract

*One of the useful applications mathematical logic theory is the Automated theorem proving. This is a set of techniques that allow one to verify mathematical statements mechanically using logical reasoning. Although, it can be used to solve engineering problems as well, for instance to prove security properties for a software system or an algorithm. Furthermore, automated theorem proving is an essential part of the Artificial Intelligence theory, which became highly evolving these days. In this paper, the bases of formal systems and automated deduction theory are described. Then, two widespread tools for automated theorem proving, Coq [1] and Isabelle [2], are compared with respect to expressive power and usability.*

*KEYWORDS: logic, proof theory, automated theorem prover, Coq, Isabelle.*

## 1 Introduction

Modern world cannot be imagined without mathematics. Almost every human technical achievement is based on the groundwork of mathematical methods. Formal models allow one both to describe existent phenomena and to develop new tools. Although in most cases formal models that one built work efficiently, in previous century mathematics experienced deep fundamental crisis caused by the need for a formal definition of the very bases of mathematics. That time many paradoxes in some mathematics have been discovered, so, in 1920s, the three main confrontation schools appeared: the *logicism* based on work of Gottlob Frege, Bertrand Russell, and Alfred Whitehead, which stated that mathematics is an extension of logic and therefore it can be reduced to logic partly or fully; the *formalism* led by David Hilbert, also called the 'proof theory', which advocated classical mathematics with its axiomatic approach; and the radical *intuitionism* led by Luitzen Brouwer, which rejected this formalisation as unnecessary and even meaningless, claiming that only the objects, which we know how to construct, may exist. Although the formalism was the leading school, all these schools have contributed important ideas and techniques to mathematics, philosophy and many other scientific and engineering areas [3].

One solution for the foundational crisis was proposed by the school of formalism, it is known as Hilbert's program, and aimed to base all existing theories on finite set of axioms, and prove that these sets are consistent [4]. Thus Hilbert proposed to reduce the consistency of all mathematics to basic arithmetic. Unfortunately, these intentions turned out to be rather unrealisable, when in 1931 Kurt Gödel published his two famous incompleteness theorems, that demonstrated the limitations of any formal axiomatic system containing basic arithmetic [5]. In particularly, he proved that all consistent axiomatic formulations of number theory are incomplete (such that there will always be statements which cannot be proved or disproved within that formulation), and that such formal system can not prove that itself it is consistent (i.e., it is able to prove that a statement and its negative are both true). Notwithstanding the fact that the whole mathematics can not be described in a single axiomatic system, the formal approach allows one to build though restricted, but fine, concise and verifiable theories.

Informally, *formal proof* is the sequence of statements, based on finite set of fundamental axioms and satisfying the rules of logical inference. *The axiom* is a statement claimed to be true evidently. *The logical inference* is the transfer from one statement (premise) to another (consequence), which preserve truth, while the rule of logical inference is a principle that allows one to infer the validity of such transfer.

In formal logic, inference is based entirely on the structure (i.e., form) of those statements, which allows one to apply basic logical rules to any type of proof and thus construct the formal system. The main goal of the formal system is to be verifiable, i.e. one could *check* its validity. At present, a lot of tools are being developing to automate the process of such checking to run it on the computer. In particular, the systems *Isabelle/ZF* [2], *Coq* [1] *PVS* [6], *ACL2* [7] work in a form of axiomatic set theory and allow the user to enter theorems and proofs into the computer, which then verifies that the proof is correct (these are also called sometimes 'proof assistants'). Another goal of constructing the formal system is having the computer to automatically *discover* the formal proof, which can rely either on induction, or on meta argument, or on higher-order logic. McCune's systems *Otter* [8] and *Prover9* [9] are commonly recognized as the state-of-the-art tools.

In current paper we consider only the systems, which are built to achieve the first goal, i.e. to verify existing proof, since <...>. Two aforementioned theorem provers Coq and Isabelle are examined for the purpose of revealing expressiveness, computation power and usability. These properties are described in detail in Section 6. In Section 2, one can find an overview of the history of logic, providing thorough definitions, typology and properties of formal system and formal proof; issues related to theoretical limitations of formal systems are discussed there as well. Section 3 describes basic properties of formal system, which may be interesting for analysing automated theorem provers. General methods for automated reasoning are given in Section 4. Possible applications areas for automated theorem provers are enumerated in Section 5. The comparison of selected theorem provers is given in Section 6.

<... results and author's personal contribution >

## 2 Theory of logical calculi

// TODO

MOCK from wiki: /* A formal system or logical calculus is any well-defined system of abstract thought based on the model of mathematics. A formal system need not be mathematical as such; for example, Spinoza's Ethics imitates the form of Euclid's Elements. Spinoza employed Euclidiean elements such as "axioms" or "primitive truths", rules of inferences etc. so that a calculus can be built using these. For nature of such primitive truths, one can consult Tarski's "Concept of truth for a formalized language". */

Basics briefly: set theory (?), its problems (paradoxes) (?), Zermelo–Fraenkel (?), formal languages (?), propositional and 1st ordered logic (introduce notation here), else?

MOCK from plato.stanford.edu: /* A third important consideration in the building of an automated reasoning program is the selection of the actual deduction calculus that will be used by the program to perform its inferences. As indicated before, the choice is highly dependent on the nature of the problem domain and there is a fair range of options available: General-purpose theorem proving and problem solving (first-order logic, simple type theory), program verification (first-order logic), distributed and concurrent systems (modal and temporal logics), program specification (intuitionistic logic), hardware verification (higher-order logic), logic programming (Horn logic), and so on. */

Type system (+dependent type, where type checking although may be undecidable, but it verifies the correctness of ), Nominal vs. structural type system

## 3 Basic properties of logical system

// TODO: link with previous sections

According to Formalists: Independence, Consistency, Completeness, Decideability.

Formalists defined four basic properties which every logical system must have:

1. Independence, which means that there aren't any superfluous axioms. There's no axiom that can be derived from the other axioms.
2. Consistency, which means that no theorem of the system contradicts another.
3. Completeness, which means the ability to derive all true formulae from the axioms.
4. Decidability, the Entscheidungsproblem, which asks for an algorithm that takes as input a statement and answers "Yes" or "No" according to whether the statement is universally valid, i.e., valid in every structure satisfying the axioms.

Formula (i.e. statement, program) may be:

- provable (either true or false, according to selected set of axioms)
- valid
- sound
- ...

## 4 Methods for automated reasoning

// TODO

// MOCK from plato.stanford.edu:

/* techniques in common words (and in introduced previously notation), e.g.:

- Clause rewriting
- Resolution
- Sequent Deduction
- Natural Deduction
- The Matrix Connection Method
- Term Rewriting (+lambda calculus)
- Mathematical Induction

*/

## 5 Some applications of theorem provers

// TODO

// e.g. from slides: https://www.andrew.cmu.edu/user/avigad/Talks/baltimore.pdf

/* Formal methods can be used to verify correctness: - verifying that a circuit description, an algorithm, or a network or security protocol meets its specification; or - verifying that a mathematical statement is true. Two approaches: - Model checking: reduce to a finite state space, and test exhaustively. - Interactive theorem proving: construct a formal axiomatic proof of correctness */

Also: Curry–Howard correspondence, the direct relationship between computer programs and mathematical proofs.

## 6 Comparison of some theorem provers

//TODO: in two words: here we consider Coq and Isabelle, bla-bla

MOCK from "Certified Programming with Dependent Types" by Adam Chlipala. /* ACL2 is notable in this field for having only a first-order language at its foundation. That is, you cannot work with functions over functions and all those other treats of functional programming. By giving up this facility, ACL2 can make broader assumptions about how well its proof automation will work, but we can generally recover the same advantages in other proof assistants when we happen to be programming in first-order fragments

Isabelle/HOL and Coq both support coding new proof manipulations in ML in ways that cannot lead to the acceptance of invalid proofs. Additionally, Coq includes a domain-specific language for coding decision procedures in normal Coq source code, with no need to break out into ML.

A language with dependent types may include references to programs inside of types. For instance, the type of an array might include a program expression giving the size of the array, making it possible to verify absence of out-of-bounds accesses statically. Dependent types can go even further than this, effectively capturing any correctness property in a type. PVS's dependent types are much more general, but they are squeezed inside the single mechanism of subset types, where a normal type is refined by attaching a predicate over its elements. Each member of the subset type is an element of the base type that satisfies the predicate. */

### 6.1 The Coq theorem prover

// TODO

Coq is a formal proof assistant system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs [1]. Coq uses the Calculus of Construction, a higher-order formalism for constructive proofs in natural deduction style, developed by Thierry Coquand [10]. The Calculus of Construction can be considered as an extension of the Curry–Howard isomorphism in a way that the latter associates a term in the simply typed lambda calculus with each natural-deduction proof in intuitionistic propositional logic, when the Calculus of Construction extends this isomorphism to proofs in the full intuitionistic predicate calculus, which includes proofs of quantified statements (which are also called "propositions").

## 6.2 The Isabelle theorem prover

// TODO

The Isabelle theorem prover is an interactive theorem prover, which relies on higher-order logic. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus [2].

MOCK from wiki: /* it is based on a small logical core to ease logical correctness. Isabelle is generic: it provides a meta-logic (a weak type theory), which is used to encode object logics like first-order logic (FOL), higher-order logic (HOL) or Zermelo–Fraenkel set theory (ZFC). Isabelle's main proof method is a higher-order version of resolution, based on higher-order unification. Though interactive, Isabelle also features efficient automatic reasoning tools, such as a term rewriting engine and a tableaux prover, as well as various decision procedures. Isabelle has been used to formalize numerous theorems from mathematics and computer science, like Gödel's completeness theorem, Gödel's theorem about the consistency of the axiom of choice, the prime number theorem, correctness of security protocols, and properties of programming language semantics. The Isabelle theorem prover is free software, released under the revised BSD license. */

## 6.3 Joint comparison

//TODO: in table:

- expressiveness of logic used
- time of proving
- num of supporting theories
- set of techniques to prove automatically
- Volume of proof (as text)
- num of user interaction steps
- usability

- etc ...

## 7 Results

// TODO

Conclusions of comparison

## 8 Future work

// TODO

< to apply this survey to software verification >

## References

[1] "Coq proof assistant." https://coq.inria.fr/.

[2] "Isabelle, a generic proof assistant." https://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[3] J. Ferreirós, "The crisis in the foundations of mathematics," 2008.

[4] R. Zach, "Hilbert's program then and now," in *Philosophy of Logic. Handbook of the Philosophy of Science*, vol. 5, p. 411–447, Amsterdam: Elsevier, 2006.

[5] P. Raatikainen, "Gödel's incompleteness theorems," 2015.

[6] "Pvs specification and verification system." http://pvs.csl.sri.com/.

[7] "Acl2: a computational logic for applicative common lisp." http://www.cs.utexas.edu/users/moore/acl2/.

[8] W. McCune, "Otter and mace2," 2003. https://www.cs.unm.edu/~mccune/otter/.

[9] W. McCune, "Prover9 and mace4."

[10] G. H. T. Coquard, "The calculus of constructions," 1986.