NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DANIEL GONÇALVES FUSETA ROSA MACAU

BSc in Computer Science

INTERACTIVE TOOL FOR PRACTICING AND EVALUATING LOGIC EXERCISES

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2025

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

INTERACTIVE TOOL FOR PRACTICING AND EVALUATING LOGIC EXERCISES

DANIEL GONÇALVES FUSETA ROSA MACAU

BSc in Computer Science

Adviser: Ricardo Gonçalves
Assistant Professor, NOVA University Lisbon

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

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2025

Abstract

Logic is a core topic in areas like mathematics and computer science and is a key discipline in the computer science curriculum. In this context, students and teachers often lack tools to complement logic classes, especially for practicing exercises, as it is not always possible for students to interact with teachers to address their difficulties.

Online courses are a good example of tools that provide accessible resources to a wide variety of people. However, in the field of logic, these courses typically do not allow for the addition of new material and are often limited in the number of exercises. Logic includes various kinds of exercises, with natural deduction posing the greatest challenges for students. Unfortunately, there are no tools available to support these exercises. Additionally, many of these tools lack effective feedback mechanisms.

Developing an effective feedback system is essential to achieving a successful tool. However, it is a challenging task, as we do not want a system where students are always dependent on the feedback and stop thinking by themselves, but at the same time, we do not want them to lose interest in learning because they are always stuck in exercises.

In this thesis, we propose the development of an interactive online tool whose goal is to help students practice logic exercises. We want a system that is meant for everyone, from the novice user that is starting their journey in learning logic to the experienced user that wants to improve even more their skills. Our primary focus will be on creating an effective feedback system that will guide students throughout the construction of proofs. Additionally, we aim to enable the creation and evaluation of natural deduction exercises. The tool will support future expansions for new exercises and will be integrated with e-learning platforms like Moodle for class and grade management.

With this project, we want students to have a studying tool that is accessible to everyone and capable of assisting them in overcoming, in an efficient and engaging way, the challenges they face in logic exercises.

Keywords: Logic, Deductive System, Feedback, Interactive tool, Natural Deduction, Propositional Logic, First-Order Logic

Resumo

Lógica é um tópico essencial em áreas como matemática e ciência da computação, sendo uma disciplina chave no currículo de informática. Neste contexto, alunos e professores carecem de ferramentas para complementar as aulas, especialmente para praticar, pois nem sempre os alunos podem interagir com os professores para esclarecer dúvidas.

Os cursos online são um bom exemplo de ferramentas que oferecem recursos acessíveis a todos. No entanto, no campo da lógica, estes cursos tipicamente não permitem a adição de novos materiais e são limitados no número de tipos de exercícios. A lógica inclui diversos tipos de exercícios, sendo a dedução natural sendo o mais desafiador. Infelizmente, não existem ferramentas disponíveis com esses exercícios. Além disso, muitas ferramentas carecem de mecanismos eficazes de feedback, deixando os utilizadores perdidos.

Desenvolver um sistema de feedback eficaz é essencial para alcançar uma ferramenta de sucesso. No entanto, é uma tarefa desafiante, pois não queremos um sistema onde os alunos dependam do feedback, mas ao mesmo tempo, não queremos que percam o interesse em aprender.

Nesta tese, propomos o desenvolvimento de uma ferramenta online interativa com o objetivo de ajudar os alunos a praticar exercícios de lógica. Queremos um sistema que seja acessível a todos, desde o utilizador novato em lógica até o utilizador experiente. O nosso foco principal será criar um sistema de feedback eficaz que guiará os alunos ao longo da construção das provas. Além disso, pretendemos fornecer mecanismos para adicionar novos exercícios e avaliá-los. Esta ferramenta suportará exercícios de dedução natural, sendo desenhada para permitir futuras expansões, incluindo outros tipos de exercícios. Por fim, integraremos a nossa ferramenta com uma plataforma de e-learning online, como o Moodle, uma vez que oferece mecanismos para gerir aulas e notas.

Com este projeto, pretendemos que os alunos tenham uma ferramenta de estudo acessível a todos, capaz de ajudá-los a superar, de uma forma eficiente e envolvente, os desafios que enfrentam nos exercícios de lógica.

Palavras-chave: Lógica, Sistemas Dedutivos, Feedback, Ferramentas Interativas, Dedução Natural, Lógica Proposicional, Lógica de Primeira Ordem

Contents

List of Figures

List of Tables

Acronyms

DAM Directed Acyclic Multigraph *(pp. 17, 18)*

FOL First-Order Logic *(pp. 6–8, 14–17, 20, 21, 27, 30)*

ND Natural Deduction *(pp. 8, 11–13, 15, 17, 22, 23, 27–30, 33)*

PL Propositional Logic *(pp. 4–8, 14, 17, 27, 30)*

WFF Well Formed Formula *(pp. 5, 7)*

1

Introduction

1.1 Motivation

Logic is a fundamental topic in numerous fields, especially in mathematics and computer science. In computer science, it plays a crucial role across various domains, including architecture, software engineering, programming languages, databases, artificial intelligence, algorithms, and the theory of computation [7]. As such, understanding computational logic is essential for anyone studying computer science. It is especially important for formal reasoning, which is key to solving problems and making decisions in computing. Nowadays, many universities offering computer science degrees include logic courses as part of their curriculum.

Despite the importance of logic, both learning and teaching it present significant challenges. Many logic courses require extensive practice and exposure. While traditional methods like books can be useful, they are not as effective as online interactive tools in supporting student learning. Additionally, professors often struggle to provide and assess exercises efficiently, making it more difficult for students to practice and receive timely feedback.

Digital platforms offer valuable tools to complement traditional classes, making educational resources more interactive and accessible to everyone. A notable example of those digital platforms is Massive Open Online Courses (MOOCs) [2]. These courses grew in popularity during the COVID-19 pandemic and have had a significant impact on higher education, as they provide some benefits that normally traditional classes do not. They are flexible by allowing students to study at their own pace whenever they want. This freedom is particularly important for students who may need additional time and practice to understand certain ideas or even for those who want to progress faster. MOOCs can help with a common issue in classrooms, where students often hesitate to ask questions because they worry about looking unintelligent or asking something silly. This can effectively affect the way students progress in their learning, as they create gaps in their understanding that impede or slow down their progress. These digital platforms often provide material and exercises equipped with feedback, allowing learners to identify

and address misconceptions in real time, without the pressure of interrupting a class or asking a teacher for help.

However, in the field of logic education, there are few online tools with characteristics similar to those mentioned, such as Iltis [10, 11] and Logic4Fun [21]. Most of the available courses were created years ago, relying on outdated interfaces that can make the user experience less intuitive and engaging. Additionally, some teaching methods have changed, but many of these tools remain the same. They offer a limited variety of exercises and do not allow teachers to add new material.

Logic courses usually include various types of exercises, and natural deduction exercises are considered one of the most difficult for students due to the complex reasoning involved. These exercises are important because they cover both Propositional Logic and First-Order Logic, which extends the first. For this reason, they appear in all tests and exams at NOVA University Lisbon, as they cover both parts of the course. According to professor Ricardo Gonçalves, coordinator of the Computational Logic course at NOVA University Lisbon, these exercises are particularly challenging and essential to the subject. Unfortunately, there are currently no effective online tools to support these exercises.

In addition to these limitations, another significant issue in the realm of online logic education is the lack of effective feedback mechanisms. Most tools do not provide feedback at all, and those that do often make it too vague to be useful. The lack of feedback can cause students to feel lost in the exercises, leading to a loss of interest or motivation. It can also lead to lower student retention rates, potentially impacting the platform's long-term success. Research shows that any feedback, even if negative, is better than none at all [26]. Developing a tool with effective feedback mechanisms is a key factor in achieving a successful one. However, implementing feedback is not an easy task, as it is necessary to find a balance between a system where students are always dependent on feedback and stop thinking for themselves and a system where students lose interest in learning because they are always stuck in exercises. It becomes even harder when considering a tool that will be used by a large number of students with different levels of skills [8].

Recent efforts have started to address this problem, which could provide the basis for an online system for logic. However, the range of specific types of logic exercises that have been implemented is very limited, both in number and depth, and they were created solely for testing purposes, so they do not include assistance systems.

This highlights the need for the development of a more complete and engaging platform that complements logic courses. Such a platform should place particular emphasis on providing effective feedback mechanisms to enhance the teaching and learning of logic.

1.2 Problem Formulation

Considering the topics previously mentioned, we propose, in this dissertation, a plan to design and implement an online platform that complements logic classes, enabling students to practice natural deduction exercises. The primary challenge of this work will

be identifying and implementing feedback mechanisms that provide valuable guidance while keeping the user engaged and motivated.

The platform will have two distinct perspectives: one for students and another for teachers.

From the students' perspective, we aim to provide an online environment that presents Natural Deduction exercises in tree shape, covering both Propositional Logic and First-Order Logic, allowing students to practice effectively. We will equip these interactive exercises with an advanced feedback and hint system. It will not only be capable of reporting structural and conceptual errors, but it will also be able to adapt its assistance to the students' solutions to guide them efficiently through the exercises. There will be different levels of feedback and hints. Some will offer more generic guidance, while others will provide more precise help but in a balanced manner. The system will automatically adjust its assistance level according to the students' proficiency, determined by their past mistakes. Additionally, students will be able to submit their exercises for evaluation and access the solutions.

From the teachers' perspective, we also want to provide an online interface with an intuitive way to add new exercises that can be evaluated and a way to grade them.

Finally, our tool should be integrated with an online e-learning platform like Moodle, utilizing its built-in mechanisms for managing classes and grades.

1.3 Document Structure

This document is organized into several chapters, each addressing a key aspect of the project. Chapter 2 provides the necessary theoretical background, covering fundamental concepts in logic, Natural Deduction, and proof assistants. Chapter 3 examines existing online tools relevant to our work, with a particular focus on their implemented feedback systems. Chapter 4 explores the challenge of providing meaningful feedback, presenting key elements and a potential algorithmic approach to tackle this issue. Finally, Chapter 5 outlines our methodology, detailing our strategy, work plan, and evaluation process to ensure the project's success.

2

Background

In this chapter, we will provide the necessary context to understand the main concepts explored throughout this document. We begin with an introduction to logic focusing on Propositional Logic and First-Order Logic. For each of these logic branches, we will describe briefly its syntax and semantics. With this basic knowledge as a foundation, we will then introduce Natural Deduction, a fundamental topic for this work, as it explains what it is and presents examples of the types of exercises we will develop. Finally, we will discuss proof assistants, with a particular focus on Isabelle/HOL and Lean, exploring how these tools can help us in developing our feedback system.

2.1 Propositional Logic

Logic in general is defined as the study of the principles of reasoning. Propositional Logic (PL) is a branch of logic that focuses on the study of propositions and their relationships. The goal of logic in computer science is to create languages that help us represent situations we deal with as computer scientists. These languages allow us to think about and analyze situations in a structured way. By using logic, we can build clear and valid arguments about them, ensuring they make sense and can be tested, defended, or even carried out by a machine [15]. Propositions are the basic building blocks of PL. A proposition is a declarative statement that has a truth value, which can be either true (denoted as T or 1) or false (denoted as F or 0), but not both.

Examples of Propositions:

p: "It is raining." **q:** "It is cold." **r:** "It is snowing."

To define a formal language, one must choose the alphabet of the language and establish the set of words that make up the language. These words are usually called formulas when the formal language is associated with a logic, as is the case here. The alphabet of the language is a set of symbols that, when combined, form formulas, with each formula being a finite sequence of symbols from the alphabet [12]. In PL, we have propositional symbols, conventionally represented by lowercase letters (p, q, r), and symbols that represent

relations between propositions ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$), also known as logical connectives. To represent generic formulas, we use Greek letters (ϕ, ψ, γ). Another important symbol in PL is parentheses, which help resolve ambiguities in formulas, similar to their use in mathematics to indicate operation priority. Additionally, we have two constants, one to represent truth, ($\top$), and one to represent falsehood ($\bot$).

For the sake of interpretation, we will use the propositions (p, q and r) introduced in the previous example in the following explanations. Tables 2.1 and 2.2 list all the logical constants and connectives of the PL alphabet.

Symbol	**Name**	**Value**
$\top$	Top	True or 1
$\bot$	Bottom	False or 0

Table 2.1: Logical constants in Propositional Logic

Symbol	**Name**	**Arity**	**Example**
$\neg$	Not	1	$\neg p$: "It is not raining."
$\wedge$	And	2	$p \wedge q$: "It is raining and it is cold."
$\vee$	Or	2	$p \vee q$: "It is raining or it is cold."
$\rightarrow$	Implication	2	$p \rightarrow q$: "If it is raining, then it is cold."
$\leftrightarrow$	Equivalence	2	$p \leftrightarrow q$: "It is raining if and only if it is cold."

Table 2.2: Logical connectives in Propositional Logic

We have defined the symbols that make up the PL alphabet. Now, we will specify which sequences of these words are valid by defining Well Formed Formula (WFF). A WFF in PL is defined inductively using a set of rules that specify the conditions for a formula to be considered well-formed. These rules build upon each other, allowing for the construction of more complex logical expressions.

$$\begin{cases} \top \text{ is a WFF}, & \\ \bot \text{ is a WFF}, & \\ \alpha \text{ is WFF} & \text{if } \alpha \text{ is a proposition}, \\ \neg\alpha \text{ is a WFF} & \text{if } \alpha \text{ is a WFF}, \\ (\alpha \wedge \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs}, \\ (\alpha \vee \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs}, \\ (\alpha \rightarrow \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs}, \\ (\alpha \leftrightarrow \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs} \end{cases}$$

Examples of WFF:

$\top$: "True".

$((p \wedge q) \rightarrow r)$: "If it is raining and it is cold, then it is snowing."

$((p \rightarrow q) \wedge (q \rightarrow r))$: "If it is raining, then it is cold, and if it is cold, then it is snowing."

To define a language, we also need to define its semantics. In PL, this is no different. Given a formula, we want to determine its truth value, which depends on the truth values of its propositional symbols. To do so, we must define an interpretation or valuation (V), which is a function that associates a truth value with each propositional symbol in a set (P), formally $V : P \to \{0, 1\}$ [12]. We say that a formula φ is satisfiable by an interpretation if the interpretation satisfies the formula (if it evaluates to true in that interpretation), denoted by $V \Vdash \varphi$. Otherwise, the formula is not satisfiable under that interpretation, denoted by $V \nVdash \varphi$.

$$
\begin{cases}
V \Vdash \top, & \\
V \Vdash p & \text{if } p \text{ is a proposition and its interpretation is true,} \\
V \Vdash \neg\alpha & \text{if } V \nVdash \alpha, \\
V \Vdash (\alpha \vee \beta) & \text{if either } V \Vdash \alpha \text{ or } V \Vdash \beta, \\
V \Vdash (\alpha \wedge \beta) & \text{if both } V \Vdash \alpha \text{ and } V \Vdash \beta, \\
V \Vdash (\alpha \to \beta) & \text{if whenever } V \Vdash \alpha, \text{ then } V \Vdash \beta, \\
V \Vdash (\alpha \leftrightarrow \beta) & \text{if both } V \Vdash \alpha \text{ and } V \Vdash \beta \text{ are either both true or both false.}
\end{cases}
$$

With this definition, we can evaluate the nature of a formula.

Possible: Exists an interpretation that satisfies it.

Valid (Tautological): Is satisfied by all interpretations.

Contradictory: Is not satisfied by any interpretation.

A key task in PL and in Logic in general is to determine whether a formula ϕ is a semantic consequence of a set of formulas Γ, denoted by $\Gamma \vdash \phi$ [12]. Given a set of premises and a conclusion, the conclusion is considered a semantic consequence of the premises if, for every interpretation of the propositions, whenever the interpretation satisfies all the premises, it must also satisfy the conclusion. In other words, if the set of premises is true under an interpretation, the conclusion must also be true under that interpretation.

2.2 First-Order Logic

Another branch of logic is First-Order Logic (FOL), also known as predicate logic. Unlike PL, which focuses solely on simple declarative statements, FOL extends this by introducing new components that enable us to express more complex declarative sentences, capturing relationships between objects and their properties within a specified domain [15]. In this context, objects refer to specific entities within the domain of discourse, such as animals, persons, numbers, or things, which can be described or related through properties and predicates.

Examples of First-Order Sentences:

"There's a black cat that likes baths."

"John is a friend of Mary."

"If a variable is an integer and positive, then it is greater than zero."

FOL extends PL syntax by adding features that make it more expressive. It introduces quantifiers, which allow us to generalize or specify expressions, making it possible to express universal truths ($\forall$) or existential statements ($\exists$) [15]. Moreover, FOL enables the use of variables, typically represented by (x, y, z) within a given domain. Predicates, which express properties or relationships, allow FOL to capture facts about objects and their interactions. Predicates are represented by words starting with capital letters, such as Cat(x), which expresses that x is a cat. Predicates always return a truth value and can have different arities. We can also define binary relations, such as equality, which is represented by the symbol =. Equality is a binary relation that indicates when two terms refer to the same object in the domain. Similar to predicates, FOL includes functions that allow referring to objects in the domain based on other objects in the domain. Functions are represented by words starting with lowercase letters, such as father(x), which return a specific value from the domain (e.g., the father of x). Tables 2.3 and 2.4 present examples of symbols in FOL.

Symbol	**Name**	**Example**
x, y, z	Variables	x: "An individual object"
$black$	Constants	$black$: "Value that is fixed"
$Cat(x)$	Predicates	$Cat(x)$: "True if x is a cat"
$color(x)$	Functions	$color(x)$ = black: "The color of x is black"

Table 2.3: Variables, constants, predicates, and functions in First-Order Logic

Symbol	**Name**	**Example**
$\forall$	Universal	$\forall x\,(Cat(x) \rightarrow Mammal(x))$: "All cats are mammals"
$\exists$	Existential	$\exists x\,((Cat(x) \wedge (color(x) = \text{black})) \wedge LikesBaths(x))$: "There's a black cat that likes baths"

Table 2.4: Quantifiers in First-Order Logic

We can extend the definition of a WFF from PL to represent a WFF in FOL. To accomplish this, we must first introduce a new concept known as a term. A term is an expression that represents a specific object in the domain. A simple version of an inductive definition of a WFF within FOL is shown below.

$$\begin{cases} c \text{ is a term} & \text{if } c \text{ is a constant,} \\ x \text{ is a term} & \text{if } x \text{ is a variable,} \\ f(t_1, \ldots, t_n) \text{ is a term} & \text{if } f \text{ is a function with arity } n \text{ and } t_1, \ldots, t_n \text{ are terms.} \end{cases}$$

$$\begin{cases} \top \text{ is a WFF} & \\ \bot \text{ is a WFF} & \\ P(t_1, \ldots, t_n) \text{ is a WFF} & \text{if } P \text{ is a predicate with arity } n \text{ and } t_1, \ldots, t_n \text{ are terms,} \\ \neg\alpha \text{ is a WFF} & \text{if } \alpha \text{ is a WFF,} \\ \forall x\, \alpha \text{ is a WFF} & \text{if } \alpha \text{ is a WFF and } x \text{ is a variable,} \\ \exists x\, \alpha \text{ is a WFF} & \text{if } \alpha \text{ is a WFF and } x \text{ is a variable,} \\ (\alpha * \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs and } * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}, \\ (\alpha = \beta) \text{ is a WFF} & \text{if } \alpha \text{ and } \beta \text{ are WFFs.} \end{cases}$$

The semantics of FOL is an extension of that of PL, incorporating the interpretation of quantifiers and terms, and allowing for a more expressive representation of logical relationships. The concepts of possibility, validity, tautology, and contradiction are also applicable in FOL, maintaining the same general meaning as in PL. Similarly, the concept of logical consequence holds in FOL, where the truth of a conclusion depends on the truth of its premises, just as in PL.

2.3 Natural Deduction

We have already presented the notion of semantic consequence. Now, we will show how to prove whether an expression is a semantic consequence of a set of formulas using deduction systems. There are two ways to approach a deduction system: the semantic approach looks at what the formulas mean in different ways, and the syntactic approach looks at how to use symbols in a deductive system [12]. There are numerous deductive systems in logic, with some of the most well-known being resolution, tableau, and natural deduction. In this thesis, we will concentrate on the Natural Deduction (ND) system, a type of syntactic deduction system that uses a pre-defined set of rules known as inference rules. By applying inference rules to the premises, we hope to get some more formulas, and by applying more inference rules to those, to eventually reach the conclusion [15].

There are different styles that represent these proofs. For example, the Fitch style uses a linear structure with deeper indentation levels to represent assumptions or intermediate steps in the proof, and the Gentzen style organizes the proof in a tree-shaped structure. In this thesis, we will only focus on the tree-style representation. These tree-shaped structures, also known as deduction trees, represent proofs and are built by starting with individual trees and successively applying rules of inference to generate new trees. Individual trees are constructed from nodes, which are formulas. The formulas at the leaves are called hypotheses and are associated with marks (numbers). The formula at the root is the conclusion of the proof. Marks are used to identify distinct hypotheses that derive from the rules and premises. The rules are represented using fractions (a horizontal line), where the premises or hypotheses appear above the line and the rule's conclusion appears below it. The rule's name and the marks for the hypotheses are normally placed on the right-hand side of the fraction.

There are two types of rules for each logical connective: introduction (I), which constructs more complex formulas from simpler ones, and elimination (E), which extracts information from complex formulas. Additionally, a special rule, known as absurdity ($\bot$), allows deriving any conclusion from a contradiction. Each rule has its own characteristics, and some can only be applied under specific circumstances, called side conditions. For simplicity, we will not cover them here. Some rules close hypotheses, while others do not. A hypothesis is considered closed if its mark is used in a rule of the tree; otherwise, it is open.

An example of a rule that can close a hypothesis is the Introduction rule for Implication. This rule introduces a new hypothesis containing the antecedent of the implication, which can then be used in subproofs within the proof where the rule was applied. A different example is the Introduction rule for Conjunction, which creates branching by splitting the proof into two subproofs, requiring the proof of both the left and right sides of the conjunction. Below, we present the complete list of inference rules. In these rules, $\mathcal{D}$ represents subtrees within branches, and marks are denoted by m and n.

Propositional Logic Rules

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \varphi & \psi\end{array}}{\varphi \wedge \psi}\ (\wedge_I)$$

Conjunction Introduction

$$\frac{\begin{array}{c}\mathcal{D} \\ \varphi \wedge \psi\end{array}}{\varphi}\ (\wedge_{E_r}) \qquad \frac{\begin{array}{c}\mathcal{D} \\ \varphi \wedge \psi\end{array}}{\psi}\ (\wedge_{E_k})$$

Conjunction Elimination, right **Conjunction Elimination, left**

$$\frac{\begin{array}{c}\mathcal{D} \\ \varphi\end{array}}{\varphi \vee \psi}\ (\vee_{I_r}) \qquad \frac{\begin{array}{c}\mathcal{D} \\ \psi\end{array}}{\varphi \vee \psi}\ (\vee_{I_l})$$

Disjunction Introduction, right **Disjunction Introduction, left**

$$\frac{\begin{array}{ccc} & [\varphi_1]^m & [\varphi_2]^n \\ \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ \varphi \vee \psi & \psi & \psi\end{array}}{\psi}\ (\vee_E, m, n)$$

Disjunction Elimination

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ \varphi & \varphi \to \psi\end{array}}{\psi}\ (\to_E) \qquad \frac{\begin{array}{c}[\varphi]^m\\ \mathcal{D}\\ \varphi\end{array}}{\varphi \to \psi}\ (\to_I, m)$$

Implication Elimination **Implication Introduction**

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ \varphi & \neg\varphi\end{array}}{\bot}\ (\neg_E) \qquad \frac{\begin{array}{c}[\varphi]^m\\ \mathcal{D}\\ \bot\end{array}}{\neg\varphi}\ (\neg_I, m)$$

Negation Elimination **Negation Introduction**

$$\frac{\begin{array}{c}[\neg\varphi]^m\\ \mathcal{D}\\ \bot\end{array}}{\varphi}\ (\bot, m)$$

Contradiction

First-Order Logic Additional Rules

$$\frac{\begin{array}{c}\mathcal{D}\\ \forall_x \varphi\end{array}}{[\varphi]^x_t}\ (\forall_E) \qquad \frac{\begin{array}{c}\mathcal{D}\\ [\varphi]^x_y\end{array}}{\forall_x \varphi}\ (\forall_I)$$

Universal Elimination **Universal Introduction**

$$\frac{\begin{array}{cc}\mathcal{D}_1 & (\,[\varphi]^x_y)^m\\ \exists_x \varphi & \mathcal{D}_2\\ & \psi\end{array}}{\psi}\ (\exists_E, m) \qquad \frac{\begin{array}{c}\mathcal{D}\\ [\varphi]^x_t\end{array}}{\exists_x \varphi}\ (\exists_I)$$

Existential Elimination **Existential Introduction**

Building these proofs can be done in two different ways: bottom-up by starting from the conclusion and top-down by starting from the premises or open clauses. Let's prove $\{\psi\} \vdash \varphi \to (\psi \wedge \varphi)$ using a bottom-up approach (Figure 2.1), where ψ is the premise and $\varphi \to (\psi \wedge \varphi)$ is the conclusion. The first step is to apply the Introduction rule for Implication. From this rule, we derive a new hypothesis (φ), which we will mark with the number two, since we already have a premise marked with the number one. At this point in the proof, we have $\psi \wedge \varphi$, which cannot be associated with any mark, so we need to

proceed. The second step is to apply the Introduction rule for Conjunction. Subsequently, both expressions in the leaves can now be associated with marks one and two, respectively. Since the only leaves, at this point, are ψ and φ, with ψ being a premise and φ being closed by the Introduction rule for Implication, we can conclude $\{\psi\} \vdash \varphi \rightarrow (\psi \wedge \varphi)$. More generally, we say that a formula φ is a consequence of a set of formulas if there exists a derivation tree where the root is φ and all open hypotheses in the tree are premises.

$$\frac{\psi \wedge \varphi}{\varphi \rightarrow (\psi \wedge \varphi)}\ (\rightarrow_I, 2) \qquad \frac{\dfrac{\psi^1 \quad \varphi^2}{\psi \wedge \varphi}\ (\wedge_I)}{\varphi \rightarrow (\psi \wedge \varphi)}\ (\rightarrow_I, 2)$$

First step **Second step**

Figure 2.1: Example of a deduction tree proving $\{\psi\} \vdash \varphi \rightarrow (\psi \wedge \varphi)$ using a bottom-up approach.

In logic courses, students often struggle with ND exercises. The reason why this happens is the fact that some steps in the proofs are not immediately obvious, and trees can become quite large with many branches. Becoming proficient requires significant exposure and practice. Figure 2.2 illustrates a more complex example of a proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg(\varphi \vee \psi)^1 \quad \dfrac{\psi^2}{(\varphi \vee \psi)}\ (\vee_{I_l})}{\bot}\ (\neg_E)}{\neg\psi}\ (\neg_I, 2)}{\neg(\varphi \vee \psi) \rightarrow \neg\psi}\ (\rightarrow_I, 1)$$

Figure 2.2: Example of a more complex deduction tree proving $\vdash \neg(\varphi \vee \psi) \rightarrow \neg\psi$.

2.4 Proof Assistants

Proof assistants are software tools designed to help their users formalize programs or mathematical concepts and prove theorems about them [20]. Besides that, they can check step-by-step that the proof is correct according to a set of axioms and rules ensuring its correctness. Several proof assistants can also automate some steps, or even the full proof. Libraries that provide reusable theorems, definitions, and strategies can extend them, enhancing efficiency and simplifying complex proofs. Proof assistants can have a big impact on education, particularly for teaching mathematical reasoning and formal semantics [3]. This type of tool can be used in Logic, for example, to help in constructing proofs in deduction systems. Some tools have a user-friendly interface and can provide feedback. For instance, the user can navigate through the steps of the proof to see the

state on the step, can display information about the current goals, and can provide little hints/suggestions about the steps to follow.

In the following sections, we present two examples of proof assistants that can be used in ND.

2.4.1 Isabelle/HOL

Isabelle is a generic framework for interactive theorem proving. Isabelle/HOL is a large application within the generic framework that focuses on higher order logic (HOL). It includes a wide range of tools for logic-specific tasks and a large theory library [24, 5]. Isabelle/HOL is based on tactic functions that manipulate the proof state. These tactics can either solve a proof goal directly or break it into smaller subgoals. For instance, Blast is a first-order tableau prover, and Metis is a resolution prover.

Sledgehammer is an extremely powerful tool in Isabelle/HOL, which connects it with external provers by sending its problems to remote servers, increasing the efficiency of the prover. Additionally, it can automate proofs by utilizing various tactics that external provers have discovered. This automatization can be useful when combined with large proofs, as it can omit certain steps by using tactics. However, it may also hide some of the underlying reasoning behind the proof, making it harder for users to understand the intermediate steps. Since this tool cannot provide a full proof or a step-by-step resolution, it may not be suitable for developing our feedback.

However, Isabelle/HOL has tools for making counterexamples. For example, Nitpick uses a solver to systematically look for edge cases, and QuickCheck creates tests at random to test the properties of the expressions. These tools can be used in our feedback system to provide counterexamples to students, assisting them in identifying errors in their reasoning and improving their comprehension of the exercises. Figure 2.3 shows an example of how these tools are used and the corresponding counterexample found.

```
lemma "φ ⟶ (φ ∧ ψ)"
  quickcheck (*φ = True, ψ = False*)
  nitpick (*φ = True ψ = False*)
  oops
```

Figure 2.3: Example of code in Isabelle testing Nitpick and Quickcheck.

2.4.2 Lean

Lean is a functional programming language that can be used as an interactive theorem prover [30]. The structure of the proofs in this tool is very similar to the one presented in 2.3, where it is possible to use rules defined in ND, in contrast to Isabelle/HOL. Figure 2.4 shows an example of a proof in ND using Lean's style, while Figure 2.5 presents its representation in tree form.

```
theorem example1 (h : p ∨ q) : q ∨ p :=
  Or.elim h
    (fun hp : p =>
      show q ∨ p from Or.intro_right q hp)
    (fun hq : q =>
      show q ∨ p from Or.intro_left p hq)
```

Figure 2.4: Example of code in Lean proving $\{p \vee q\} \models q \vee p$.

$$\dfrac{p \vee q^{1} \qquad \dfrac{p^{2}}{q \vee p}(\vee I_l) \qquad \dfrac{q^{3}}{q \vee p}(\vee I_r)}{q \vee p}\ (\vee E, 2, 3)$$

Figure 2.5: Example of a deduction tree proving $\{p \vee q\} \models q \vee p$.

Lean also has a tool to automate proofs called Aesop [28]. Unlike Isabelle/HOL, Aesop can provide a step-by-step proof, but not in the format presented above. The generated proof uses different tactics that are genereraly not easy to directly map to ND rules. However, Aesop alows users to define their own rules/tactics to help with automation. It is possible to restrict the domain of the rules used in the automation process. Perhaps by defining the set of all ND rules, we could achieve the desired output, enabling the generation of proofs using only valid rules. If we successfully accomplish this, we will have a reliable method for implementing our feedback. Another interesting feature of Aesop is that it allows the user to use part of their proof to generate the rest of it when possible.

3

Related Work

In this section, we will present some online tools that align with the goals of our work. For each of these tools, we begin by describing their purpose and highlighting some of the key features. We then give a special focus on how they handle the feedback. Finally we conclude by discussing the positive points of each tool that we may consider when developing our solution, as well as explaining why they do not fulfill our requirements.

3.1 Iltis Web-Based System for Teaching Logic

Iltis is an interactive online tool that assists students in learning logic from its foundation [10, 11]. The goal of this tool is to provide a system that supports a wide variety of content (modal logic, PL and FOL), along with a valuable feedback system that helps the learner better understand their mistakes.

The developers of this web application divided it into multiple sections. Each section consists of a series of tasks, or exercises, that intensify in difficulty as the learner progresses through them. For each kind of task, this application provides a custom feedback generator. Feedback generators are pre-implemented pieces of code that dynamically provide various forms of feedback in tasks. This feedback can vary depending on the mistakes made by the learner. Some tasks have different levels of feedback that may differ based on the learner's proficiency. Low feedback levels provide a vaguer hint, and the high ones a more precise and explicit hint. Table 3.1 provides a list of the currently available types of exercises in Iltis.

Task	Description	Input	Output
Logical tasks			
PickVariable	Choose suitable propositional variables from a list.	–	variables $A_1, \ldots, A_m$
CreateFormula	Translate statements into formulas.	variables $A_1, \ldots, A_m$	formulas $\varphi_1, \ldots, \varphi_k$
InferenceFormula	Combine formulas $\varphi_1, \ldots, \varphi_k$ and a formula φ into a formula ψ that is unsatisfiable if and only if $\varphi_1, \ldots, \varphi_k$ imply φ.	formulas $\varphi_1, \ldots, \varphi_k$ and φ	a formula ψ
ManualTransformation	Textfield-based transformation of a formula φ into conjunctive, disjunctive or negation normal form, or into another formula.	a formula φ	the transformed formula ψ
GuiTransformation	Same as previous, but graphical user interface.	a formula φ	the transformed formula ψ
Resolution	Resolve the empty clause from the clauses of the CNF of φ.	a formula φ	–
Administrational tasks			
Questionaire task (ask a list of multiple choice questions), tasks to display messages, and a task to collect data and feedback from students.			

Table 3.1: List of tasks available in Iltis.

From the teacher's perspective, this framework provides a way to create more tasks. Teachers can achieve this by creating an XML file where they specify a set of tasks and a list of feedback generators to be presented to the learner.

3.1.1 Feedback

Feedback generators comprise the Iltis feedback system. Teachers are allowed to associate more than one feedback generator with the task, creating different levels of assistance. Some exercises rely on feedback generators constructed using reversion rules, providing better and more accurate feedback. One example of those generators is the reversion rules. They were built based on a previous study, where researchers collected some of the most frequent mistakes made by learners. A common example of a reversion rule in the "Propositional Formulas" exercises is to switch the order of the antecedent and consequent in implications. Whenever a learner switches two parameters, the feedback generator tries to apply reversion rules to find the correct solution. If successful, this indicates that the solution is close to the correct one, making it possible to provide more precise feedback based on the applied rule(s). Otherwise, it suggests that the solution is far from the correct one.

3.1.2 Conclusion

There are some positive aspects to consider from this system when developing our own tool, such as the intuitive way (it presents a low learning curve, and it is fundamental for these kinds of tools) that the exercises are presented to the learner, the advanced feedback system, and the simple access to the tool. It also provides a vast set of exercise types and a modular way to create them. On the other hand, teachers need to specify tasks in XML, and this requires some extra knowledge. Some types of exercises are still missing in this tool, like the ND. However, the biggest drawback is that it is not open-source, meaning it cannot be expanded.

3.2 Logic4Fun

Logic4Fun is an online tool with a wide range of logical problems and puzzles focused on logical modeling and formalization [21].

This tool has been under development since 2001 by an Australian university. It was projected to help students practice and develop skills in formalizing logical problems, as this is a challenging topic to teach, and learners often struggle with it.

Logic4Fun uses many-sorted first-order logic (MSFO) [1] languages to express the problems. It has a solver that takes as input a set of formulas and searches for finite

[1] Many-sorted first-order logic is an extension of FOL. In FOL, all variables come from the same domain, limiting flexibility when modeling exercises with multiple distinct domains. MSFO extends this by allowing the assignment of types (or sorts) to variables and predicates, making the language more expressive.

models of this set. This tool presents a web page with different levels of problems: Beginner, Intermediate, Advanced, Expert, and Logician, with increasing complexity. It starts with trivial exercises to help students better understand how to use the site (declare vocabulary, set constraints, and read the solver output) and progresses to more complex and challenging exercises that require a strong background in logic. One of the key advantages of using this tool is the ability to receive immediate and accurate feedback, in contrast with traditional teaching methods. This helps keep learners motivated and encourages them to invest more time and effort into solving problems. This site also allows users to enroll in a course by using the credentials provided by the teacher.

3.2.1 Feedback

Logic4Fun tracks two kinds of errors: syntactic and semantic. Syntactic errors are mistakes in the structure or arrangement of words that violate the grammar rules of the language. These errors can be captured by the parser or type checker. When a user attempts to submit an exercise with syntactic errors, a message is presented with some suggestions. When the type checker detects an error, it provides more information, especially about the expected and given types. Semantic errors are mistakes in logic or meaning in a programming language that occur in program execution. Since there are no predefined solutions, these errors are harder to classify and to deal with. Given these difficulties, Logic4fun created a diagnostic tool to provide more informative responses to the users when a solution cannot be found. The diagnostic tool uses two approaches to provide information: approximate models and unsatisfiable cores.

- In the approximate models approach, the solver starts by marking some unsatisfied constraints as "soft" and then attempts to satisfy as many as possible. Then a user can adjust constraints and rerun the solver, iterating to find the optimal approximations.

- In the unsatisfiable cores approach, the solver can try to identify groups of unsatisfied constraints that are causing the problem. Each group must contain at least one contradiction, giving useful clues for troubleshooting the problem.

3.2.2 Conclusion

Logic4Fun has several positive aspects, such as allowing exercises to be saved, enabling learners to pause their work and resume it later, progressively increasing the difficulty of exercises, helping learners integrate with the tool, and incorporating a diagnostic tool to address the lack of feedback. It has a class system where professors can invite their students to enroll. However, it only provides a restricted range of exercises based on FOL. It has some limitations with the solver's performance (the number of models presented to the user is restricted), and it is still facing issues with feedback. Sometimes, the reported errors are overly detailed or unclear, which can become frustrating for the learners.

3.3 LOGAX

LOGAX is a tool designed to help students in constructing Hilbert-style axiomatic proofs[2]. in PL [17]. This tool is capable of providing feedback and hints at different levels of the proofs. It can give the solutions for the steps to follow, as well as the complete solution to the problem.

The team behind LOGAX focused on various methods to provide better assistance to users during the exercises. One of the methods that they found entails the teacher providing a hidden solution or deriving it from a set of student solutions. However, this method has some drawbacks: it can only recognize solutions that are nearly identical to the stored proofs, it is limited to a fixed set of exercises, and every time a teacher wants to add a new exercise, they must provide a hidden solution. Another method that they found relies on Bolotov's algorithm [6], which addresses all the issues of the previous method.

Bolotov's algorithm is a proof searching algorithm for the natural deduction in FOL. For any given problem, if solvable, the algorithm terminates, either finding a corresponding natural deduction proof or giving a set of constraints, from which a counter-example can be extracted [6].

In ND, students are not strictly required to solve the exercises in one way. Proofs of the same problem can have different shapes, use different rules/axioms, and the order of the steps that the users follow can be different, as stated in Chapter 2.3. LOGAX tool has a powerful adaptation of this algorithm that covers all the drawbacks previously mentioned in the hidden solution approach, as well as the flexible characteristics of solving ND proofs. LOGAX's algorithm can adapt its solution space to better assist learners based on the user's steps. If the user takes a step in the current solution space, the algorithm can give feedback and hints directly. However, if the step diverges from the solution space, it is necessary to recalculate it, and then the system uses the new solution space to compute feedback and hints. This dynamic behavior enables the system to consistently provide feedback and hints to the user, preventing them from becoming lost in the exercise. To make this happen, the algorithm creates a Directed Acyclic Multigraph (DAM) that can hold more than one possible answer to a given proof. A better description of the algorithm and its adaptations can be seen in [17, 6]. In this DAM, vertices represent statements, and edges connect dependent statements (rule application).

Figure 3.1 shows an example of generating a DAM, where three assumptions are given (p, $p \rightarrow q$ and $q \rightarrow r$) and the goal is to prove $q \rightarrow r \vdash (p \rightarrow q) \rightarrow (p \rightarrow r)$. There are solid red edges that show how to use the Modus Ponens rule (which is the same thing as the Implication Elimination rule) and dashed blue edges that show how to use the Deduction Theorem (which is the same thing as the Implication Introduction rule). The statements $(q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$ and $p \rightarrow (q \rightarrow r) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$ are axioms. In this

[2]Axiomatic proofs are a kind of proof in formal deduction where each step of the proof is supported by axioms or inference rules previously established.

example the DAM captures three different solutions for the same proof: one that uses axioms a and b, one that uses the Deduction Theorem and axiom a and one that uses no axioms and applies the Deduction Theorem twice.

Storing this information in a DAM makes it easier to provide feedback about the steps required to complete the proof at any given level. The user can apply the rules in any order they choose. The system can adapt to the user's solution if it diverges, simply by following the sequence of rules chosen by the user. This allows the system to easily provide information about next steps (top-down proof) or previous steps (bottom-up proof) using the edges. To provide a complete solution to the user, it is necessary to extract and trim the solutions from the DAM.

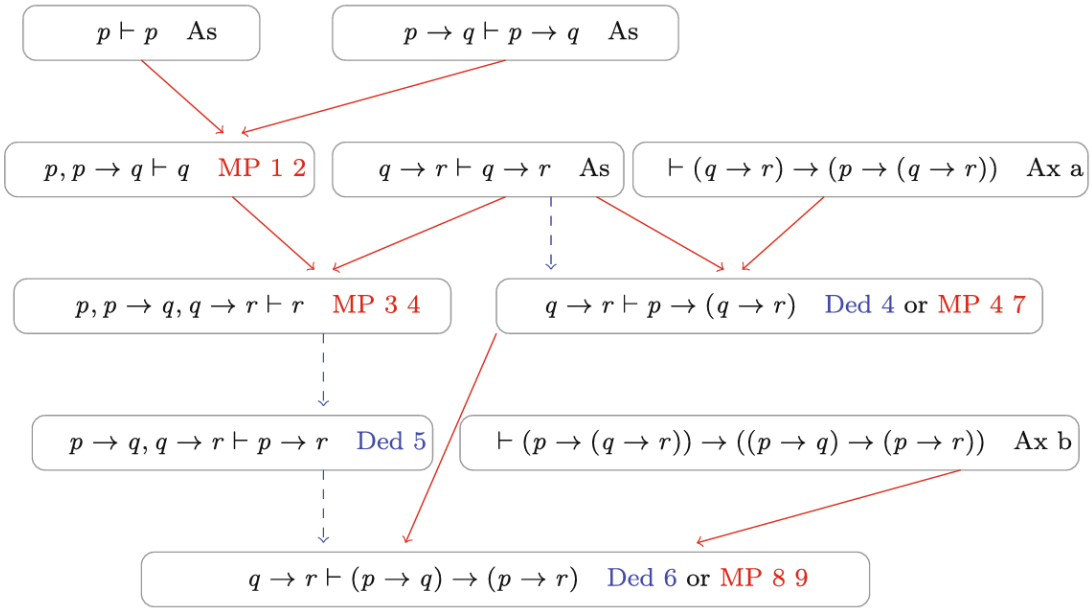

Figure 3.1: Example of a Directed Acyclic Multigraph generated by LOGAX.

LOGAX's team not only focused on the feedback but also the way they presented the exercises to the users. The design of this tool focuses on interfaces that allow students to concentrate on the goal of solving proofs rather than waste time figuring out syntactic errors. Shortening the number of steps and distractions required to reach the goal leads to better learning outcomes.

3.3.1 Feedback

The LOGAX algorithm primarily covers the feedback and hints system. It can provide the directions for the next steps, indicate the next rule to be applied, and offer an explicit step-by-step procedure for performing the next step. To make the assistance system even more powerful, the team extended it to provide not only informative support but also information about subgoals. For example, rather than trying to directly prove the conclusion, the helping system offers hints in the form of sub-proofs that lead step-by-step

to the final proof. By providing feedback that includes information about subgoals, the system can help students understand why a certain step is useful, and students are more likely to succeed in correcting mistakes.

The team also did some studies on students' common mistakes during proofs, and they came up with the following types of mistakes:

- Oversights: Correspond to syntactic errors, for example, when a user forgets to close a parenthesis in a sentence or when logical symbols are not placed in the correct position.

- Conceptual errors: These occur when a student fundamentally misunderstands a concept or its application. For example, this error can occur when choosing the statements to apply a certain rule.

- Creative rule adaptations: These occur when students try to invent their own rules. For example, from this $\neg p \rightarrow (q \rightarrow \neg r)$ and q, we can conclude $\neg p \rightarrow \neg r$ using the Modus Ponens rule. This usually happens when the student does not know how to proceed.

The helping system underwent a final adaptation to track these mistakes. This way, the system can point out a mistake and, if possible, mention exactly which formula, subformula, or set of formulas does not match the chosen rule.

3.3.2 Conclusion

For the Hilbert-style axiomatic proofs exercises, LOGAX appears to be a perfect fit. It covers a wide range of aspects to consider when developing a system like this. Starting with the algorithm that finds multiple possible solutions for a given proof, the fact that it ignores the order of the steps, and the ability to adjust the guidance based on the user's solution. It also includes different approaches for giving feedback and hints, as well as the idea of giving subgoals as hints to help the student understand the proof. Additionally, there are some important aspects regarding the design of the interfaces. Unfortunately, this is an old project, and the tool is no longer available. This project has some minor drawbacks that we can consider when developing our solution. One of them is that the system doesn't provide fading strategies to reduce the amount of feedback. We might want to control the amount of feedback sent to the student based on their level of expertise. Another problem this tool faces is that the user can not erase lines of the proof. As a consequence, the final proof can be more extensive than expected. Overall, it seems to meet all the requirements for a good tutoring system, and for sure, this tool will be used as a reference for the one that we are going to develop.

3.4 MineFOL

MineFOL is a game for learning FOL [13]. This game is very similar to a well-known game called Minesweeper. Minesweeper is a game that features a grid containing empty cells and cells with hidden mines. The objective is to locate all the hidden mines using hints provided as the player explores the grid. These hints indicate the number of mines adjacent to a given cell, helping the player deduce their locations.

MineFOL is an adaptation of that game where, instead of giving hints with the number of the nearest mines, it gives messages with FOL expressions to help locate them, for example: $\neg\exists x\, \text{mine}(1, x)$. These expressions are hidden in safe squares, and the goal is to find the maximum number of expressions to have enough information to locate all mines in as few steps as possible. The player can only travel through safe squares. A square is considered safe if it is possible to prove it using the collected messages. If the player steps out of the proven zone, the game ends. The game always starts at the top left corner of the grid($cell(1, 1)$) with an initial expression. Players always have information about how many mines there are and the maximum number of moves to solve the problem. We can combine this information with the collected expressions to extract more valuable insights. Figure 3.2 illustrates a game where the objective is to locate one mine within 35 movements.

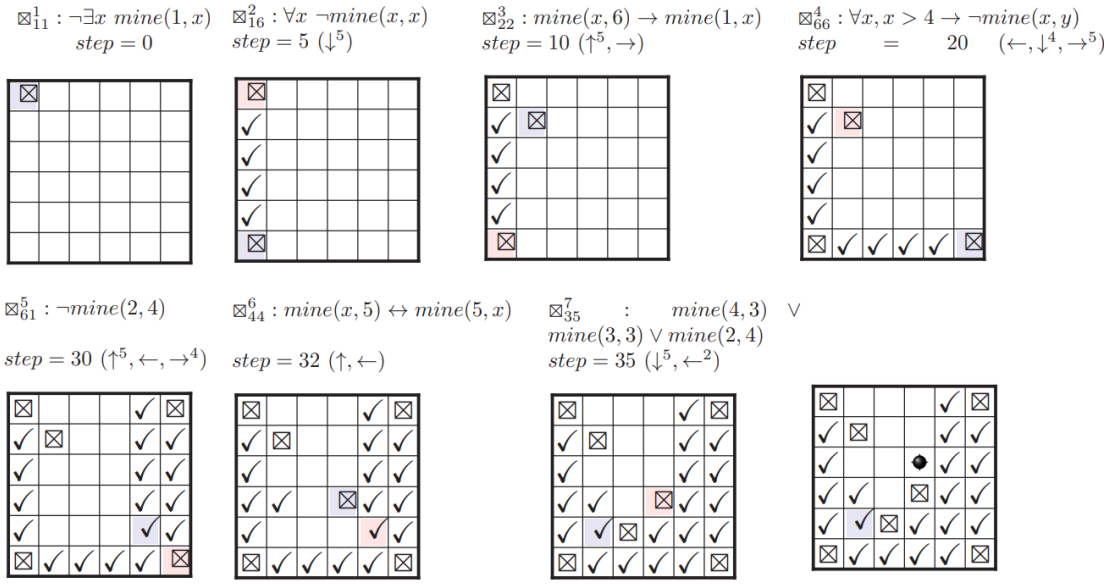

Figure 3.2: MineFOL example with one mine and 35 movements. Blue squares represent the player's current position, and the red ones represent the previous position. Cells with: ⊠ contain a message and ✓ represent safe squares.

MineFOL has three different modes:

- "Play it yourself!": In this mode the user tries to locate all mines based on a set of discovered FOL expressions. This can help the student practice reasoning in FOL,

since the student has to translate the expressions to natural language to understand the message in order to solve the game.

- "Challenge a software agent!": In this mode, the computer is responsible for solving the game. The student does not need to worry about anything. If the game is valid, the agent will find a solution for it, and they will present it to the student.
- "Create your own game!": In this mode, the user is responsible for setting up the game. Students can define their own FOL expressions, as well as the size of the grid and the number of mines. The complexity of the game can vary based on the grid size, the number of mines, the size and complexity of the FOL expressions, and the number of steps required to complete it. This mode is a good way for students to practice formalization in FOL. To check if the game is valid (can be solved), the student can challenge the software agent.

3.4.1 Feedback

This game includes a feedback system that provides assistance every time the player loses a game. When a player leaves the safe zone, the system provides an explanation of why the cell is unsafe by displaying a resolution-based proof.

3.4.2 Conclusion

MineFOL introduces a new concept that none of the previously described tools has. The concept of gamification consists of applying game mechanics and concepts in non-gaming environments. Studies show that if the learning platform is gamified, it does not only drastically increase the user enrollment but also increase user engagement throughout the course [22]. Using MineFOL, students develop skills in reasoning and formalization while playing the game. It also allows competitiveness and creativity between students by creating and challenging other students to complete their own maps. The game also has different levels of complexity, allowing students with less experience to have a chance to learn, while more advanced students can challenge themselves with harder levels to further improve their skills.

4

Advanced Feedback

Providing meaningful feedback is a complex challenge. In this section, we present the key elements that would contribute to producing meaningful feedback, along with a potential solution using an algorithm that is currently under development. This concept is still in the research phase, and some aspects remain unsolved. However, initial scripts have been developed to explore its feasibility and support some of the claims made in this text.

4.1 Feedback Components

To develop an advanced feedback system for ND exercises, we first need to define its key components. An effective system should provide relevant information to assist students in solving proofs, making the process more teaching-like. With this in mind, we identify three fundamental aspects of a well-designed feedback system:

- **Providing guidance on rule applications:** Some rule applications in ND are not obvious, making it difficult for students to progress. The system should identify such situations and suggest applicable rules. When students are unsure how to proceed, it can also provide step-by-step guidance or even a complete resolution, depending on the proficiency level.
- **Breaking proofs into smaller sub-proofs:** To simplify reasoning, the feedback should allow students to focus on smaller proofs. By dividing proofs into smaller steps, it will reduce the cognitive load and encourage incremental learning.
- **Indicating the distance to a solution:** Showing how many steps (rule applications) are needed to complete the proof helps students maintain focus and gain a clear sense of progress.

These components offer several advantages in the learning process. By providing structured and clear information, the system becomes more robust, helping students overcome challenges throughout the proof.

In the next section, we present an algorithm that is still under development and addresses the previously mentioned points.

4.2 Algorithm

Research has already started on developing an algorithm to provide the previously mentioned components. The motivation behind this algorithm is the lack of development tools that offer meaningful and powerful feedback for ND. While some tools were referenced in Section 2.4, they do not fully address the feedback components. By designing our own algorithm with a focus on these components, we gain greater control over the assistance system, ensuring a more adaptable and effective environment.

The complexity of ND proofs, with their multiple rules (as outlined in 2.3 and 2.3), presents a challenge. Some rules generate multiple branches, while others introduce new elements into the proof. Developing an algorithm to handle this complexity is not a simple task. The algorithm we present is based on the Bounded Model Checking (BMC) algorithm [4], but with a different purpose. BMC is traditionally used in verification, where it checks whether a system violates a given property by searching for counterexamples within a finite execution trace. It encodes the system and property into a SAT[1] formula and verifies correctness within a bounded number of steps. In contrast, our version of the algorithm shifts the focus from verification to structured solution generation. Instead of identifying counterexamples, we use the algorithm to generate a bounded graph of states that will store solutions within a certain range.

In the graph, each node represents a distinct state within the proof. By state, we refer to the logical expressions in the ND proof, which, at different stages, can have available different sets of hypotheses. Below is an example where, in step **A** (at the beginning of the proof), the expression $a \to (a \vee b)$ has no hypotheses available. In contrast, in step **B** (later in the proof), the same expression has $\neg(a \to (a \vee b))$ as a hypothesis.

$$\dfrac{\dfrac{\textbf{B.}\quad a \to (a \vee b) \qquad \neg(a \to (a \vee b))^{1}}{\bot}\,(\neg E)}{\textbf{A.}\quad a \to (a \vee b)}\,(\bot, 1)$$

Each edge represents the application of a rule, transitioning from one state to another. Some edges have constraint expressions, which means the algorithm can only move to the next stage if the current state contains the corresponding constraint in its set of hypotheses (represented in red). Other edges can generate new hypotheses that are added to the set of hypotheses (represented in blue). Figure 4.1 illustrates an example of a graph that can be used to prove $a \to (a \vee b)$.

There are potential questions that may arise. The first one might be: How do we build the graph? To build the graph, we first need to compute the different states. For that, we will need to apply rules successively to the expression we want to prove, generating hypotheses at each step. Then, starting with the hypotheses previously generated and

[1] SAT (Satisfiability Problem) is a well-known problem in logic, where the goal is to determine if there exists a valuation that makes a given boolean expression satisfiable [25]. SAT formulas are logical expressions typically written in Conjunctive Normal Form (CNF), which is a conjunction of disjunctions.

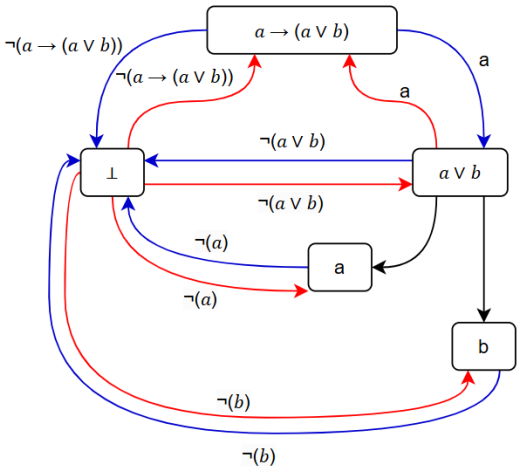

Figure 4.1: Graph to prove $a \to (a \vee b)$.

using the provided premises, we will attempt to decompose them further by applying the rules again and breaking them down into smaller components until we find a point of connection between the derived expressions. As we proceed with this process, we can begin constructing the state machine, using the rules applied to generate the new states.

This could lead to the question, "How will we extract a solution from the graph?" To find a solution, we need to traverse the entire graph by testing all combinations of paths starting from the conclusion and verifying if, using the list of hypotheses in each state, we can close the current node. This can be a computationally intensive operation since we have loops and may pass through the same operations multiple times. To address this, we will define a depth limit for the proofs, selecting an appropriate value based on the typical size of proofs. Increasing this limit unnecessarily would lead to an exponential growth in the number of possible combinations. There are also ways to avoid looping through unnecessary paths, which can significantly increase the efficiency of the algorithm. A possible solution for $a \to (a \vee b)$ using the graph in 4.1 would be $\boxed{a \to (a \vee b)} \Rightarrow \boxed{a \vee b} \Rightarrow \boxed{a}$, since we can close a using the hypothesis that we collect from transitioning from $\boxed{a \to (a \vee b)} \Rightarrow \boxed{a \vee b}$.

Another question that may come up is, "How will this algorithm adapt to the user's solution?". To handle that, we first need to check if the set of the student's derived expressions aligns with those generated by the algorithm. If there is a mismatch, we will expand the graph by adding the new expressions and derivations from those expressions, in the same way the graph was initially constructed. Once the graph is updated, we can select one of the open leaves in the student's proof and query the algorithm by providing the leaf and its state in the student's solution.

However, there are certain challenges that we have not yet fully addressed, such as how to handle rules that generate multiple branches, like the Disjunction Elimination rule. Despite these challenges, we believe this is a solid starting point for developing an effective algorithm. The algorithm is sound, meaning it will only generate valid proofs,

but it is not complete due to the imposed restrictions on proof size and the set of derived expressions.

If we can efficiently implement this algorithm and incorporate rules that generate multiple branches, we will be able to create a highly effective feedback and hint system. This algorithm will address all the key components previously mentioned. Since it can adapt to the user's solution and find solutions, it can be used to provide guidance on rule applications, assist in breaking down the proof into sub-goals, and indicate the distance to a solution, as the algorithm is aware of the steps required to complete the proof.

As mentioned earlier, this is still a concept under study, and some of the topics discussed remain somewhat abstract.

4.2.1 Results

In this section, we present some results of the key features of our current algorithm. We will use the proof $(P \to (Q \to R)) \to (P \to Q) \to (P \to R)$ as an example to illustrate these features.

- **Query full proofs:** The algorithm can be used to find multiple possible solutions to a given problem. For example, Figure 4.2 shows the output from running the algorithm, and Figure 4.3 displays the corresponding deduction tree. In the output, each line represents a formula from the tree, with the TAB indentation indicating the tree's structure. In our current implementation, we are not keeping track of the rule names, but in the output we can still deduce which rule was applied by examining the hypotheses at each expression and the number of children. The algorithm uses Breadth-First Search (BFS)[2] to find potential solutions. The characteristics of BFS ensure that the output always returns the shortest solution to the problem.

```
Solvable: true
#1
(p → (q → r)) → ((p → q) → (p → r))   hypotheses:[]
    (p → q) → (p → r)   hypotheses:[p → (q → r)]
        p → r   hypotheses:[p → q, p → (q → r)]
            r   hypotheses:[p, p → q, p → (q → r)]
                q → r   hypotheses:[p, p → q, p → (q → r)]
                    p → (q → r)   hypotheses:[p, p → q, p → (q → r)]
                    p   hypotheses:[p, p → q, p → (q → r)]
                q   hypotheses:[p, p → q, p → (q → r)]
                    p → q   hypotheses:[p, p → q, p → (q → r)]
                    p   hypotheses:[p, p → q, p → (q → r)]
```

Figure 4.2: Algorithm proving $(P \to (Q \to R)) \to (P \to Q) \to (P \to R)$.

[2] Breadth-First Search (BFS) is an algorithm used to traverse or search through graphs and trees. It explores the nodes level by level, starting from the root or source node, visiting all nodes at the current depth before moving on to the next level.

$$\dfrac{\dfrac{\dfrac{\dfrac{[P]^1 \quad [P \to (Q \to R)]^3}{Q \to R} \to\text{Elim} \quad \dfrac{[P]^1 \quad [P \to Q]^2}{Q} \to\text{Elim}}{R} \to\text{Elim}}{P \to R} \to\text{Intro}^1}{(P \to Q) \to (P \to R)} \to\text{Intro}^2}{(P \to (Q \to R)) \to (P \to Q) \to (P \to R)} \to\text{Intro}^3$$

Figure 4.3: Deduction tree for $(P \to (Q \to R)) \to (P \to Q) \to (P \to R)$.

- **Query parts of the proof:** The algorithm can also be used to query specific parts of the proof by providing the state we wish to complete. For instance, in Figure 4.4 we want to finish the proof, but we do not know how to proceed. By querying the algorithm with our current state in this case the expression R with the hypotheses $\{P \to (Q \to R), P \to Q, P\}$ the algorithm can find solutions for that state. Figure 4.5 shows the result of this query.

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{R} \to\text{Intro}^1}{P \to R} \to\text{Intro}^2}{(P \to Q) \to (P \to R)} \to\text{Intro}^3}{(P \to (Q \to R)) \to (P \to Q) \to (P \to R)}$$

Figure 4.4: Incomplete proof.

```
Solvable: true
#1
r   hypotheses:[p → q, p, p → (q → r)]
    q → r   hypotheses:[p, p → q, p → (q → r)]
        p → (q → r)   hypotheses:[p, p → q, p → (q → r)]
        p   hypotheses:[p, p → q, p → (q → r)]
    q   hypotheses:[p, p → q, p → (q → r)]
        p → q   hypotheses:[p, p → q, p → (q → r)]
        p   hypotheses:[p, p → q, p → (q → r)]
```

Figure 4.5: Result of querying the algorithm using expression R and hypotheses $\{P \to (Q \to R), P \to Q, P\}$.

5

Proposed Work

Finally, in the last section, we will outline our approach to tackle this project. We begin by describing and justifying our strategy, including the chosen approach and the technologies we will use. Then, we present our work plan, discussing ideas that may be useful for the development of the project. This section is divided into multiple iterations, which will span the duration of our thesis, including our evaluation plan, which will help ensure the success of the project.

5.1 Technical Approach

As mentioned in 1.2, the aim of this thesis is to design and implement an interactive tool to practice and evaluate logic exercises. The idea is to provide an online environment that will be easily accessible for everyone, where students can practice what they have learned in classes and be evaluated, while teachers can provide exercises for assessment. Unlike most online tools for learning logic, this one will pay special attention to the way it delivers assistance. We want to implement a system that is suitable for everyone, starting with the student who has no prior knowledge about logic to the expert who wants to improve his skills even more.

To define our approach, we first need to select a set of exercises to implement. In logic, there are numerous types of exercises. According to professor Ricardo Gonçalves, who is currently the coordinator of the Computational Logic course at NOVA FCT, the exercise where students show more difficulties is the ND proofs. As explained in Chapter 2.3, this type of exercise requires significant exposure and practice to master. This exercise is also important, as it constitutes a significant part of the course at NOVA and is present in all tests and exams. Therefore, this tool will initially focus on implementing ND exercises, but it will be designed to allow for future expansion.

ND proofs can be presented in many different ways, but we will work with a tree structure. These proofs can be done in both PL and FOL, so we will split this exercise into two different ones, with the second being an extension of the first, as it includes more rules and symbols. Besides that, we will have to establish a way to develop an effective and

impactful feedback system that not only reports the students' mistakes but also provides mechanisms to assist them when they do not know how to progress.

Some of the work related to the assistance system has already begun with the study and development of scripts to create an algorithm capable of generating multiple proofs for a given ND problem. With such a system, we would be able to build an extremely powerful feedback mechanism, providing step-by-step guidance for proofs (helping students avoid getting stuck), offering hints about sub-goals (simplifying the problem by breaking it into smaller parts), and indicating how far the student's solution is from a possible correct one. A more detailed description of the feedback features and the algorithm has already been presented in Chapter 4.

5.1.1 Architecture and Technologies

To start with, we need to determine which architecture is most suitable for the development of a tool with these characteristics. The three-tier architecture is the one that best fits the requirements. Online tools scale significantly, especially this type of tool, where the number of simultaneous users is highly unpredictable [16]. This architecture, by separating concerns into different layers, makes it easier to replicate and maintain. It also improves system reliability by facilitating the implementation of multiple levels of redundancy while providing ease of deployment. This architecture is divided into three tiers:

Data (Database): The data tier stores data in a persistent format. In our tool, this tier will be responsible for storing information about students, such as their proficiency level and their completed or in-progress proofs. It will also store exercises provided by teachers, as well as the grades for these exercises. For this, we will use a MySQL database [29], as it is relational, and the data we intend to store is also relational. Additionally, MySQL's ability to handle simultaneous operations from numerous users makes it suitable for online platforms with large user bases [14]. It also provides better support for transactions and data integrity, which is crucial for Systems used to evaluate students.

Application (Server): The application tier is the brain of the system. Not only is it responsible for communicating with the other tiers, but it also contains all the core functionality behind the tool. In this tier, we will implement the logic for the ND exercises, the mechanism that manages users and provides access to different resources, and the system that generates the feedback. The technology that is ideal for us to use is the Spring Framework [32]. Spring simplifies web application development, leading to increased productivity. It provides a decoupled way of developing web applications, making the process simpler through its various features [23]. Support for REST API[1] is one of the key features offered by Spring, facilitating communication with the presentation layer via HTTP requests and responses, enabling data exchange between the other two tiers of the

[1] Representational State Transfer Application Programming Interface (REST API) is a popular architectural style for creating efficient and scalable web services [19]. It has become an important component of modern information technology ecosystems, facilitating interoperability among diverse Systems and platforms.

application. Additionally, it offers built-in features to simplify communication with the database and to handle transactions.

Presentation (Website): The presentation tier is where users normally interact with the system, typically through a web page. Here, we will implement distinct environments for the different users that will be using the system. For this layer, we will use the React framework [31]. It uses a declarative and object-oriented approach, which makes the code more intuitive and easier to maintain [18]. React's component-based architecture in hierarchy facilitates the creation of modular and reusable code by allowing developers to build complex pages from smaller, self-contained pieces. Another key feature of React is its ability to efficiently update user interfaces in response to state changes, which is a particularly useful feature when developing an interactive tool. Another technology that we will be using is Redux [9]. It provides a centralized data store outside of the React component hierarchy, reducing system complexity by eliminating the need to pass state through multiple levels of components. This approach results in a more natural and organized application structure, making the state easier to manage and maintain.

5.2 Work Plan

An iterative model approach will guide the project's development process that focuses on continuous improvement. We will divide the implementation of this tool into four iterations. The first iteration will comprise the basic features of an online tool with the ND exercises. However, it will not include a feedback system. The second iteration will have a basic feedback system that will adjust according to the students proficiency. The third iteration will include an advanced feedback system that will be able to guide the student throughout the exercises. Each iteration will culminate in a final product that the students and teachers can test and use. The idea behind this is to always have a safe checkpoint throughout the development if something goes wrong. Since the feedback topics are harder to implement, we will keep them separated in two different iterations. Each iteration will involve conducting different evaluation processes to ensure the success of our project. In the final iteration, we will integrate our tool with the Moodle online e-learning platform. The sections below will present the different iterations that are divided into server and website implementations.

5.2.1 First Iteration

By the end of this first iteration, we aim to have a fully functional system that supports ND exercises. Teachers will be able to provide new exercises in both languages, as well as view the answers and evaluations of the exercises. Students will be able to construct and validate their proofs. There will not be any detailed feedback in this iteration. The only feedback provided to the student will be a message indicating whether the proof is correct. This approach allows us to focus on the essential functionality of the system

without the complexity of providing in-depth feedback at this stage. Future iterations may incorporate more detailed evaluations to enhance the learning experience.

5.2.1.1 Server

Our first task will be the implementation of a parser[2] for both PL and FOL expressions according to the grammar presented in 2.1 and 2.2. After implementing the parsers, we will conduct a battery of tests to guarantee their correctness.

Moreover, we will also implement a parser for ND expressions. By defining a language to represent our own proofs, we can easily detect structural errors made by students. These steps are very important for setting up the feedback system because they can be used to give detailed help on syntax, like showing where mistakes happened and suggesting other ways to fix them. Similarly, the proof parser will undergo extensive testing.

The next step will be implementing the proof interpreter. This interpreter will take a well-formed proof as input and verify its correctness. It will begin by checking whether all derivations from the rules are correct. Then, it will verify the hypotheses and marks used in the proof, ensuring there are no unmarked leaves or missing hypotheses in the tree. Like the parsers, the interpreter will also go through extensive testing to ensure correctness.

Finally, we will implement a system to manage users and the system's exercises. There will be two types of users: students, who will have access to exercises and their grades, and teachers, who will create exercises and view grades of those exercises. This will require creating database tables to store user and exercise information, as well as defining the REST API endpoints that will interact with the webpage.

5.2.1.2 Website

After developing the server, we will move on to the implementation of the website. We will begin by defining the pages for students and teachers.

Focusing only on the ND proof environment, the goal is to create an intuitive and interactive platform for students to practice. To achieve this, proofs will be built using building blocks, with each block representing a sub-tree. The system will present users with a board on which they can add and edit blocks. Blocks can also be dragged and merged to form larger ones, resulting in bigger proofs solving the issue of directional flexibility discussed in Chapter 2.3. At this stage, since we do not have a feedback system implemented, we will provide the list of available rules as blocks, so the users do not need to define it manually. Figure 5.1 and Figure 5.2 both show a prototype of the block-based construction environment. The first uses a small proof, while the second uses a large one.

[2] A parser is a software component used in programming language processing that analyzes and interprets the structure of code or text according to a defined grammar [1].

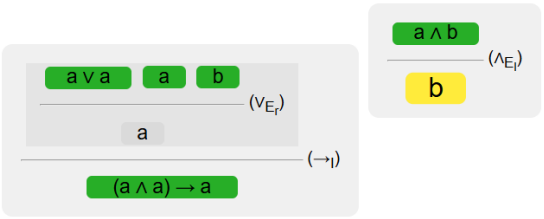

Figure 5.1: Building blocks prototype with a simple proof.

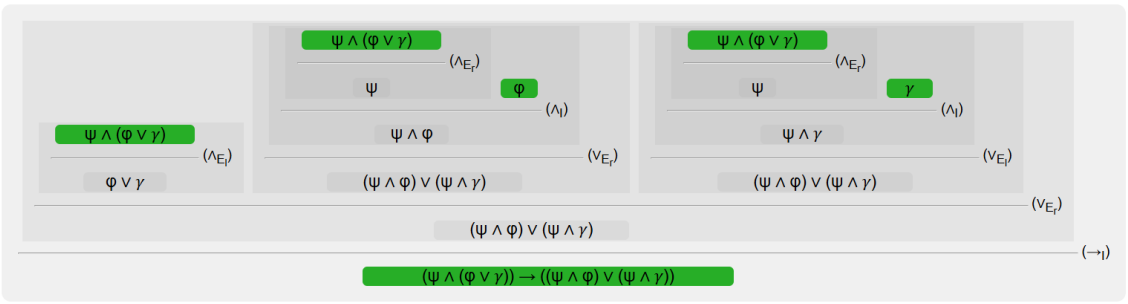

Figure 5.2: Building blocks prototype with a big proof.

5.2.2 Second Iteration

By the end of this iteration, we aim to have a tool with a simplified feedback system, featuring different levels based on the students' proficiency. The system will identify both syntax and semantic errors in the submitted proofs.

5.2.2.1 Server

In this iteration, we will start implementing the feedback system, focusing on providing simple assistance. The system will provide balanced support to prevent students from relying solely on corrections. At the same time, it will make sure users have guidance so they know how to process it.

To adjust the feedback to each student's level, we will assign a proficiency score, similar to what Iltis does. New users will start with a low score, which will increase as they complete exercises or decrease as they make mistakes. Higher scores will lead to more general feedback, while lower scores will provide more detailed help. To compute the score, we will categorize the exercises into different levels (e.g., easy, medium, and hard) as well as classify the mistakes (e.g., critical, moderate, and minor). This iteration will focus on syntax errors, which are related to the structure of expressions and proof grammar, and semantic errors, which concern the internal logic of the proof. Below are some examples of feedback for a syntactic error, based on proficiency levels:

- **High proficiency:** "Your proof contains invalid expressions."
- **Intermediate proficiency:** "Expression $a \vee b \rightarrow c$ is not valid."

- **Low proficiency:** "Expression $a \vee b \rightarrow c$ is ambiguous. Consider including parentheses, possible correction: $(a \vee b) \rightarrow c$."

5.2.2.2 Website

The feedback system will be fully integrated into the website and presented in a clear and interactive manner. The system will adapt dynamically based on the user's proficiency level. To achieve this, we will create distinct environments tailored to each user's proficiency level. For example, students with low scores will be presented with detailed information regarding the proof's current state, similarly to what some proof assistants do. Additionally, rules will be auto-filled to minimize cognitive load, and hypotheses will be automatically added to the list of hypotheses. As users progress and their proficiency increases, these assistive features will be gradually removed, requiring users to input rules manually and encouraging independent thinking. Another way to provide feedback is through visual changes in parts of the page. Visual feedback mechanisms, such as highlighting invalid expressions or marking tree segments with errors, will also be implemented to improve the user experience. Figure 5.3 illustrates this feedback by highlighting the incorrect part of the proof.

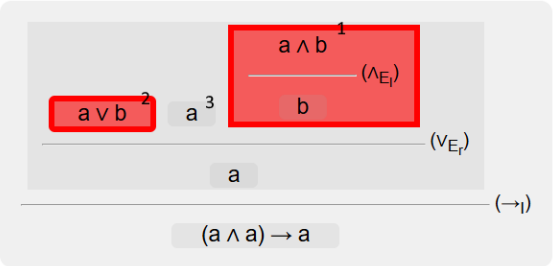

Figure 5.3: Building blocks feedback prototype.

5.2.3 Third Iteration

By the end of the third iteration, we will refine the feedback system, supporting advanced error detection and user interactions. We will introduce features such as displaying full proofs as possible solutions and providing hints about goals and the steps to follow during the proofs. Additionally, the interface will be polished, and the entire system will undergo extensive testing to ensure stability and reliability.

5.2.3.1 Server

In this iteration, we will start by defining the mechanism that will guide the student throughout the proof. Currently we found three possible ways to do that.

The first method involves the teacher providing a hidden solution while creating the exercise. This method encounters several challenges, including the inability to adjust to the student's solution and the constant need for a hidden solution.

Another solution is to use the Lean functional programming language [30] to automate this process for us. We can define our own set of rules and use Aesop [28] for this purpose. While this approach seems straightforward, defining some of the rules from ND (Sections 2.3 and 2.3) and mapping our code to the Lean programming language can be challenging. Moreover, Aesop is not particularly powerful, as it can only find the more obvious proofs.

A final method is to develop an algorithm that attempts to find multiple solutions for the same proof, similar to what LOGAX does. However, since LOGAX focuses on a different domain of proofs and the rules provided by Bolotov's algorithm differ from the ones we will use, we need to create our own version of the algorithm. We have already developed some ideas on how to implement this algorithm, which are presented in Chapter 4. By defining our own algorithm, we can create a fully controllable environment for the assistance system.

5.2.3.2 Website

Lastly, in this iteration, we will integrate advanced feedback into our webpage. This will involve implementing a system that highlights parts of the student's solution that could lead to dead ends or blocks that are irrelevant to the proof. Additionally, we will need to find an appropriate way to display feedback, either providing a full proof or partial hints in the form of sub-goals when the student gets stuck.

5.2.4 Integration with Moodle

The final iteration will be the integration of the tool with the Moodle online e-learning platform, allowing seamless access for students and teachers. This integration will enable features like automated grading and real-time tracking of student progress, ensuring a more interactive and efficient learning experience.

5.2.5 Evaluation Plan

To ensure the success of our project, we will conduct an evaluation process, focusing on key aspects such as correctness, usability, interactivity, and feedback effectiveness. We will use verification tests to ensure the system operates as intended and meets its specifications. We will use validation tests to assess the usability and interactivity of the tool, ensuring that it provides an intuitive experience for students. Additionally, we will evaluate the feedback mechanisms to ensure they provide meaningful and constructive assistance to users and to guarantee that they do not get stuck.

Verification tests will be conducted using the JUnit framework [27] with extensive test cases. This will be done in the early stages of the project's development, particularly when developing the parsers and the interpreter, and also immediately after each iteration to ensure the correctness of the system.

For the validation tests, we will focus on the usability and interactivity of the tool to guarantee that the system is user-friendly and has a low learning curve. To do this, we will organize testing sessions with participants of varying levels of proficiency in logic to interact with the system. During these sessions, we will observe how the users interact with the tool and assess their ability to use it efficiently. At the end of the sessions, we will conduct a survey to collect user feedback on the system's usability, the clarity of the feedback provided, and their overall experience with the tool. This will help us identify areas for improvement and ensure that the tool meets the needs of the users. These tests will occur right after each iteration.

5.3 Gantt Chart

Below is the Gantt chart presenting all the stages previously mentioned in our plan.

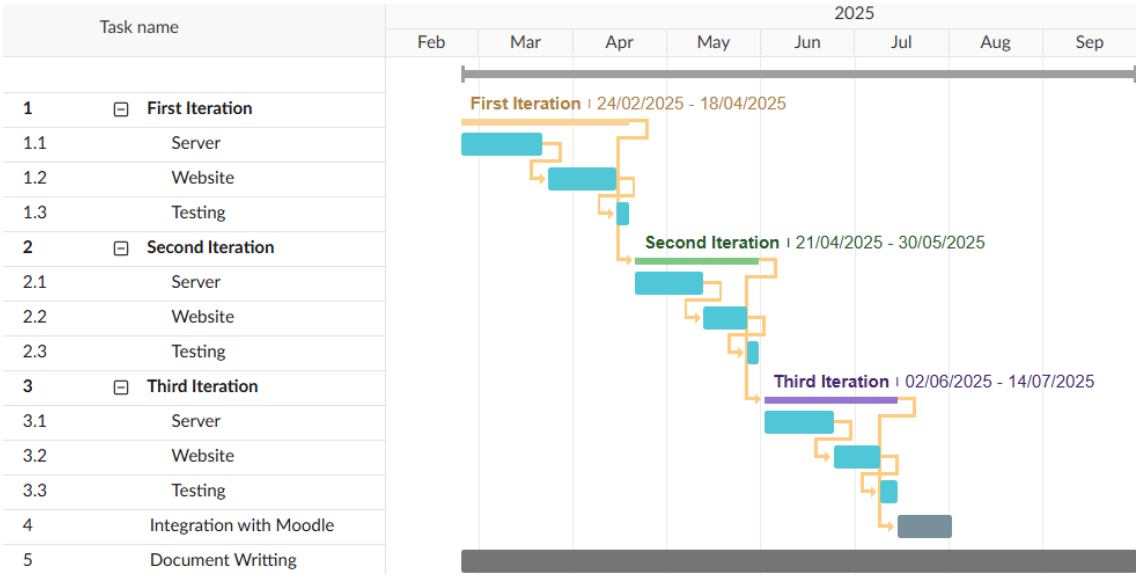

Figure 5.4: Proposed Gantt Chart for the work plan.

Bibliography

[1] A. Afroozeh and A. Izmaylova. "One parser to rule them all". In: *HAL (Le Centre pour la Communication Scientifique Directe)* (2015-10), pp. 151–170. DOI: 10.1145/2814228.2814242 (cit. on p. 30).

[2] N. Alhazzani. "MOOC's impact on higher education". In: *Social sciences & humanities open* 2.1 (2020), p. 100030. DOI: 10.1016/j.ssaho.2020.100030 (cit. on p. 1).

[3] E.-I. Bartzia, A. Meyer, and J. Narboux. *Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis*. 2023. URL: https://www.semanticscholar.org/paper/Proof-assistants-for-undergraduate-mathematics-and-Bartzia-Meyer/caeca647fd9b051fabb186b42853d1bd50d6e6f8 (cit. on p. 11).

[4] A. Biere. "Bounded model checking". In: *Handbook of satisfiability*. IOS press, 2021, pp. 739–764 (cit. on p. 23).

[5] J. C. Blanchette, L. Bulwahn, and T. Nipkow. "Automatic Proof and Disproof in Isabelle/HOL". In: *Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings*. Ed. by C. Tinelli and V. Sofronie-Stokkermans. Vol. 6989. Lecture Notes in Computer Science. Springer, 2011, pp. 12–27. DOI: 10.1007/978-3-642-24364-6_2 (cit. on p. 12).

[6] A. Bolotov et al. "Automated First Order Natural Deduction". In: *Proceedings of the 2nd Indian International Conference on Artificial Intelligence, Pune, India, December 20-22, 2005*. Ed. by B. Prasad. IICAI, 2005, pp. 1292–1311 (cit. on p. 17).

[7] K. B. Bruce et al. "Panel: logic in the computer science curriculum". In: *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 1998, Atlanta, Georgia, USA, February 26 - March 1, 1998*. Ed. by J. Lewis et al. ACM, 1998, pp. 376–377. DOI: 10.1145/273133.274341 (cit. on p. 1).

[8] A. P. Cavalcanti et al. "An Analysis of the use of Good Feedback Practices in Online Learning Courses". In: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)* 2161-377X (2019), pp. 153–157. URL: https://api.semanticscholar.org/CorpusID:201832980 (cit. on p. 2).

[9] A. Freeman. "Using a Redux Data Store". In: *Apress eBooks* (2019-01), pp. 531–559. DOI: 10.1007/978-1-4842-4451-7_19 (cit. on p. 29).

[10] G. Geck et al. *Iltis: Learning Logic in the Web*. 2022. arXiv: 2105.05763 [cs.LO]. URL: https://arxiv.org/abs/2105.05763 (cit. on pp. 2, 14).

[11] G. Geck et al. "Introduction to Iltis: an interactive, web-based system for teaching logic". In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: Association for Computing Machinery, 2018, pp. 141–146. ISBN: 9781450357074. DOI: 10.1145/3197091.3197095 (cit. on pp. 2, 14).

[12] P. Gouveia and F. Dionísio. *Lógica Computacional Capítulo 1 -Lógica Proposicional* (cit. on pp. 4, 6, 8).

[13] A. Groza, M. M. Baltatescu, and M. Pomarlan. "MineFOL: a Game for Learning First Order Logic". In: *16th IEEE International Conference on Intelligent Computer Communication and Processing, ICCP 2020, Cluj-Napoca, Romania, September 3-5, 2020*. Ed. by S. Nedevschi, R. Potolea, and R. R. Slavescu. IEEE, 2020, pp. 153–160. DOI: 10.1109/ICCP51029.2020.9266174 (cit. on p. 20).

[14] C. Gyorödi et al. "A Comparative Study Between the Capabilities of MySQl Vs. MongoDB as a Back-End for an Online Platform". In: *International Journal of Advanced Computer Science and Applications* 7 (2016). DOI: 10.14569/ijacsa.2016.071111 (cit. on p. 28).

[15] M. Huth and M. Ryan. *Logic in Computer Science*. 2004 (cit. on pp. 4, 6–8).

[16] M. Kircher and P. Jain. *The Three-Tier Architecture Pattern Language Design Fest*. European Conference on Pattern Languages of Programs, 2025. URL: https://www.semanticscholar.org/paper/The-Three-Tier-Architecture-Pattern-Language-Design-Kircher-Jain/861943e9e8da9799cba4819242a6e566c7b2e545 (cit. on p. 28).

[17] J. Lodder et al. "Generation and Use of Hints and Feedback in a Hilbert-Style Axiomatic Proof Tutor". In: *Int. J. Artif. Intell. Educ.* 31.1 (2021), pp. 99–133. DOI: 10.1007/S40593-020-00222-2. URL: https://doi.org/10.1007/s40593-020-00222-2 (cit. on p. 17).

[18] M. Madsen, O. Lhoták, and F. Tip. "A Semantics for the Essence of React". In: *34th European Conference on Object-Oriented Programming (ECOOP 2020)*. Ed. by R. Hirschfeld and T. Pape. Vol. 166. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 12:1–12:26. DOI: 10.4230/LIPIcs.ECOOP.2020.12 (cit. on p. 29).

[19] D. Prasetyawan and P. D. Rahmanto. "Pengembangan Sistem Seleksi Proposal Penelitian Berbasis Web Service Menggunakan REST API". In: *JTIM Jurnal Teknologi Informasi dan Multimedia* 6 (2024-09), pp. 283–295. DOI: 10.35746/jtim.v6i3.585 (cit. on p. 28).

[20] A. Schlichtkrull. "Formalization of Algorithms and Logical Inference Systems in Proof Assistants". In: *Thirteenth Scandinavian Conference on Artificial Intelligence - SCAI 2015, Halmstad, Sweden, November 5-6, 2015*. Ed. by S. Nowaczyk. Vol. 278. Frontiers in Artificial Intelligence and Applications. IOS Press, 2015, pp. 188–190. DOI: 10.3233/978-1-61499-589-0-188 (cit. on p. 11).

[21] J. Slaney. "Logic for fun: an online tool for logical modelling". In: *Journal of applied logics* 4.1 (2017), pp. 171–192 (cit. on pp. 2, 15).

[22] A. Vaibhav and P. Gupta. "Gamification of MOOCs for increasing user engagement". In: *2014 IEEE International Conference on MOOC, Innovation and Technology in Education (MITE)*. IEEE. 2014, pp. 290–295. DOI: 10.1109/MITE.2014.7020290 (cit. on p. 21).

[23] R. Wali and P. S. Kumar M. "Rapid Web Application Development Using Spring Framework: A Case Study". In: *International Journal of Innovative Research in Computer Science and Technology* 7 (2019-05), pp. 104–107. DOI: 10.21276/ijircst.2019.7.3.14 (cit. on p. 28).

[24] M. Wenzel, L. C. Paulson, and T. Nipkow. "The isabelle framework". In: *Theorem Proving in Higher Order Logics: 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings 21*. Springer. 2008, pp. 33–38 (cit. on p. 12).

[25] J. You et al. "G2SAT: Learning to Generate SAT Formulas". In: *Advances in neural information processing systems* 32 (2019), pp. 10552–10563. URL: https://api.semanticscholar.org/CorpusID:202767882 (cit. on p. 23).

[26] K. Zhu, W. Khern-am-nuai, and Y. Yu. "Any Feedback is Welcome: Peer Feedback and User Behavior on Digital Platforms". In: *SSRN Electronic Journal* (2022). URL: https://api.semanticscholar.org/CorpusID:247248739 (cit. on p. 2).

Webography

[27] JUnit. *JUnit 5*. Junit.org. 2018. URL: https://junit.org/junit5/ (cit. on p. 33).

[28] leanprover-community. *GitHub - leanprover-community/aesop: White-box automation for Lean 4*. GitHub. 2021. URL: https://github.com/leanprover-community/aesop (cit. on pp. 13, 33).

[29] Oracle. *MySQL*. Mysql.com. 2024. URL: https://www.mysql.com/ (cit. on p. 28).

[30] *Programming Language and Theorem Prover — Lean*. lean-lang.org. URL: https://lean-lang.org/ (cit. on pp. 12, 33).

[31] M. O. Source. *React*. react.dev. 2024. URL: https://react.dev/ (cit. on p. 29).

[32] Spring. *Spring Projects*. Spring.io. 2019. URL: https://spring.io/projects/spring-boot (cit. on p. 28).