# ProofJudge: Automated Proof Judging Tool for Learning Mathematical Logic

**Jørgen Villadsen**
DTU Compute, Denmark, jovi@dtu.dk

## ABSTRACT

Today we have software in many artefacts, from medical devices to cars and airplanes, and the software must not only be efficient and intelligent but also reliable and secure. Tests can show the presence of bugs but cannot guarantee their absence. A machine-checked proof using mathematical logic provides strong evidence for software correctness but it requires advanced knowledge and skills. We have developed a tool which helps the student to practice their skills and also allows a better conceptual understanding of state-of-the-art proof assistants. Previously the proofs has been carried out using pen and paper because no adequate tool was available. The learning problem is how to make abstract concepts of logic as concrete as possible.

ProofJudge is a computer system and teaching approach for teaching mathematical logic and automated reasoning which augments the e-learning tool NaDeA (Natural Deduction Assistant). We believe that automatic feedback on student assignments would allow the students to enhance their skill in natural deduction proofs which are fundamental in formal verification and artificial intelligence applications. The teachers will benefit too and can put more emphasis on the semantics. Natural deduction is taught at most if not all universities but few tools exist. Initially we plan to have former students on the course to evaluate ProofJudge and later it will be employed in the course.

*Keywords* – E-Learning, Automated Tool, Mathematical Logic, Computer Science

## I INTRODUCTION

Every year since 2006 around 70 BSc and MSc students in computer science have taken DTU course 02156 Logical Systems and Logic Programming (5 ECTS). The aim of the logical systems half of the course is to give an introduction to mathematical logic for automated reasoning. The students are expected to have a rudimentary knowledge of mathematical logic from previous courses on discrete mathematics (e.g. DTU course 01017) and/or artificial intelligence (e.g. DTU course 02180). The students are also expected to have taken introductory courses in imperative / object-oriented programming as well as in algorithms and data structures (e.g. DTU courses 02101 and 02105). This corresponds to 15-20 ECTS standard computer science bachelor courses. We will provide the most necessary background in the following section. In the remaining part of the introduction we describe the course, in particular its aim and structure.

In the autumn semester 2014 the grade average was 7.2 on the Danish grading scale, 85% of the registered students passed the course and in the anonymous but publicly available course evaluation 75% agreed or strongly agreed that they had learned at lot in the course (http://www.kurser.dtu.dk/02156). More than 80% found that their performance during the course was equivalent to the expected 9 hours per week and in the answer to the final evaluation question, “In general, I think this is a good course”, less than 6% disagreed and no student strongly disagreed. Similar good results exist for the preceding years and the course has only been adjusted a little from year to year.

The structure of the course is rather traditional:

- Textbook: *Mathematical Logic for Computer Science* (Mordechai Ben-Ari, Springer 2012). The first edition was published in 1993 and now we have the third edition.

- Exam: Mandatory individual assignments and a 2-hour written exam without computer (all written works of reference are permitted). Several sample exams with solutions are provided.

- Lessons: 13 weeks each with 2 hours of lectures followed by 2 hours of exercise classes with two teaching assistants.

Despite the rather traditional structure it seems that the students are active learners and that there is no reason to update our teaching. Nevertheless we are probably going to change more or less every aspect of the course in the coming years. Our goal is simply to teach more advanced topics while keeping it motivating and engaging for the students. It will be more fun and challeging for us to teach more advanced topics and the university and the society will benefit too.

The more advanced topics are not normally taught for BSc and MSc students. Elements have recently been considered in MSc and PhD courses at universities like TUM (Technischen Universität München), but these elements cannot be directly transferred to a course for BSc students.

No suitable textbooks are available for the more advanced topics. For more than a year we have been working on the e-learning tool NaDeA: Natural Deduction Assistant (Villadsen *et al.* 2015). It has been tested during the summer 2015 on selected BSc and MSc students and will be used in the autumn 2015 for the 70 students on the course (52 are Danish BSc students and the rest is a mix of international and Danish BSc/MSc students). Furthermore we have obtain funding for the development of ProofJudge, which is a separate component supporting student assignments and automatic feedback and/or grading, to be ready in the summer 2016. In the present paper we describe both NaDeA and ProofJudge.

NaDeA is available online (http://nadea.compute.dtu.dk/), but although there is quite some welcome information, a tutorial, exercises with solutions and a help system it requires basic skills in mathematical logic which we provide in the first 5 weeks of the course.

In the following section II we provide the background needed to understand the purpose of NaDeA and ProofJudge. In section III we elaborate on our design decisions, briefly discuss our results and conclude.

## II BACKGROUND

This section provides the most necessary background on mathematical logic and automated reasoning.

Let us start with the following quote from the large 2-volume *Handbook of Automated Reasoning* (2001):

> *Automated reasoning has matured into one of the most advanced areas of computer science. It is used in many areas of the field, including software and hardware verification, logic and functional programming, formal methods, knowledge representation, deductive databases, and artificial intelligence.*

The kind of reasoning that we are interested in here is the reasoning in mathematics and science, in particular engineering science. By automated reasoning we mean reasoning by a machine, which in practice is just a program on a standard computer.

Today's computers are fast and can perform millions of operations per second. Since computers do not understand the purpose of the operations it is essential that the operations are correct. Otherwise a lot of incorrect reasoning will be produced in no time. In mathematical logic we study the correctness of the rules for automated reasoning. A few concepts has a longer history but

In *De Arte Combinatoria* (1666) Gottfried Wilhelm Leibniz was the first to tackle effective reasoning as a technical problem. But he did not get very far. In Augustus De Morgan's *First Notions of Logic* (1839) and *Formal Logic* (1847) and George Boole's *The Mathematical Analysis of Logic* (1847) and *Laws of Thought* (1854) we find what we now call the Boolean expression involving the so-called truth values (corresponding to 0 and 1 in the modern digital computer). But the first proposal including support for also the natural and real numbers we find in *Begriffsschrift* (1879) by Gottlob Frege. Unfortunately the rules are inconsistent such that we do not only have the truth 2+2=4 but also the falsehood 2+2=3.

Bertrand Russell's *The Principles of Mathematics* (1903) consists of 500 sections and with later simplifications of the rules we reach today's foundations of mathematics in the form of first-order logic and higher-order logic. Kurt Gödel showed in 1931 that all foundations of mathematics are essentially incomplete in a technical sense. Furthermore in 1936 Alonzo Church showed that even first-order logic is essentially undecidable and when Alan Turing later the same year defined the universal computer the limitations of mathematical logic was generally accepted. However, it is important to understand that these results are theoretical limitations of mathematical logic. Selected results about the limitations are briefly discussed in the course.

We may take the axioms in Kurt Gödel's *The Consistency of the Continuum Hypothesis* (1940) as the standard foundation of mathematics. The main results were announced a few years before (Gödel 1938). The details of the standard foundation are very difficult but a few glimses are included in the course. Also the details of higher-order logic are too difficult so the present course we consider only first-order logic. Although this is sufficient from at theoretical point of view it definitely would be better from a practical point of view to consider higher-order logic as well (it is probably not a good idea to skip first-order logic and jump to higher-order logic although it would have its benefits).

The main point is that mathematical logic is a relative new discipline and very tricky because the students have to learn to reason about reasoning, which most likely also explains why it took so many year to obtain the results. But the above results up to 1940 are only about the foundations and the limitations. They do not really consider how to use mathematical logic in practice, in particular, how to use a computer to make proofs using axioms and rules. In 1954 Martin Davis programmed the first computer to make a proof, namely that the sum of two even numbers is again an even number. Today computers can make proofs for which ordinary pen and paper proofs are not available (Hales *et al.* 2015).

In the 1930s Gerhard Gentzen and Stanisław Jaśkowski independently discovered natural deduction, which is a technical term referring to proofs with a changing set of assumptions. Many textbooks use natural deduction but it can be confusing to student. We have not used it before in the course but it is of course the core of NaDeA Natural Deduction Assistant (Villadsen *et al.* 2015). Here is a sample proof as displayed by NaDeA (browser screenshot):

```
1   Boole    [ ] ∃x.A(x)→(∀y.A(y))
2   Imp_E      [(∃x.A(x)→(∀y.A(y)))→⊥] ⊥
3   Assume       [(∃x.A(x)→(∀y.A(y)))→⊥] (∃x.A(x)→(∀y.A(y)))→⊥
4   Exi_I        [(∃x.A(x)→(∀y.A(y)))→⊥] ∃x.A(x)→(∀y.A(y))
5   Imp_I          [(∃x.A(x)→(∀y.A(y)))→⊥] A(c)→(∀x.A(x))
6   Uni_I            [A(c), (∃x.A(x)→(∀y.A(y)))→⊥] ∀x.A(x)
7   Boole              [A(c), (∃x.A(x)→(∀y.A(y)))→⊥] A(c')
8   Imp_E                [A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] ⊥
9   Assume                 [A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] (∃x.A(x)→(∀y.A(y)))→⊥
10  Exi_I                  [A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] ∃x.A(x)→(∀y.A(y))
11  Imp_I                    [A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] A(c')→(∀x.A(x))
12  Boole                      [A(c'), A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] ∀x.A(x)
13  Imp_E                        [(∀x.A(x))→⊥, A(c'), A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] ⊥
14  Assume                         [(∀x.A(x))→⊥, A(c'), A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] A(c')→⊥
15  Assume                         [(∀x.A(x))→⊥, A(c'), A(c')→⊥, A(c), (∃x.A(x)→(∀y.A(y)))→⊥] A(c')
16      *
```

The formula proved is the so-called *Drinker Paradox* (Smullyan 1978). The columns to the right with the line numbers and the rule names are not required but most helpful for students. One the other hand, the way the formulas is indented is crucial. It is not always the case that formulas are placed further and further to the right going towards higher line numbers. The special language with many symbols is also important. In a way the above sample proof using mathematical logic can be compared to a mathematical calculation like 10101*10101 = 102030201; it is something which all engineers can do using pen and paper but we rather use a computer or pocket calculator for the task.

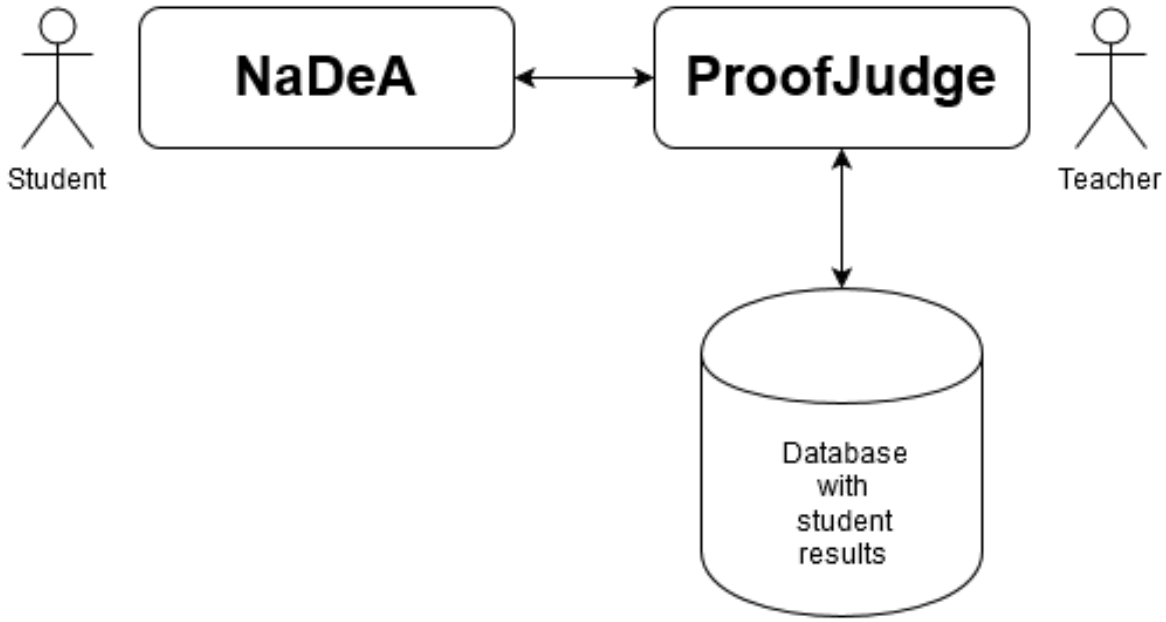
## III DESIGN DECISIONS AND CONCLUSIONS

The course should prepare students for the use of proof assistants and NaDeA is an important step in this direction. The relevant tasks are as follows (Adams 2014): (1) the proof script must be executed to see that it produces the claimed formula as the final formula, (2) the definitions must be examined to see that the meaning of the final formula agrees with the common understanding, and (3) the proof assistant must be audited to make sure there is no foul play.

NaDeA (Natural Deduction Assistant) is a new tool for teaching logic based on natural deduction and with a formalization in the proof assistant Isabelle such that the usual informal descriptions can be avoided (Villadsen *et al.* 2015). ProofJudge is a separate component supporting student assignments and automatic feedback and/or grading via a database with student results.

Both tools work in a browser without any software installation and is open source software. It is expected to make the current course textbook optional (Ben-Ari 2012). Like for programming it is important to practice the syntax (Moth *et al.* 2011). Additionally NaDeA functions as a relatively gentle introduction to Isabelle which allows for interactive machine-checked proof and has the potential to fundamentally change how we build and trust critical software (Klein 2015).

We can illustrate the relationships between the student, NaDeA, ProofJudge and the teacher as follows:




We believe that automatic feedback on student assignments is going to be important for the motivation of the students and will free teaching assistant resources for more feedback on conceptual problems.

We find that the following requirements constitute the key ideals for any proof assistant. It should be:

– Easy to use.

– Clear and explicit in every detail of the proof.

– Based on a formalization that can be proved at least sound, but preferably also complete.

Based on this, we saw an opportunity to develop NaDeA which offers help for new users, but also serves to present an approach that is relevant to the advanced users.

## ACKNOWLEDGEMENTS

Thanks to Alexander Birch Jensen and Anders Schlichtkrull for help with the development of NaDeA.

## REFERENCES

Adams, M. (2014). Flyspecking Flyspeck. Pages 16-20 in *Lecture Notes in Computer Science*, Vol. 8592, Springer.

Ben-Ari, M (2012). *Mathematical Logic for Computer Science (Third edition)*. Springer.

Gödel, K. (1938), The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis. Pages 556-557 in *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 24.

Hales, T. *et al.* A Formal Proof of the Kepler Conjecture. *arXiv* 1501.02155.

Klein, G. (2015). *ProofCraft – The Craft, Art, and Science of Interactive Machine-Checked Proof.* http://proofcraft.org/index.html#about

Moth, A., Villadsen, J & Ben-Ari, M (2011). SyntaxTrain: Relieving the Pain of Learning Syntax. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Darmstadt, Germany.

Smullyan, R. (1978). *What is the Name of this Book? The Riddle of Dracula and Other Logical Puzzles.* Prentice Hall.

Villadsen, J., Jensen, A. B. & Schlichtkrull, A. (2015). NaDeA: A Natural Deduction Assistant with a Formalization in Isabelle. Pages 253-262 in *Proceedings of 4th International Conference on Tools for Teaching Logic*, 9-12 June 2015, Rennes, France. Also *arXiv* 1507.04002.

## BIOGRAPHICAL INFORMATION

Jørgen Villadsen is associate professor at DTU Compute (Algorithms, Logic and Graphs Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark). Since 2007 Jørgen Villadsen has been programme director for the MSc in Computer Science and Engineering programme.

## APPENDIX

We illustrate the central ideas using a simplified version of a small example from the DTU course 02156 Logical Systems and Logic Programming. A similar example is briefly discussed on the first page of *Logic: Techniques of Formal Reasoning* by Donald Kalish, Richard Montague and Gary Mar (Oxford University Press 1980). The following slides are from the course and use one of the proof systems in the course textbook. Currently the students do not use a tool for automated proof judging.

Consider the following argument in natural language:

If Joe studies then he passes the exam.

If Joe does not study then he has a good time.

If Joe does not pass the exam then he does not have a good time.

Therefore Joe passes the exam.

Consider the corresponding formulas in propositional logic (propositional calculus):

$s \to p$

$\neg s \to t$

$\neg p \to \neg t$

$p$

$s$ corresponds to "studies", $p$ corresponds to "passes the exam", and $t$ corresponds to "has a good time"...

$((s \to p) \wedge ((\neg s \to t) \wedge (\neg p \to \neg t))) \to p$

Proof:

| | | |
|---|---|---|
| 1. | $\vdash t, \neg t, \neg(s \to p), p$ | Axiom |
| 2. | $\vdash \neg\neg t, \neg t, \neg(s \to p), p$ | $\alpha\neg$,1 |
| 3. | $\vdash \neg p, \neg t, \neg(s \to p), p$ | Axiom |
| 4. | $\vdash \neg t, \neg(\neg p \to \neg t), \neg(s \to p), p$ | $\beta\to$,3,2 |
| 5. | $\vdash \neg p, t, \neg s, p$ | Axiom |
| 6. | $\vdash s, t, \neg s, p$ | Axiom |
| 7. | $\vdash t, \neg s, \neg(s \to p), p$ | $\beta\to$,6,5 |
| 8. | $\vdash \neg\neg t, \neg s, \neg(s \to p), p$ | $\alpha\neg$,7 |
| 9. | $\vdash \neg p, \neg s, \neg(s \to p), p$ | Axiom |
| 10. | $\vdash \neg s, \neg(\neg p \to \neg t), \neg(s \to p), p$ | $\beta\to$,9,8 |
| 11. | $\vdash \neg(\neg s \to t), \neg(\neg p \to \neg t), \neg(s \to p), p$ | $\beta\to$,10,4 |
| 12. | $\vdash \neg(s \to p), \neg((\neg s \to t) \wedge (\neg p \to \neg t)), p$ | $\alpha\wedge$,11 |
| 13. | $\vdash \neg((s \to p) \wedge ((\neg s \to t) \wedge (\neg p \to \neg t))), p$ | $\alpha\wedge$,12 |
| 14. | $\vdash ((s \to p) \wedge ((\neg s \to t) \wedge (\neg p \to \neg t))) \to p$ | $\alpha\to$,13 |

We provide a simplified version of the example. We consider the following logical formulas for the natural language sentences where the minus sign is used for the "not" construction and the arrow is used for the "if … then" construction:

| | |
|---|---|
| S → P | If Joe studies then he passes the exam. |
| -S → T | If Joe does not study then he has a good time. |
| -P → -T | If Joe does not pass the exam then he does not have a good time. |
| P | Joe passes the exam. |

The goal is to show that the final sentence follows logically from the 3 conditional sentences above. This must be shown without appealing to intuition because we want to automate the reasoning.

Intuitively we can reason as follows. Assume -P (we will show that this is impossible, so therefore P). Remember that -P means "not" P. Then from S → P we have -S because otherwise (that is S) we would have P (and we already assumed -P). And then from -S and -S → T we have T. But from -P and -P → -T we have -T. So both T and -T, which is impossible, hence we do not have -P, and therefore we have P.

This intuitive reasoning is not immediately suitable for automation. A format more suitable for automation is the following formal proof:

| | |
|---|---|
| 1.1 | -(S → P) \| -(-S → T) \| -(-P → -T) \| P |
| 2.1 | S \| -(-S → T) \| -(-P → -T) \| P |
| 3.1 | S \| -S \| -(-P → -T) \| P |
| 3.2 | S \| -T \| -(-P → -T) \| P |
| 4.1 | S \| -T \| -P \| P |
| 4.2 | S \| -T \| T \| P |
| 2.2 | -P \| -(-S → T) \| -(-P → -T) \| P |

The line numbers 1.1, 2.1, 2.2, … are not part of the proof as such. The bar | is used to separate alternatives. Hence line 1.1 states that either one of the 3 conditional sentences must be false (note the added minus signs) or the conclusion (P) must be true. In lines 2.1 and 2.2 the requirements for the falsehood of S → P are examined, namely that S is true (2.1) and P is false, that is, -P is true (2.2). Analogously for lines 3.1 and 3.2 (considering 2.1) and for lines 4.1 and 4.2 (considering 3.2). In all relevant cases (2.2, 3.1, 4.1 and 4.2) we find a tautology (like P or -P for 2.2):

| | |
|---|---|
| 2.2 | -P \| P |
| 3.1 | S \| -S |
| 4.1 | -P \| P |
| 4.2 | -T \| T |

So the formal proof is correct and appropriate feedback can be provided to the student by ProofJudge.