NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DIOGO DA SILVA CAGICA CARVALHO

ONLINE INTERACTIVE TOOL FOR LEARNING LOGICAL PROOF SYSTEMS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

ONLINE INTERACTIVE TOOL FOR LEARNING LOGICAL PROOF SYSTEMS

DIOGO DA SILVA CAGICA CARVALHO

Adviser: Ricardo Gonçalves
Assistant Professor, NOVA University Lisbon

Co-adviser: João Costa Seco
Associate Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2023

Abstract

For many areas, including mathematics and computer science, logic is a basic but important topic. In particular, the notions of natural deduction and proof systems for natural deduction are of fundamental interest.

Students learning logic benefit from the use of interactive, visual tools where they can solve exercises and receive feedback and hints from their attempts. However, most of these tools are based on installable programs, or static, making it hard to expand them with additional learning material and exercises. The few available online tools and courses focusing on logic also tend to skip the topic of natural deduction.

The goal of this thesis is to leverage the work done for these tools and extend it to provide practice for natural deduction systems, both for propositional and first-order logic. To achieve this, an online tool that teaches the concepts of natural deduction is proposed, hosting an interactive, visual proof assistant which hosts exercises common in in-person classes of logic courses, alongside the capability of providing feedback and hints to help students learn and advance at their own pace.

To achieve this, the implementation of some basic logic algorithms and connections to external reasoners is necessary, as well as mechanisms to provide feedback and control how it's provided. Additionally, the capability for teachers to create exercises as they see fit, and to be able to store and load these exercises must also be implemented.

Finally, it's intended to leverage its usability for automatic evaluation purposes, by incorporating the system, if possible, with existing online e-learning platforms, such as Moodle.

Keywords: Logic, Proof System, Natural Deduction, Online, Interactive, Propositional Logic, First-order Logic

Resumo

Em várias areas, incluindo matemática e informática, logica é um topico basico, mas important. Em particular, as noções de dedução natural e sistemas dedutivos de dedução natural são de interesse fundamental.

Alunos que estejam a aprender lógica beneficiam do uso de ferramentos interativas e visuais, onde conseguem resolver exercises e receber feedback e dicas durante a atividade. No entanto, maior parte destas ferramentos são baseadas em programas instaláveis ou estáticos, tornardo dificil a tarefa de expandi-los com material de aprendizagem e exercicios adicionais. Ferramentas e cursos que estejam disponveis online e que sejam focados em logica também tendem a saltar o topico de dedução natural.

O objetivo desta tese é aproveitar o trabalho feito para estas ferramentas e cursos e extende-lo para oferecer ensino e pratica de sistemas de dedução natural, tanto para lógica proposicional como para lógica de primeira-ordem. Para alcançar isto, uma ferramenta online que ensine os conceitos de dedução natural é proposta, fornecendo um assistente de prova visual e interativo, completo com exercises comuns em aulas presenciais em cursos de lógica e capacidade de fornecer feedback e dicas que ajudem alunos a aprender e avançar ao seu proprio ritmo.

Para isto, a implementação de alguns algoritmos de logica basico e coneções com raciocionadores lógicos externos é necessario, bem como mecanismos para fornecer e controlar a maneira de fornecimento de feedback. Adicionalmente, a capacidade para professores conseguirem criar exercises como achem que é adequado, bem como a capacidade de os guardar e carregar também terá de ser implementado.

Por fim, existe a intenção de aproveitar a usabilidade da ferramente para propositos de avaliação automática, ao incorporar o sistema, se possivel, com plataforms online e-learning existentes, como o Moodle.

Palavras-chave: Logica, Sistema Dedutivo, Dedução Natural, Online, Interativo, Logica Proposicional, Logica de Primeira-Ordem

Contents

List of Figures

List of Tables

Acronyms

JNLP Java Network Launch Protocol *(p. 24)*

LEM Law of Excluded Middle *(pp. 5, 6)*

MOOC Massive Open Online Course *(pp. vii, 1, 15, 27, 28)*

1 Introduction

1.1 Context

Logic plays a fundamental role in mathematics and computer science. Mathematical logic is used in syntax specification, and logical reasoning plays an important part in, for example, the design and analysis of algorithms, the development of verification tools, and others. As such, computational logic is an important topic to learn when studying computer science, and most universities that offer computer science degrees will include a course that teaches it in the curriculum of the degree.

In recent times, there has been an increase in both supply and demand for online teaching, even before the COVID-19 pandemic. The popularity of Massive Open Online Courses or MOOCs, for short, has been ramping up, with MOOCs such as EdX, Udacity and Coursera offering a large variety of courses as early as 2015 [11]. MOOCs have shown that there are many people interested in learning but either cannot or do not want to attend in-person classes. Through online teaching, these people can follow courses in areas of their choice from anywhere in the world. Technically, they can offer everything an in-person class gives, from learning material to exercises, homework, tests, and help through contact with teachers and feedback given with exercises. Some also have discussion forums where students can trade ideas with each other. However, only students with more expertise in subjects tend to use them, with the others remaining passive. [16]. The idea of student autonomy is very prevalent in MOOCs, from the way students can pace themselves to how much effort they apply to the course and more. While autonomy is often desirable, the number of people who enroll in these far outnumber those who complete them, even though the number of people who do complete them is still significant [5].

With the recent successes of online teaching, a closer look must be given at how it can be used to help teach logic. Online teaching tools can take complementary roles to in-person classes. For example, teachers can utilize online teaching tools to provide users with exercises and feedback, complementing the materials taught in class. This can help the students progress at the own pace, which can be hard during in-person practical classes for example, which have a limited schedule and many students at the same time.

Back in 2016 [12], the number of MOOCs tackling logic was very small, nearing none. Nowadays, the number remains small, but MOOC focusing mainly on logic has appeared, in the form of Iltis [9, 8]. Even so, Iltis does not cover all aspects of logic, including a very important one, natural deduction.

Natural deduction is a method of formal logical reasoning that is based on inferences and is an important part of computational logic. It is based on proof systems and inference rules that change the system's state. For new students, it is often the most difficult part of the logic course and poses a wall that can only be climbed through much study and practice. To practice this, students are posed with exercises where they must correctly build a proof from a given set of premises and a conclusion, or complete a partially built one. The most common way to do these exercises, other than by hand, is by utilizing proof assistants, which are tools available on computers where students can write down their proof and have the assistant check the validity of it. If or when the result of the assistant comes back as invalid, the student then attempts to redo it and, if they get stuck, asks for help from a professor or another student. However, help is not always available, especially when the student isn't in a practical class.

The goal of this thesis is the development of an online tool that can help students better learn and understand the concepts of natural deduction autonomously. The next section will introduce the requirements and explain what problems arise from them.

1.2 Problem

Having seen the potential that online tools have for teaching, a search was conducted to find a tool to complement an in-person logic course, namely the topic of natural deduction. As previously said, this tool is expected to come in the form of a proof assistant, and while there are many of those available, we wanted one that could fulfill certain criteria, which are:

- Capable of visually representing proofs in a Gentzen Tree format.
- Support for propositional classical logic and, if possible, first-order classical logic.
- The logical reasoning must be sound and complete.
- Capable of creating and loading exercises.
- Capable of providing feedback and/or hints to the user.
- Online, to later connect it with online platforms.
- If possible, the capability of collecting data for certain metrics.

While there are many visual proof assistants available online or as an installable, it is difficult to find one that meets all of these requirements. The results of the search lead us

to many assistants, but even the most robust ones feature-wise end up missing some of the criteria. This search leads us then to conclude that no such tool exists currently, and as such, it must be developed instead. The features that seem to be missing most often are the visual representation in Gentzen Tree format, the capability of creating exercises and the capability of providing feedback, and as such, a closer look must be given to how different assistants provide these criteria.

Many proof assistants represent their proofs in the Fitch-style proof format as it seems to be the most popular format. Unfortunately, the Fitch-style format and the Gentzen tree-style format are very different in regards to visuals and have some characteristics that are unique to them and not present in the other, such as marks in Gentzen trees and subproofs in Fitch. Proof assistants that don't follow the Gentzen tree format would require a full transformation which can be very hard to do. The way the assistants program their logical reasoning is also of some importance for this topic. Assistants rely on one of two different options, those being writing and programming the reasoning from scratch and utilizing high-end proof assistants such as Coq and Isabelle/HOL to do the verification. Both of these options need to be considered for the development of a tool, as they also influence the choice of programming language for the development of the tool. For the second option, for example, the programming language would need to be capable of communicating with an API that in turn communicates with the high-end assistant.

Exercise creation is mostly done by saving the proof in a viable format that allows it to later be reloaded in the same state that it was left in previously. This would allow for the existence of the different types of exercises mentioned prior. As for the format, it varied from tool to tool. Some tools import the proof into the LaTeXformat, but they cannot reload the proof this way. While the Iltis MOOC doesn't offer natural deduction, it generates its exercises in its specific format, which is then stored in an XML file. The same is done by another proof assistant, namely, Panda [7], which indicates that it is possible to create a format for an exercise that is then saved in an XML file. For this criteria, the main concerns are the format in which to save the exercise, and where to store them.

Lastly, the feedback and hints. It seems most proof assistants have this criterion as a second thought, with the feedback offered by many ranging from none to minimal. Still, there are a few who provide it, and notes can be taken from them. Some assistants show which parts of the proof are valid and which aren't. Assistants that require the proof to be written via textual input show feedback when something is incorrectly typed, ranging from pointing errors out to explaining what was expected to be written. Some provide hints on how to start the proof or how to continue in case the student gets stuck. An interesting way of providing feedback is the way the Iltis MOOC does it. Feedback is generated and provided to users based on a set of rules. The users can also control the amount of feedback they receive. All of these and more must be considered, as the quality of feedback given, as well as how it is given, is vital to the usefulness of this tool for learning. Not giving enough feedback can cause users to get stuck while giving too much feedback can lead to users defaulting to it as soon as they face any difficulties, which

hinders the learning process of a student.

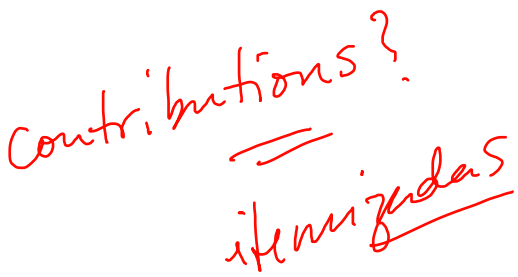
1.3 Document Structure

There are 3 more chapters in this document. They are divided into chapters 2, 3 and 4.

Chapter 2 covers background information that serves to explain and contextualize some of the concepts that have been used throughout the document. This includes propositional and predicate logic, natural deduction, proof assistants and others.

Chapter 3 covers related work, describing some of the visual proof assistants that have been studied and analyzed, as well as the previously mentioned Iltis MOOC. At the end of this chapter, there is an overview section that measures these to some of the criteria proposed earlier in chapter 1.

Finally, Chapter 4 concludes the document by discussing a proposed solution and work plan.



2

Background

This section will introduce and describe some concepts which could be of use to someone without knowledge of logic. It introduces classical and intuitionistic logic, distinguishing them. It is then followed by a short explanation of propositional and predicate classical logic, alongside their proof rules, a description of natural deduction, including its two most prominent proof formats as well as an introduction to the concepts of soundness and completeness. Finally, it introduces the concept of proof assistants in addition to introducing Coq and Isabelle/HOL, two of the most widely used non-visual proof assistants.

2.1 Classical & Intuitionistic Logic

Classical logic, also known as non-constructive logic, is a type of logic in which every formula is always assumed to be either true or false, but not both at the same time. Classical logic distinguishes itself from intuitionistic by including the Law of Excluded Middle (LEM) [19], which describes the fact that a formula is either true or false, even if its actual value is unknown, and tends to be represented as such:

$$A \vee (\neg A)$$

example ?

From this law, other important rules and characteristics can be derived:

- Negation ($\neg$) acts as a toggle between true and false.
- Double negation ($\neg\neg$) elimination theorem.
- Reductio ad Absurdum theorem, also known as proof by absurdity ($\bot$).

Intuitionistic logic, also known as constructive logic, is another type of logic that follows intuitionistic mathematics and doesn't assign formulas with values of true or false. Instead, formulas are either provable or disprovable, as intuitionists believe that the verification of formulas holds much importance. For example, a formula can only be

asserted as true when it has been verified as such [19]. This follows constructive reasoning, which seeks a definitive, explicit answer to questions.

Unlike classical logic, intuitionistic logic rejects the LEM, as well as the rules and characteristics that can be derived from it, which makes some proofs that would otherwise hold for classical logic invalid.

2.2 Classical Logic Branches

2.2.1 Propositional Logic

Propositional logic is a branch of logic that focuses on propositions and the relations between these propositions. These propositions are atomic formulas, such as "I am alive"or "She is sleeping", which can only be true or false. Their content is irrelevant, and as such, they are denoted only by capital letters. [10] We can relate propositions via the use of connectors and then test or prove the resulting formula for certain characteristics such as Validity or Satisfiability. One way to test these formulas is by natural deduction, which, as explained above, contains proof rules. The most common connectors used in propositional logic are:

- And or Conjunction ($\wedge$).
- Or or Disjunction ($\vee$).
- Not or Negation ($\neg$).
- Implies or Implication ($\Longrightarrow$).
- Equivalence or Biconditional ($\leftrightarrow$).

The rules have properties such as arity, which counts the number of formulas required to apply the rule, and mark elimination, or hypothesis elimination, which means the rule removes a mark from one of the formulas, which closes that formula. With some context given, here are the proof rules for propositional logic:

Acronym	Rule	Arity	Eliminates Marks
$\wedge$I	Introduction of Conjunction	1	
$\wedge$E	Elimination of Conjunction	1	
$\vee$I	Introduction of Disjunction	1	
$\vee$E	Elimination of Disjunction	3	✓
$\Longrightarrow$ I	Introduction of Implication	1	✓
$\Longrightarrow$ E	Elimination of Implication	2	
$\leftrightarrow$I	Introduction of Biconditional	2	
$\leftrightarrow$E	Elimination of Biconditional	2	
$\neg$I	Introduction of Negation	2	✓
$\neg$E	Elimination of Negation	2	✓

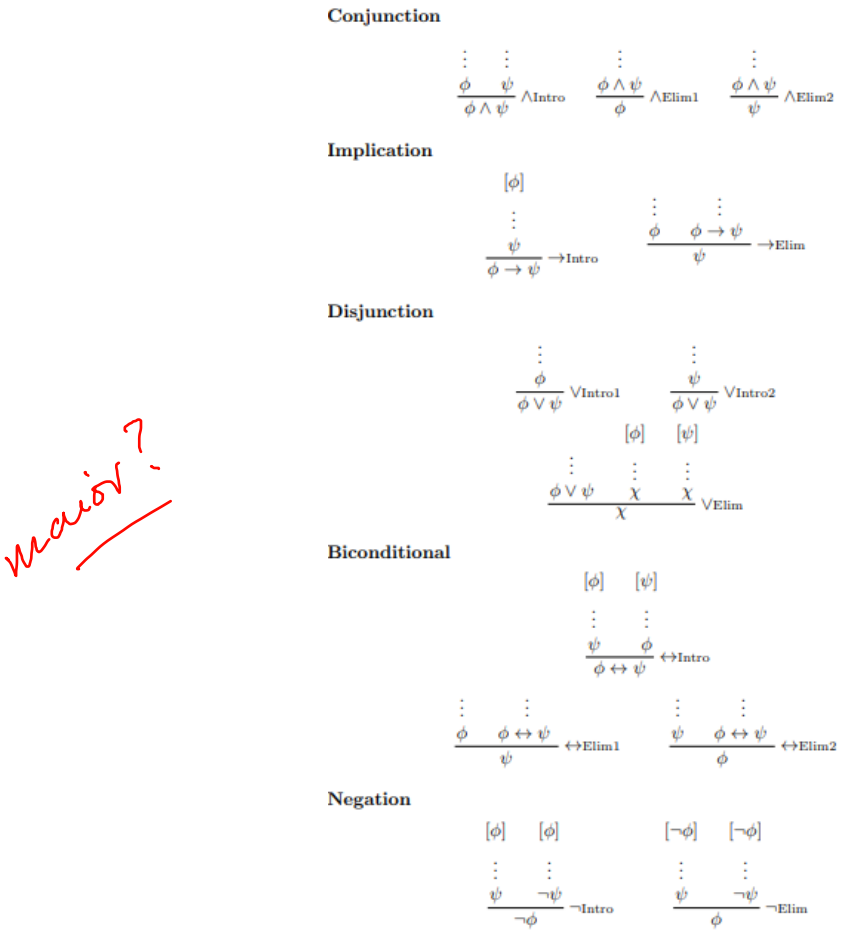


Figure 2.1: Proof rules for propositional logic. Taken from the NDPack by Alastair Carr [25].

It should be noted that instead of introducing and eliminating negation, the concept of contradiction or absurdity ($\perp$) can be used. Just like them, it will eliminate a mark.

Figure 2.1 shows a graphical representation in the tree-style format of the propositional logic proof rules.

2.2.2 Predicate or First-Order Logic

Predicate or first-order logic is an extension of propositional logic that deals with predicates, expressions that can only return true or false and have one or more variables, which in turn can be assigned values or quantified via quantifiers. First-order logic counts with various symbols, which can be grouped into five categories:

- Quantifiers, which are used to give a notion of quantity. In this case, we have $\exists$, meaning "there exists at least one"and $\forall$, meaning "for all".

- Variables, which represent arbitrary elements of a set and are represented by lower-case letters from the end of the alphabet such as x or y.

- Constants, which represent a specific element of a set and are represented by lowercase letters from the beginning of the alphabet, such as a or b.
- Functions, which take some variables and produce an output. They are typically represented by the lowercase letters f, g or h.
- Relations, which denote relations between variables. They are represented by capital letters.

By combining these symbols we can make more complex expressions. For example, the expression "All cats are sneaky", which cannot be expressed under propositional logic, can now be expressed as $\forall$x (cats(x) $\implies$ sneaky(x)).

This branch of logic retains the previously shown connectors and proof rules from propositional logic but adds to it four more rules, two for each quantifier, which are:

Acronym	Rule	Arity	Eliminates Marks
$\exists$I	Introduction of Existential	1	
$\exists$E	Elimination of Existential	2	✓
$\forall$I	Introduction of Universal	1	
$\forall$I	Elimination of Universal	1	

The concept of substitution is also important for first-order logic natural deduction. As can be seen in Figure 2.2, which shows a graphical representation of the first-order proof rules, the application of the elimination rules substitutes quantified formulas for non-quantified formulas by assuming a constant in place of the quantifier. Likewise, the application of introduction rules does the opposite, turning non-quantified formulas into quantified formulas. As such, applying these rules has the added challenge of knowing which constants to substitute in or out.

$$\frac{A(x)}{\forall y\ A(y)}\ \forall\text{I} \qquad \frac{\forall x\ A(x)}{A(t)}\ \forall\text{E} \qquad \frac{A(t)}{\exists x\ A(x)}\ \exists\text{I} \qquad \frac{\exists x\ A(x) \qquad \begin{array}{c}\overline{A(y)}^{\,1}\\ \vdots \\ B\end{array}}{B}\ 1\ \exists\text{E}$$

Figure 2.2: Proof rules for first-order logic. [24]

2.3 Natural Deduction

Natural deduction is a type of logical system that was initially proposed by Gerhard Gentzen and Stanislaw Jakowski in the year 1934 [21]. It is a method that uses proof systems, which are composed of inference/proof rules that are applied to formulas, to prove or derive new formulas with the intent to prove the validity of a goal. These rules and their application are designed in a way that tries to behave how a person normally

reasons and draws conclusions. Different natural deduction systems can differ in aspects such as proof format, the approach used to construct the proof and even which rules they use. Proofs in natural deduction can be built from the bottom up (backward reasoning) or from the top down (forward reasoning). Gentzen Tree and Fitch style proofs are two natural deduction proof formats that are widely used in logic teaching and proof assistants.

2.3.1 Proof Formats

2.3.1.1 Gentzen Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\psi_1\,^1 \quad \psi_1 \to (\psi_2 \wedge \psi_3)\,^2}{\psi_2 \wedge \psi_3}\to E}{\psi_2}\wedge E_d}{\psi_1 \to \psi_2}\to I,1 \qquad \dfrac{\dfrac{\dfrac{\psi_1\,^3 \quad \psi_1 \to (\psi_2 \wedge \psi_3)\,^4}{\psi_2 \wedge \psi_3}\to E}{\psi_3}\wedge E_e}{\psi_1 \to \psi_3}\to I,3}{(\psi_1 \to \psi_2) \wedge (\psi_1 \to \psi_3)}\wedge I$$

Figure 2.3: An example of a tree style proof system.

In tree-style proof systems, the root of the tree is the formula to be proved (the goal), and its leaves are the premises, with the tree growing upwards. Each "branch"of the tree represents a different subproof. Applying rules to formulas in a branch of the tree will grow the branch upwards and generate new subgoals, separating the new subgoals from the existing formula with a line that shows the rule and marks used. Hypotheses can be assumed when needed, as long as they are properly marked.

Marks are a way to keep track of a formula by assigning them some sort of identification, most commonly numbers. All marked formulas are either open (undischarged) or closed (discharged), depending on whether the mark has been eliminated by a proof rule or not, with only certain rules being able to eliminate proof rules, which will be shown in the next section. A proof can only be considered complete when all of its non-premise formulas have been closed.

An example of a tree proof can be seen in Figure 2.3. The goal of this proof is (ψ1 $\implies$ ψ2) $\wedge$ (ψ1 $\implies$ ψ3) and there's a premise that is ψ1 $\implies$ (ψ2 $\wedge$ ψ3). All instances of the premise in the proof are marked and will remain marked as the premise cannot be closed. At some point in the proof, ψ1 is assumed in both branches of the tree and properly marked with different numbers so as not to conflict. As these are assumptions and not premises, they must eventually be closed, which they are when the rule $\implies$ I, or introduction of implication, is applied, referencing the mark for the assumption. This eliminates the mark and closes the hypothesis.

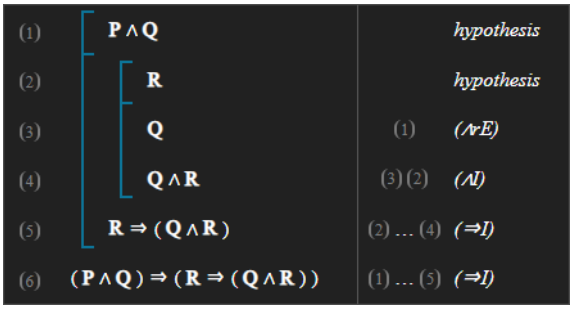


Figure 2.4: An example of a Fitch style proof system.

2.3.1.2 Fitch

Fitch-style proof systems follow a linear representation, with each line containing a formula and either the rule (and necessary formulas for it to be valid) from which it was inferred or, in the case of assumptions, an indication that this is the case. It differs from Gentzen trees in the following aspects:

- No marks to represent assumptions.
- Formulas can be inferred from a single assumption.
- To apply a proof rule there must be a reference to all lines or subproofs needed to make it valid.

The lack of marks leaves the system with problems concerning keeping up with and representing open and closed formulas. To solve this issue, the Fitch system utilizes subproofs, in which a new box is opened in a proof every time an assumption is made. The assumption is open while inside the scope of the box and is closed on all lines that come after the final line of the subproof. Subproofs can also contain different subproofs within themselves, which have their assumptions and scope. If the assumption of an outer subproof must be used on an inner subproof, a reiteration rule can be called, referring to the formula that is being repeated, though most Fitch-style proof systems simply refer to the original assumption if it's needed to be used in the application of a proof rule.

While in Gentzen trees there's sometimes a need to assume the same formula in different branches, for example assuming a conjunction twice so we can get both of its sides, Fitch-style systems only need a single assumption, which can be referenced at any time and any number of times as long as it is done inside the scope of its subproof.

An example of a Fitch-style proof is shown in Figure 2.4, which was taken from the Edukera [15] tool. This proof only has a conclusion, which is in line 6, and no premises. It has two subproofs, one inside the other, each with its assumption, which is lines 1 and 2, respectively. The application of proof rules references the lines for the formulas that make them valid, with lines 5 and 6 being justified by entire subproofs.

2.3.2 Soundness & Completeness

Soundness and completeness are properties of a deductive system. Due to the nature of these properties, we can use them to measure how strong the deductive system is, as the lack of these properties in a system will diminish its worth as deductions from it cannot be fully trusted. As a deductive system, natural deduction is considered to be both sound and complete. [2]

For a system to be sound, it must hold the following property:

If a formula G can be derived from a set of formulas F, then G is a consequence of F. [10]

The completeness of a system regards the opposite of soundness:

If a formula G is a consequence of a set of formulas F, then G can be derived from F.

2.4 Proof Assistants

Proof assistants are tools that serve to assist in various ways with deductive proofs. Most assistants for first-order logic exist for teaching and assisting students who are learning logic. Several of these are shown in the next chapter, Related Work, where their features are shown and compared. There are other assistants that, while still capable of assisting in teaching, are more geared towards proving mathematical theorems and other problems of the sort. This section will show two of the most widely used proof assistants that fit into the latter: Coq and Isabelle/HOL.

These two assistants are generally more complex and complete than others, characteristics which come with the added need of learning their syntax in addition to knowing logic and mathematical concepts, as they are capable of much more than just first-order logic.

Coq and Isabelle/HOL share several characteristics, which, according to Bartzia [3] and Yushkovskiy [18], are:

- Open source and still actively maintained and developed.
- Proofs are built via textual input.
- Offer little feedback, with just small messages whenever a rule does not apply or when an expression isn't typed correctly.
- Only show the current subgoals, never showing the full proof.
- Allow navigation of the steps of the proof to see what the state of the proof was directly after application of the step.
- Both have been used to formalize and prove many mathematical theorems.
- A large number of libraries filled with already proven lemmas.

- Can function as a functional programming language.
- Support forward reasoning.
- While Yushkovskiy only mentions Isabelle supporting backward reasoning, both support it [28].

While they share these characteristics, these assistants have major differences regarding other aspects.

2.4.1 Coq

Coq has been developed since 1984 by INRIA and was first released in 1989. It is based on the Calculus of Inductive Constructions [18] and as such, was implemented with intuitionistic (constructive) logic in mind. It utilizes a command-based language, with the commands being called tactics. These tactics, which follow backward reasoning, are applied to proofs, changing their state and can, in a way, be seen as similar to inference rules. Coq logic can be extended to classical logic by adding the Law of Excluded Middle to it via libraries. Due to its command-based language, proofs end up taking forms that usually don't resemble how they would be done by hand. It does, however, attempt to ease this difference by identifying and showing variables and hypotheses in the scope of the proof at each of its steps.

In Figure 2.5, we can see a complete proof done in a Gentzen tree-style format and a possible way of writing it in Coq's online IDE jsCoq [22].

$$\dfrac{\neg P \vee Q \qquad \dfrac{\dfrac{[P]^2 \quad [\neg P]^1}{Q}\,\neg\text{Elim} \qquad [Q]^1}{Q}\,\vee\text{Elim}^1}{P \to Q}\,\to\text{Intro}^2$$

```
Theorem Core5 : ∀ P Q : Prop,
  ~P ∨ Q → P → Q.
 Proof.
   intros.
   unfold not in H.
   destruct H as [H1 | H2].
   apply H1 in H0.
   exfalso. apply H0. apply H2.
Qed.
```

Figure 2.5: The core 5 proof from NDPack [25] and a possible way to write it in Coq.

2.4.2 Isabelle/HOL

Isabelle/HOL was developed by Lawrence Paulson and Tobias Nipkow and was initially released in 1986. It is built around classical higher-order logic, an extension of predicate logic with additional quantifiers and semantics. In Isabelle proofs are built by applying proof rules and tactics, not to be confused with Coq's tactics as these are closer to functions. The application is done utilizing the apply command. The proof rules are separated into introduction and elimination rules, being applied with the keywords "rule"and "erule",

respectively. They also differ in the way they are applied to the proof, with introduction rules being applied to the current subgoal's conclusion, and the elimination rules being applied to the hypotheses of the subgoal. As the proof is built, the assistant will generate the necessary assumptions, which the user can apply by utilizing the assumption tactic. While Isabelle also shows all the hypotheses of a subgoal, it doesn't explicitly identify them as Coq does. It does, however, closely resemble a textual version of a proof done by hand, as the application of proof rules is explicitly shown in a way that closely resembles the usual proofs.

Figure 2.6 shows a complete proof and a possible way of writing it in Isabelle's downloadable IDE.

$$\dfrac{\dfrac{[P\vee Q] \quad \dfrac{\dfrac{\dfrac{(P\to Q)\wedge(Q\to P)}{P\to Q}\,\wedge\text{Elim} \quad [P]}{Q}\,\to\text{Elim} \quad [P]}{P\wedge Q}\,\wedge\text{Intro} \quad \dfrac{\dfrac{\dfrac{(P\to Q)\wedge(Q\to P)}{Q\to P}\,\wedge\text{Elim} \quad [Q]}{P}\,\to\text{Elim} \quad [Q]}{P\wedge Q}\,\wedge\text{Intro}}{P\wedge Q}\,\vee\text{Elim}}{(P\vee Q)\to(P\wedge Q)}\,\to\text{Intro}$$

```
lemma "(P ⟶ Q) ∧ (Q ⟶ P) ⟹ (P ∨ Q) ⟶ (P ∧ Q)"
  apply (rule impI)
  apply (erule disjE)

   apply (rule conjI)
    apply  assumption
   apply (erule conjE)
   apply (rule mp)
    apply assumption
   apply assumption

  apply (rule conjI)
   defer
   apply assumption
  apply (erule conjE)
  apply (rule mp)
   apply assumption
  apply assumption

  done
end
```

Como se prodem automatizar?

Figure 2.6: The disjunction 13 proof from NDPack [25] and a possible way to write it in Isabelle.

3

Related Work

This chapter will present some tools which are related to this thesis' goal. It is subdivided into three sections which are MOOC, in which the Iltis logic MOOC previously mentioned in chapter 1 is presented, Visual Proof Assistants which shows many different proof assistants which work with propositional and predicate logic only and differ from previously mentioned proof assistants by being capable of showing a visual representation the proof in the format used by the tool. Lastly, there is an Overview section at the end which compares the tools according to certain metrics such as format, exercise creation and feedback.

3.1 MOOC

3.1.1 Iltis

Iltis [8, 9] is a type of MOOC developed by the University of Dortmund for teaching logic, which was very rare even just a few years ago [12]. Currently, it features learning material and exercises for Propositional and Modal logic, as well as some First-order logic. It is available for free to anyone who wants to use it.

Iltis' main features are its "composite task model" and "Sample feedback strategies and generators" according to some of the project's main contributors [9]. Regarding its composite task mode, Iltis makes up its several exercises by making several smaller tasks, which then are composed with its other. These tasks take an input and produce an output, which can then be used as the input for the following tasks. According to its contributors, Iltis allows teachers to create exercises using this task model, by specifying the various tasks in an XML file, as shown in Figure 3.1. According to its contributors [8], Iltis also supports anonymous data collection. This comes in the form of loggers that come with tasks that collect data that can be used for various purposes.

Also specified in this XML file are the feedback generators and their rules for the tasks. Iltis features a feedback system in which each task is assigned a feedback strategy and several feedback generators, each one producing different kinds of feedback according to the rules given to them. This feedback is then displayed to the student in a predetermined

```
<!-- State formulas for the statements -->
<Task type="CreateFormulas" feedbackLevels="0,1,2"
      assimilationGenerator="syntaxServer">
  <Input>VARIABLES</Input>
  <Title>Step 2: Modelling the statements</Title>
  <Description>
    Devise a formula for each observation! Use the propositional...
  </Description>
  <Formula>
    <Description>
      If the database is faulty then so is the back end.
    </Description>
    <Solution>D → B</Solution>
  </Formula>
  <Formula>
    <Description>
      The back end is only faulty if both the database and the user...
    </Description>
    <Solution>B → (D ∧ U)</Solution>
  </Formula>
  <Formula>
    <Description>Not all three components are faulty.</Description>
    <Solution>¬(B ∧ D ∧ U)</Solution>
  </Formula>
  <!-- Feedback generator -->
  <FeedbackGenerator>
    <Feedback type="VariableNames">
      <Variable name="U">the user interface</Variable>
      <Variable name="B">the back end</Variable>
      <Variable name="D">the database</Variable>
    </Feedback>
  </FeedbackGenerator>
  <Output>FORMULAE</Output>
</Task>
```

Figure 3.1: An example of the task specification for an exercise in Iltis [8].

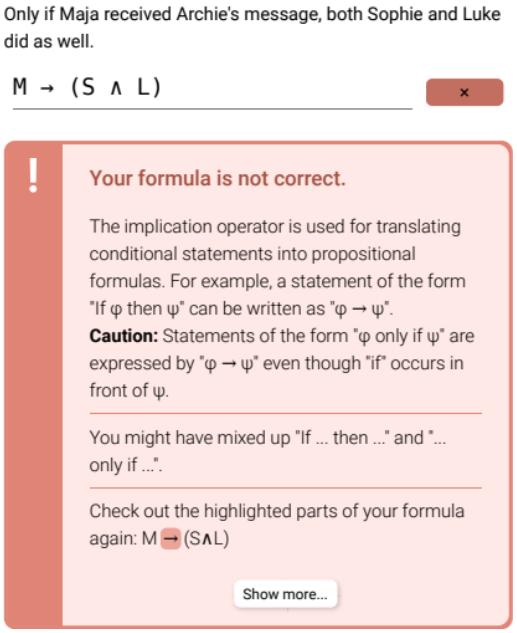


Figure 3.2: An example of the way feedback is given in Iltis [9]. The outputs given by each feedback generator are divided by a line.

order, with each piece of feedback given being from a different feedback generator. The student is also allowed to control the amount of feedback given to them, with the system providing a small amount of feedback and an option for the student to receive more, different feedback (if available) as can be seen in Figure 3.2. This system allows for control of feedback to both the professor, when specifying the task, and the student, depending on their difficulties and preferences. At the time of writing, however, Iltis doesn't feature any kind of natural deduction exercises. It does however offer great examples in regards to exercise creation and creation and distribution of feedback, as well as being hosted online, all of which are important topics for our tool.

3.2 Visual Proof Assistants

3.2.1 Edukera

Edukera [15] is an online tool whose purpose is to teach students logic and math, as such, it contains several different topics, such as calculus, formalization and logic. This will focus on the topic of logic, for which Edukera presents sets of premade tutorials and exercises for both propositional (Connectors) and first-order languages (Quantifiers).

The provided tutorials and exercises are proofs in which the student must select the correct proof rules, which are present on the left side of the page and can be selected by clicking, for each step to correctly solve the exercise, occasionally having to justify the chosen rule, as shown in Figure 3.5. After selecting a proof rule the tool automatically

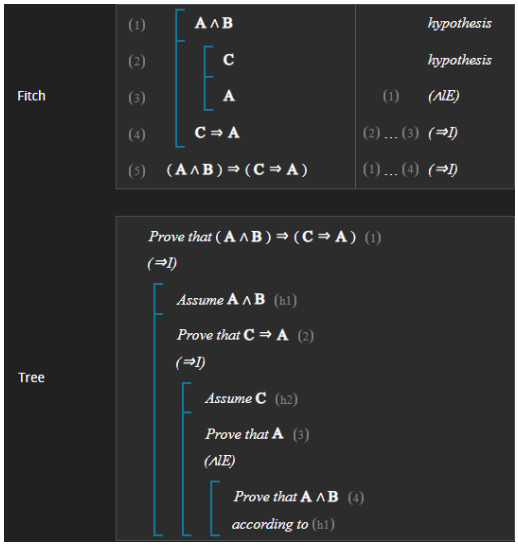


Figure 3.3: The proof formats that can be used for logic exercises in Edukera.

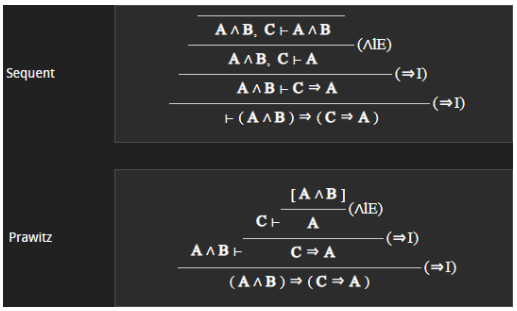


Figure 3.4: The proof formats that can only be visualized separately for logic exercises in Edukera. They can be used in mathematical exercises.

constructs the next step of the proof following the structure needed to make the rule valid. Textual input is only optionally required when justifying a proof rule, however, there are on-screen buttons to write terms instead. Two different proof formats of the proof system are available in Edukera for logic exercises, those being Fitch style and a non-Gentzen tree-style format, which isn't a Gentzen tree, shown in Figure 3.3. An additional two, Sequent and Prawitz, are shown in Figure 3.4 are also available but for visualization only for Logic exercises. Explanations of the rules and their structure are also provided by clicking the magnifying glass icons next to the specific rule. When selecting the first tutorial, the tool will give a step-by-step guided explanation of how it works and how to use it. It should also be noted that Edukera uses Coq as a "mathematical engine"[15] to verify their proofs.

There are some downsides to Edukera. To start, it is unfortunately no longer maintained [3] and as such, it cannot be expected for more features to be added to it. The tool also

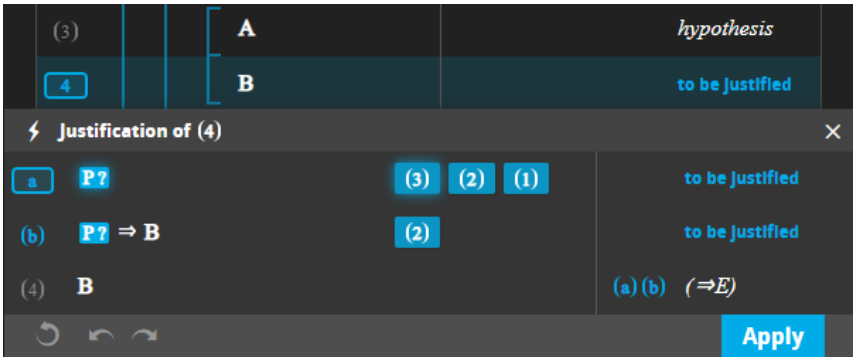


Figure 3.5: Whenever a proof rule needs to be further justified in Edukera, a box such as this one will appear, which must be complete with the correct terms to continue with the proof.

doesn't offer much feedback, with the provided feedback amounting to checking if an attempt at using a proof rule is valid for the current goal and, upon completion of the proof, showing a green checkmark alongside a button to access the next exercise.

Finally, while Edukera allows a user to create a class, which students can enroll in with the given credentials, and have the teacher select exercises for the class to complete, as well as homework and evaluations, the exercises available are those that were already available on the tool, with no possibility for the teacher to create completely new exercises on the tool.

3.2.2 Holbert

Holbert [14] is an open-source project described by its creators as a "work-in-progress pedagogical proof assistant and online textbook platform". It functions fully online, and its main aim is to help teach programming language theory. Holbert's combination of an online textbook and a proof assistant allows students to follow the topics while solving exercises without needing to switch between two different tools.

Holbert's text editor allows a user to create their theorems, axioms, inductive definitions and more. Its key features are:

- Gentzen tree format.
- Goal tag.
- Automatic proof building.
- Creation of exercises, including partially solved ones which might make good beginner exercises.

It provides graphical representations of terms, rules and proofs, allowing the creation of exercises in the form of solving proofs, which are represented as Gentzen trees. For natural deduction exercises, creating a proof involves first establishing a theorem, giving it a name, and then writing the conclusion and the premises of the theorem using Holbert's term language, which is in the form of "untyped lambda calculus"[14]. After creating the theorem, the tool will automatically generate a proof system object, which will display the conclusion of the proof and a goal tag, which is shown in Figure 3.6. The proof is then solved backwards, by clicking on the goal tag, which displays on the right side of the screen, the current goal, the available rules and possible assumptions, as shown in Figure 3.8. Upon selecting a rule, the assistant will automatically generate the structure needed for that rule to be valid, as well as the necessary goal tags for each newly generated subgoal. All of this is done using mouse clicks. The proof will be considered complete once there are no more goal tags present as shown in Figure 3.7.

While its key features are great, Holbert does lack in other aspects. Holbert's term language can start to get confusing whenever writing larger formulas containing several

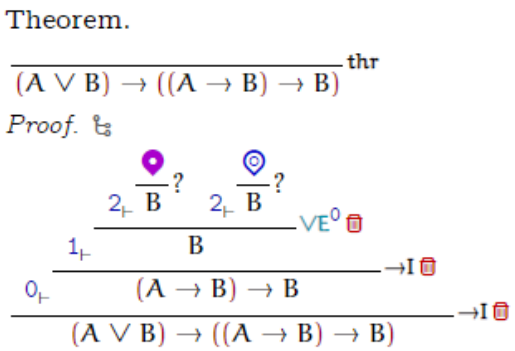


Figure 3.6: A partially built proof in Holbert, showing its goal tags which are present for every remaining subgoal.

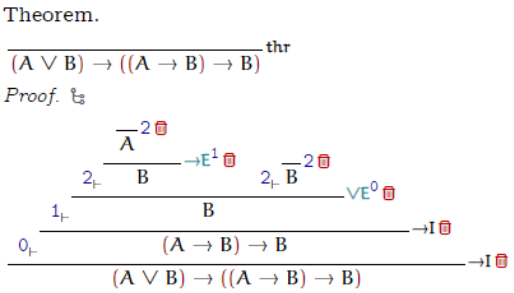


Figure 3.7: A complete proof in Holbert, marked by the lack of any goal tag.

connectors, further amplified by the lack of, for example, buttons that would automatically type out a connector or a term, present in tools such as Fitch [4], which will be introduced later in this section. It should also be noted that, according to its authors, the way the term language is written technically makes its logic unsound [14]. Holbert also noticeably lacks any kind of feedback besides the disappearance of all goal tags once the proof is complete. There is no help given by the assistant to a student who gets stuck on a part of the proof even after multiple attempts, with the student having to resort to asking for help from a teacher or simply using another proof assistant that provides some feedback.

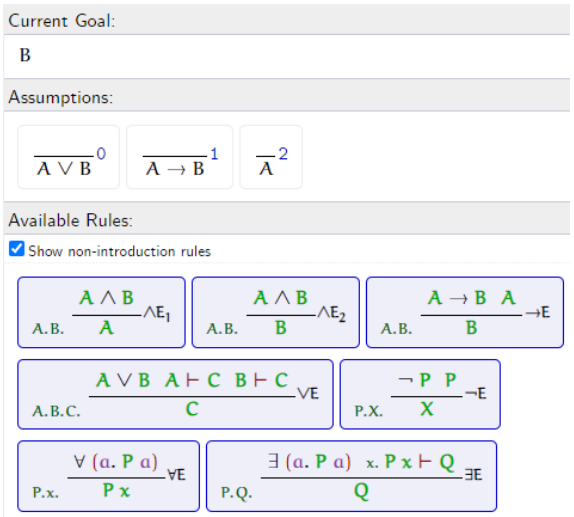


Figure 3.8: Menu displayed by Holbert when clicking on a goal tag.

3.2.3 Fitch

Fitch is a tool provided alongside the Language, Proof and Logic book [4] and whose purpose is to construct and solve formal proofs in first-order logic. It comes in the form of a local application, which can be run locally on a computer, and a set of premade exercises

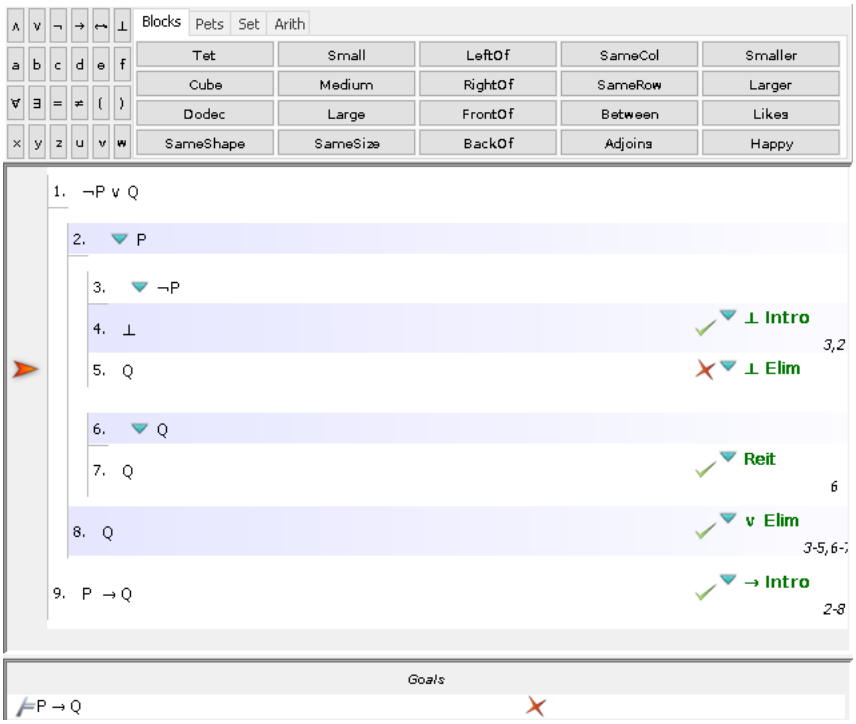


Figure 3.9: An example of Fitch's interface, proof structure and feedback. The tool is able to show which statements cause the proof to fail.

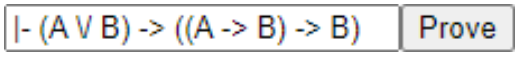
that come from the textbook.

When opening the application, an interface is shown to the user containing a section where the proof is written, in Fitch style format, a section that states the goal(s) of the proof and a set of buttons which when pressed type out the corresponding term or connector. To write in Fitch the user must move the red cursor towards the step they want to write on and then write, utilizing the term buttons on top when necessary. To construct a proof the user has the following options:

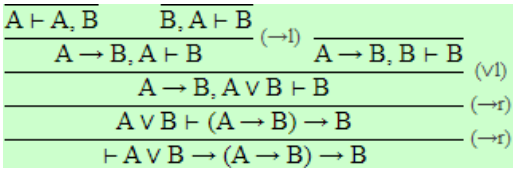
- Add premises by using the Ctrl + R.
- Add and delete goals by accessing the Goal menu in the toolbar.
- Add and delete steps of the proof with Ctrl + A for add after, Ctrl + B for add before and Ctrl + D to delete, being possible to write the formulas as well as select the proof rule for that step.

Additionally, under the Edit section of the toolbar, an option called Author Mode can be toggled. When off, the user is limited to only actions that a student can do to solve the proof. On the other side, to solve a proof a student can do the following actions:

- Add and delete steps of the proof, in the same way as mentioned above.



(a) The textual input for this proof.



(b) The complete proof, with the green background after it's complete and correct.

Figure 3.10: An example of a proof in Logitext.

Figure 3.11: This is how feedback is given in Logitext.

- Add and end subproofs with the Ctrl + P and Ctrl + E hotkeys, respectively. These function just like regular proof, with the possibility of adding steps, writing on them and selecting proof rules.
- Verify the proof with the Ctrl + F hotkey. This will also provide feedback to the student.

With these actions, the student then needs to write the necessary steps and subproofs (when needed), and apply the proof rules, which require the user to put the red cursor on top of the step which is being proven followed by selecting the steps that make the proof rule valid and then select the rule, and finish by verifying the proof. An example of a proof in Fitch, showing its buttons for terms, proof structure, feedback provided and more are shown in Figure 3.9.

Fitch allows easy construction of a proof, which is a very valuable feature, but not the only one. Another strong feature of Fitch is the feedback it provides, as it will individually check each part of the proof for validity and correctness, displaying a green checkmark if the step is well written and correct in the context of the proof or a red cross in case something fails. While it may look simple, this provides a great amount of information to the user, who will be made aware, in case of an error, where this error is and which steps are causing it.

3.2.4 Logitext

Logitext [17] is a tool available online that functions as an educational proof assistant. Logitext utilizes sequent calculus and is intended for students learning proof systems with a Gentzen tree format. Two versions of the tool are available, one which works with classic first-order logic and another which works with intuitionistic propositional logic. Logitext doesn't offer any special features that distinguish it from other proof assistants, and its existence comes from the developer's intention of reusing pre-existing software as much as possible, in an attempt to show the utility of doing so. As such, Logitext was constructed with a combination of Coq, Haskell and Ur/Web, with Coq being used to

check the validity of the proof steps. In the case of the intuitionistic propositional logic version, G4ip was used instead of Coq.

As a proof assistant Logitext requires the user to manually type the theorem that they want to prove, and after clicking the Prove button will redirect the user to a page where the proof is done. To solve the proof, which is done bottom up, the user clicks on the connector or term they want to prove and the assistant will automatically build the next part of the proof, assigning the proof rule that would generate the goal and the subgoals necessary to make it valid. In case the user needs to redo the proof, they can click the turnstile of a subgoal, which resets everything above that part of the proof. An example of a proof in Logitext is shown in Figure 3.10. Apart from the background turning green when completing a proof correctly, Logitext will also give a message when attempting to select a term or connector that isn't valid for the current goal, which is shown in Figure 3.11

Having been released in 2012 [17], this tool doesn't do much different from more recent and visually attractive proof assistants like Edukera and Holbert, but it does manage to fulfill its role in showing the use and utility of reusing pre-existing software in the creation of newer software.

3.2.5 Yoda

Yoda [**Yoda**] was a tool available online for the Universidad de la República in Uruguay. It seems to no longer be maintained, and isn't even available anymore, as the links where it used to be hosted no longer work. With the tool no longer available, the only information that can be found are in articles. According to the paper written by its developers, it was a tool that focused on natural deduction utilizing the Gentzen tree format, implemented with Javascript, JQuery and CSS which worked with both propositional and first-order logic, each one managed by different assistants.

According to its developers, to build a proof of type $\rho \vdash \varphi$ with Yoda a user starts by specifying the conclusion φ and the premises ρ via textual input. Once submitted, the assistant would show a partially built tree with the conclusion as the root, alongside a drop-down list whenever a rule, premise or formula needed to be specified. The system would then check if the action was applicable. If it was, it would build the corresponding subtree, and if it wasn't, it would present the user with an informative message. Premises are displayed in blue when they appear in the proof itself, and while marks aren't explicitly shown, a formula that has been closed will show up crossed and in red. An example of a proof is shown in figure 3.12.

3.2.6 Panda

Panda [7] is an open-source proof assistant tool written in Java and created with the intent to help in the task of teaching natural deduction. Its creators were looking for tools that could fulfill a number of requirements, which are shown in Figure 3.13, but couldn't find one that suited everything and, as such, decided to develop their tool. Panda uses the

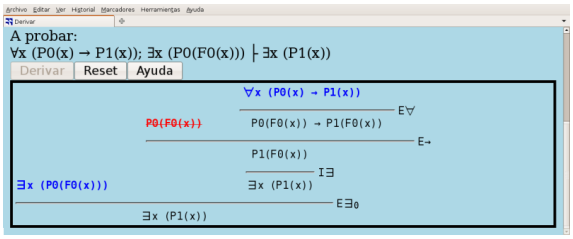


Figure 3.12: Example of a proof in Yoda [**Yoda**].

Gentzen tree style for its proofs and the creators argue in favor of this format, saying that "it seems to us that Gentzen's style proofs are more readable because the tree-like structure of proofs (...) is more evident."

- Easy to install and to use;
- Allows one to check a given proof;
- Allows one to build a proof either with some help, or without help;
- Allows both backward and forward reasoning;
- Allows to easily compose proofs from subproofs;
- Uses Gentzen's proof trees style.

Figure 3.13: The requirements needed by Panda's developers for a natural deduction tool. [7]

To create a proof in Panda, a user must click the "Add a formula..."button and write the formula they want to prove. As stated in their requirements, solving a proof can be done with both backward reasoning and/or forward reasoning. To achieve this, Panda allows the creation of partial proof trees, which can then be moved across the screen and joined to other partial proofs to form a single proof. To make sure the logic of the proof remains sound, Panda keeps track of formulas introduced in the branches of proofs and verifies their variable composition [7]. Proof rules are shown on the left side of the interface with a visual representation of how they can be applied, as shown in Figure 3.14. The rules shown are those that can be applied to the currently selected formula, with the other rules being hidden from the user. The assistant will also show the user the possible decompositions of a proof rule, which can be clicked by the user, and the assistant will automatically apply the appropriate rule, which is also shown in Figure 3.14. Panda also supports first-order logic.

As mentioned prior, the tool allows for the composition of various partial proofs into one. For example, two partial trees can be joined by connecting the root of the first to the leaf of the second. According to the developers, Panda will "use some heuristics"and be able to determine if the trees can be linked via some rule and, if so, apply the corresponding rule. Proofs can also be saved in a LaTeXformat and their specific format, stored as an XML file, which can be reloaded later and can serve as a way to create exercises.

Unfortunately, it seems that Panda is no longer maintained, with the last commit of the source code to GitHub being in 2020 and the latest date of modification found in the source code being from 2013. Both the master application and preview application

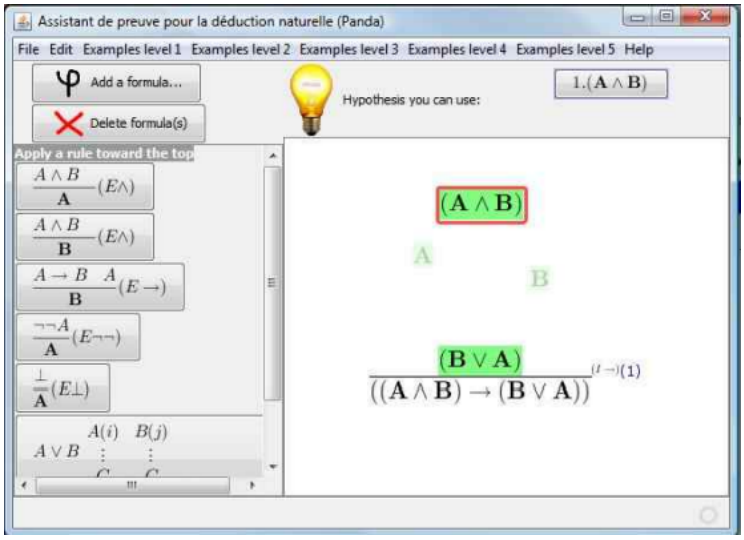


Figure 3.14: Panda's interface, showing proofs, rules and suggestions. [7]

that come with the source code were built around JNLP files, which seem to have been deprecated in JDK 9 and later removed entirely.

3.2.7 Fitch-Checker

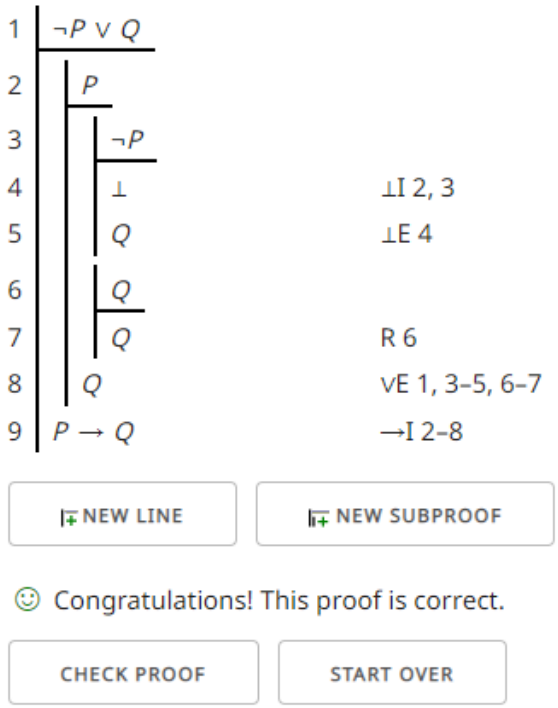


Figure 3.15: Example of a proof in Fitch-Checker.

A simple proof assistant for the proof system used with the Forall x: Calgary book [13]. It is available online [20] for free. The proof assistant, instructions for it, some sets of sample exercises and a visual representation of the proof rules are all present at all times.

The visual representation of the proof rules is on the right side of the page and follows those used in the systems taught in the book.

To start a proof, the user must first textually input the premises and conclusion into their specific boxes and then click the create problem button. This will make a proof system appear below it, which follows a Fitch-style notation and, as such, behaves in a very similar way to Fitch 3.2.3. There are buttons to add new lines and new subproofs. When creating a new line, the user must write down both the formula and the proof rule, the latter of which has to be written in the notation shown on the right side of the page. When creating a subproof, the user must write the formula that is being assumed, and then has the choice to add lines, add another subproof and finish the subproof. Every option is shown as a small button when hovering over a line, on its right side, including an option to delete the line/subproof. Additionally, at the bottom of the proof, there are buttons to start over and check the proof. Checking the proof will give an error when something isn't correctly written (or get stuck in "Checking...") and give a positive message when the proof is correct. An example of a proof is shown in Figure 3.15.

3.2.8 Logan

Logan is an open-source proof assistant developed as part of a thesis project [1] with the intent of being used by students learning logic. It is available for free online [23]. Upon entering the website (or when starting the program) we can see three sections which are Instructions, About the rules and Proof.

In the instructions sections, there are two buttons, one for a manual and another for shortcuts. The manual contains relevant information for the page, explains the layout and then shows a couple of examples of proofs done on the tool, accompanied by photos showing various details that can occur during the writing of a proof. The shortcut tab shows shortcuts for symbols and certain actions such as adding and deleting lines.

The about the rules section contains several buttons which, when clicked, display information about its rule as well as a visual representation of it. It also shows the textual shortcut to write the rule.

The proof assistant section shows 6 buttons, 2 which add lines to the proof, one which resets the proof, one that gives a hint on how to develop the proof at the start, one to save the proof as a pdf and another to export it as LaTeX. The proof assistant uses a Fitch-style format for its system and is suited to handle first-order logic. To start a proof, the user must input the premises and the conclusion into their specific boxes. The assistant will then automatically create a line for each premise, marking them accordingly on the rule slot. After this, the user has to add lines, which are composed initially of two separate boxes, the left one which holds the formula, and the right one which holds the proof rule. To create a subproof, the user has to assume a formula by writing it on the left box and then write "as"on the rule slot, which is a shortcut for "Assumption.". The assistant will automatically write it down as an assumption and give visual feedback to show that it's

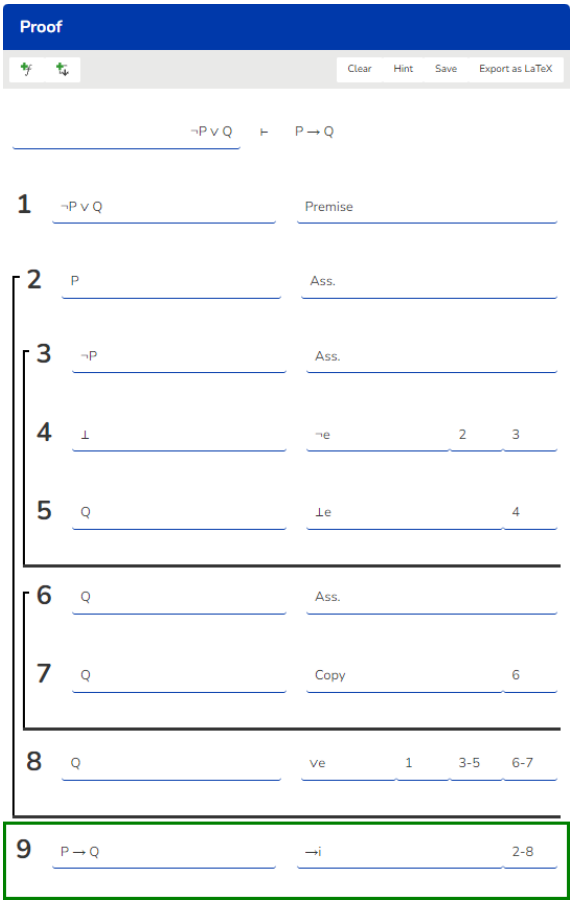


Figure 3.16: Example of a proof in Logan

now a subproof. As mentioned prior there are two buttons to add lines. The difference between these is that one will create lines inside the subproof, while the other will create the line outside of the subproof. To delete a line, the user has to click on the respective line's formula box, delete whatever is written there and then click backspace again. When writing a proof rule on the right box, the box will adapt to the arity of the proof rule, adding the corresponding number of small boxes, where the user can write down the lines which are needed to make the rule valid. The assistant is constantly checking if what is written in the rule box is valid, and will turn red when something isn't in addition to giving feedback describing the error and, when possible, what's expected to be written. Just like Fitch, the assistant will point out which lines of the proof are causing it to be incorrect. When the conclusion is reached and all lines are correct, the box for the line containing the conclusion will turn green. As fully complete proof in Logan is shown in Figure 3.16

Logan is very complete as far as proof assistants go. Unfortunately, it is only available in a Fitch-style format, having no option to be displayed as a Gentzen tree. It also isn't fully capable of creating exercises. While it does allow a user to export the proof in the LaTeXformat, there is no way to import proofs.

3.3 Overview

This section will overview and compare a series of proof assistants that have been analyzed and described. The goal of this thesis is to create a proof assistant tool for educational purposes, this task is critical as we examine what is already available, whether it meets our needs, and what we can take from it for our tool.

First, the format of proofs is an important matter. Most of the listed proof assistants follow a Fitch-style format. Yoda, Holbert, Panda, and Logitext, however, adhere to a Gentzen tree-style format which matches the format that we intend to utilize in our tool, as previously explained in Chapter 1. Of these, the last three are all open-source projects, which allows for the opportunity to study how they construct their proofs in the programming language chosen by their developers.

The assistants' logic must be sound and complete, and as such, it can be worthwhile to check how the assistants verify and reason their proofs. Most of them appear to have their logical reasoning built from the ground up by developers, with no assistance from external reasoners. One exception to this would be Logitext, which the author specifically mentions connecting to Coq to verify proofs. Reutilization of parts of previous proof assistants could prove to be worthwhile and is an angle that must be looked at, as it could save great amounts of extra work while still being rewarding.

The possibility of creating exercises is another important aspect of our tool. How to judge will differ from person to person, as one could simply assume that if a tool allows the input of premises and the conclusion of a proof, that is enough to do the job. If we follow these criteria, then every assistant previously listed except for Edukera can fulfill it. But, in educational terms, an exercise doesn't always consist in fully solving a proof from just its premises and conclusion. For example, a possible beginner's exercise could be one where the student is presented with a fully built proof, except that it doesn't have some of its proof rules written, and the student must correctly write them. Another example could be a partially constructed proof that the student must complete. Following the criteria needed for these types of exercises, tools such as Logitext won't be able to fulfill them. The tool must both allow the partial construction of proofs and then be capable of saving that state, so it can be reloaded later for the students in the same way it was before. Tools such as Fitch, which can save its proofs as files and later reload them in the same state can fulfill this criterion.

Some extra criteria would be welcomed, such as data collection and integration with online platforms, however, none of the proof assistants present here seem to be built around them, and as of their current state, they won't be allowed to provide them. In this case, despite not providing a proof assistant, we have to mention the Iltis MOOC, which features anonymous data collection for purposes similar to what is desired.

Lastly, the ever-important feedback. Feedback is critical for an online teaching platform, as it exists to assist students in place of qualified teachers. While many tools are built for educational purposes, the availability of teachers or tutors can vary in online courses, and

students can also want to test and verify their knowledge without relying on other humans. Many proof assistants were designed with the idea of simply verifying proofs that were previously written by hand, and if a student becomes stuck on the proof, they can simply seek guidance from their teachers. These tools will then see feedback as mostly trivial or completely unneeded. From the tools listed, Logan is the only one that continuously provides feedback during the construction of the proof, although Panda and Fitch also provide a good amount of feedback. It should also be noted that, of these three, only Logan is currently available online. For the remainder of the assistants, the feedback is mostly minimal. Once again, a mention must be given to the Iltis MOOC, which provides a great example of how to provide feedback for exercises of other concepts of logic, as well as allowing the feedback given and received to be dosed in the quantity intended by the teacher and student.

4

Conclusion

4.1 Proposed Work

As established in chapter 1, the goal of this thesis is to develop an online tool that can help students better learn and understand the concepts of natural deduction. To develop the tool, a system will be created that hosts a proof assistant online. This system will be capable of creating and loading proofs. The full proof will be shown to the user, as well as the current goals. The user can then proceed with the development of the proof by tackling the goals. Once a goal is resolved, the assistant will construct the next part of the proof. This is in line with many other visual proof assistants that have been shown in chapter 3. This will be done via a combination of the system's front-end and back-end, with the front-end hosting the visual proof assistant and allowing interaction with it. These interactions will generate information that will be sent to the back-end to be processed, and the back-end will send the result of what was processed back to the front-end. Figure 4.1 shows a mock-up of the system being proposed.

When constructing the proof it is of extreme importance that the logic is sound and complete. To achieve this, the system will communicate with an external logic reasoner by sending it the steps of the proof so they can be verified for validity. For an external reasoner, Coq was chosen, partially due to the existence of the SerAPI [27, 6] protocol, which eases the communication with Coq by serializing "Coq's internal OCaml datatypes from/to JSON or S-expressions (sexps).". This allows a system, for example, to have its back-end communicate with Coq by sending HTTP requests and then parsing JSON responses, allowing the back-end to be written in any programming language that an perform these tasks, such as Java or Python through REST APIs.

The system's front-end will host the proof assistant online, showing users an interface with which they can interact with proofs in a Gentzen Tree format. To achieve this, a combination of HTML, CSS and Javascript (or equivalent) will be used. Frameworks such as React [26] will also come in use here, as they help ease this process.

Regarding the creation of exercises, the assistant will allow users to store the current state of a proof as well as other options such as hints and feedback, which is done via the

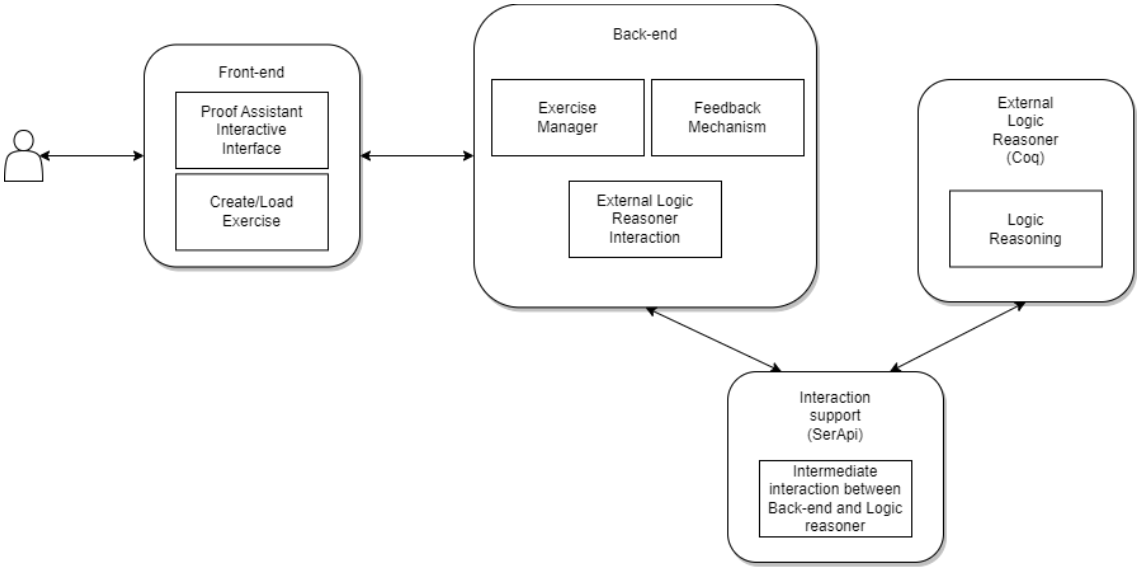


Figure 4.1: A mock-up of the proposed system.

interface hosted on the front-end. The front-end will then send the information to the back-end, which in turn will perform the translation of the proof into a to-be-determined format and save it in a database. Users can then load exercises once again via the interface of the system. The front-end will communicate with the back-end, which will get the exercise's file from the database, translate it and send the information back to the front-end, which displays it. We will call this feature of the back-end an exercise manager, for convenience. This part of the tool is expected to be used by teachers only, and, once it is integrated with an online platform such as moodle, will restrict access to this feature for users without the necessary permissions.

Finally, the system will display different kinds of feedback according to the situation. For example, whenever an invalid step is caught by Coq, the system will inform the user of what went wrong with the step. The way the feedback is shown to the user via the interface will be color-coded, with special attention paid to potentially conflicting cases such as, for example, applying a correct rule in the wrong way (such as not referencing the correct marks), which may lead the user to believe that the rule itself is the wrong one if the entire message has the same color. The type of feedback given can be dependent on the type of exercise or the state of the proof. There is value in determining specific feedback as well as hints for an exercise according to its creator's view. Iltis achieved this by allowing the creator to define rules for its feedback generators. An approach like this might also work for our system and would require the exercise format to accommodate it, which is why the format can only be finished once this situation is resolved. If possible, the proof will be tested step by step, which allows feedback to be provided as soon as the student finishes a step and the external reasoner (and/or other options) provide feedback for it.

4.2 Work Plan

Work done towards this thesis can be divided into some important stages, which are:

- Implementation of the system.
  - Implement back-end.
  - Connect back-end to the external reasoner.
  - Implement front-end.
  - Connect back-end to the front-end.
  - Implement exercise manager.
  - Implement feedback mechanism.
- Testing.
- User testing.
- Writing the dissertation document.

The first stage, the implementation of the system, is subdivided into several tasks. These tasks include the implementation of the back-end, the front-end, and the integration of the back-end with the external reasoner, as well as with the front-end. For the development of the front-end, an interface hosting the visual proof assistant will be implemented, as well as making the assistant interactive. The implementation of the back-end, besides its integration with Coq for verification purposes, includes implementing support for propositional and first-order logic, feedback mechanism and the exercise manager.

The second stage, testing, will happen consistently throughout the implementation of the system, to ensure it works correctly, but will also be applied with a larger effort during certain milestones, which are reflected in the Gantt Chart shown in Figure 4.2.

The third stage, user testing, will happen after the previous three stages are finished, and will consist of user testing to evaluate the usability of the system and receive feedback regarding the system's features and what could be changed.

The final stage, writing the dissertation document, will occur simultaneously with the other stages, with more focus in the later months.

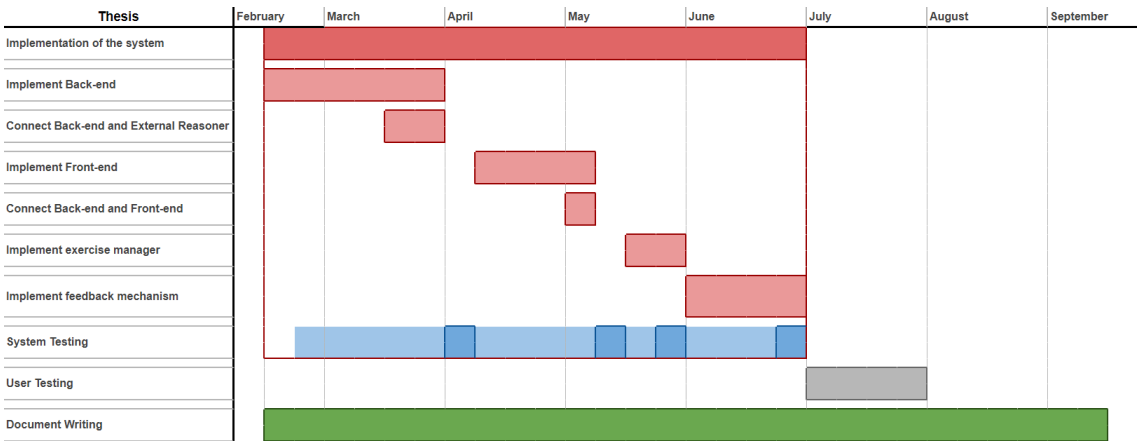


Figure 4.2: Proposed Gantt Chart for the work plan.

Bibliography

[1] F. ABRAHAMSSON et al. *Proof Editor for Natural Deduction*. 2021. URL: https://odr.chalmers.se/server/api/core/bitstreams/e3cadeaa-efab-4e66-9a18-a41af5617d3e/content (cit. on p. 25).

[2] Alrubyli and Yazeed. "Natural Deduction Calculus for First-Order Logic". In: *CoRR* (2021). DOI: 10.48550/ARXIV.2108.06015. URL: https://arxiv.org/abs/2108.06015 (cit. on p. 11).

[3] E. Bartzia, A. Meyer, and J. Narboux. "Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis". In: *INDRUM 2022: Fourth conference of the International Network for Didactic Research in University Mathematics*. Ed. by M. Trigueros. Reinhard Hochmuth. Hanovre, Germany, 2022. URL: https://hal.science/hal-03648357 (cit. on pp. 11, 17).

[4] J. Barwise et al. *Language, proof and logic*. CSLI publications, 2000. ISBN: 157586374X (cit. on p. 19).

[5] J. M. Fitzpatrick et al. "Lessons Learned in the Design and Delivery of an Introductory Programming MOOC". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '17. Seattle, Washington, USA: Association for Computing Machinery, 2017, pp. 219–224. ISBN: 9781450346986. DOI: 10.1145/3017680.3017730. URL: https://doi.org/10.1145/3017680.3017730 (cit. on p. 1).

[6] E. J. Gallego Arias. *SerAPI: Machine-Friendly, Data-Centric Serialization for Coq*. Tech. rep. MINES ParisTech, 2016-10. URL: https://hal-mines-paristech.archives-ouvertes.fr/hal-01384408 (cit. on p. 29).

[7] O. Gasquet, F. Schwarzentruber, and M. Strecker. "Panda: A proof assistant in natural deduction for all. A Gentzen style proof assistant for undergraduate students". In: *Tools for Teaching Logic: Third International Congress, TICTTL 2011, Salamanca, Spain, June 1-4, 2011. Proceedings*. Springer. 2011, pp. 85–92. ISBN: 3642213499 (cit. on pp. 3, 22–24).

[8] G. Geck et al. "Iltis: Learning Logic in the Web". In: *CoRR* abs/2105.05763 (2021). DOI: 10.48550/ARXIV.2105.05763. URL: https://iltis.cs.tu-dortmund.de (cit. on pp. 2, 15, 16).

[9] G. Geck et al. "Introduction to Iltis: An Interactive, Web-Based System for Teaching Logic". In: ITiCSE 2018 (2018). DOI: 10.1145/3197091.3197095. URL: https://doi.org/10.1145/3197091.3197095 (cit. on pp. 2, 15, 16).

[10] S. Hedman. *A First Course in Logic: An introduction to model theory, proof theory, computability, and complexity*. OUP Oxford, 2004. ISBN: 0198529813 (cit. on pp. 6, 11).

[11] D. Lebron and H. Shahriar. "Comparing MOOC-based platforms: Reflection on pedagogical support, framework and learning analytics". In: *2015 International Conference on Collaboration Technologies and Systems (CTS)*. 2015, pp. 167–174. DOI: 10.1109/CTS.2015.7210417 (cit. on p. 1).

[12] S. Lovrenčić and M. Čubrilo. "OVERVIEW OF ONLINE TEACHING RESOURCES FOR LOGIC PROGRAMMING". In: *The Seventh International Conference on eLearning* (2016) (cit. on pp. 2, 15).

[13] P. D. Magnus and T. Button. *Forall X: Calgary: An introduction to formal logic*. 2021. ISBN: 9798527349504. URL: https://forallx.openlogicproject.org/forallxyyc.pdf (cit. on p. 24).

[14] L. O'Connor and R. Amjad. "Holbert: Reading, Writing, Proving and Learning in the Browser". In: *arXiv preprint arXiv:2210.11411* (2022-10). DOI: 10.48550/arxiv.2210.11411. URL: https://arxiv.org/abs/2210.11411v1 (cit. on pp. 18, 19).

[15] B. Rognier and G. Duhamel. "Présentation de la plateforme edukera". In: *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*. Ed. by J. Signoles. Saint-Malo, France, 2016. URL: https://hal.science/hal-01333606 (cit. on pp. 10, 16, 17).

[16] P. Sands and A. Yadav. "Self-Regulation for High School Learners in a MOOC Computer Science Course". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 845–851. ISBN: 9781450367936. DOI: 10.1145/3328778.3366818. URL: https://doi.org/10.1145/3328778.3366818 (cit. on p. 1).

[17] E. Z. Yang. "Academic software reuse, an experience report". In: *6.UAP report advised by Adam Chlipala* (2012) (cit. on pp. 21, 22).

[18] A. Yushkovskiy. "Comparison of Two Theorem Provers: Isabelle/HOL and Coq". In: *arXiv preprint arXiv:1808.09701* (2018). DOI: 10.48550/ARXIV.1808.09701. URL: https://arxiv.org/abs/1808.09701 (cit. on pp. 11, 12).

Webography

[19] J. Avigad. *Classical and constructive logic*. (Visited on 2023-01-28) (cit. on pp. 5, 6).

[20] *Fitch-Checker*. URL: https://proofs.openlogicproject.org/ (visited on 2023-01-16) (cit. on p. 24).

[21] *Internet Encyclopedia of Philosophy - Natural Deduction*. URL: https://iep.utm.edu/natural-deduction/ (visited on 2023-01-15) (cit. on p. 8).

[22] *JsCoq*. URL: https://coq.vercel.app/ (visited on 2023-01-14) (cit. on p. 12).

[23] *Logan*. URL: https://datx02-21-16.github.io/datx02/ (visited on 2023-01-18) (cit. on p. 25).

[24] *Logic and Proof textbook for Lean prover*. URL: https://leanprover.github.io/logic_and_proof/natural_deduction_for_first_order_logic.html (visited on 2023-01-21) (cit. on p. 8).

[25] *Natural Deduction Pack*. URL: https://users.ox.ac.uk/~logicman/carr/NDpack.pdf (visited on 2023-01-15) (cit. on pp. 7, 12, 13).

[26] *ReactJs*. URL: https://reactjs.org/ (visited on 2023-02-05) (cit. on p. 29).

[27] *SerAPI*. URL: https://github.com/ejgallego/coq-serapi (visited on 2023-02-02) (cit. on p. 29).

[28] *Tactics*. URL: https://coq.inria.fr/refman/proof-engine/tactics.html (visited on 2023-01-23) (cit. on p. 12).