Education-oriented Proof Assistant Based on Calculational Logic: Proof Theory Algorithms and Assessment Experience

Federico Flaviani
Universidad Simón Bolívar, Departamento de Computación y T.I.,
Caracas, Venezuela, 1080-A,
fflaviani@usb.ve

and

Walter Carballosa
Florida International University, Mathematics and Statistics,
Miami, USA, 33199
wcarball@fiu.edu

Abstract

This work presents an interactive proof assistant, based on Dijkstra-Scholten logic, aimed at teaching logic and discrete mathematics in higher education. The assistant interface is web and easy to use since inferences can be made just with the mouse. The educational experience is presented showing a correlation between the grades of the assessments in class and those made with the application web. Additionally, an algorithm proof theory for the Disjktra-Scholten system are done and the following algorithms are shown: a versatile printing algorithm that allows the administrator to configure the symbols of a theory, by assigning the desired presentation with LaTeX; an algorithm, based on Broda and Damas combinators, for generate monotonic or anti monotonic inferences in the Dijkstra-Scholten logic; an algorithm to generate the proofs of dual theorems in Boolean Algebra theory.

Keywords: Algorithm Proof Theory, Calculational Logic, Interactive Theorem Prover, Education, Propositional Logic, Boolean Algebras, Finite Difference, Combinatory Logic, Broda and Damas Combinators.

1 Introduction

A proof assistant is a program that assists mathematicians in the production of proofs, which are verified mechanically by the computer. A proof assistant can produce proofs with interactive help of the user or completely automated.

CalcLogic is an interactive web prover, based on the Dijkstra-Scholten Calculational Logic [1], a formal system popularized thanks to the book [2]. The user interacts with CalcLogic making one inference at a time. For each user inference the application notifies an error (if exists), or performs the inference printing its result. Each inference is printed according to the Dijkstra-Scholten vertical notation (Section 4).

Calculational logic works based on the replacement of an expression or subformula by another equal or equivalent. Therefore, the decision to implement an assistant based on Calculational Logic is because the proof process is the same as the way in which algebra exercises are solved, in the first years of secondary education. In this way, it is desired that students associate their already existing skills in algebraic manipulation, with said proof mechanism through the assistant.

CalcLogic was fully implemented at Simón Bolívar University during the COVID 19 pandemic as a solution for online assessments, avoiding cheating between students [3]. The situation forced a fast implementation, therefore a simple tree data structure was used to represent the proofs, and an applicative untyped language to represent the formulas. Despite the fact that formulas are being written with an untyped language, the variety of theories that this assistant supports were enough to cover some discrete mathematics

content during pandemic. Instances of CalcLogic for Propositional Logic and Boolean Algebras have been used in an educational experiment about using a proof assistant in Discrete Math courses at the Florida International University (FIU). The results of the experiment, up to Spring 22, has been presented in the 2022 XLVIII Latin American Computer Conference (CLEI) [4].

The current version of the application can be adapted to the notation of the courses of different universities without changing the source code. This is because a configurable printing algorithm was designed that allows changing the presentation of the formulas. In this work, we explain how CalcLogic was adapted to a course at FIU and the result of the educational experiments (at this university) are shown. Additionally, new algorithms are described: 1) a versatile printing algorithm that allows the administrator to configure the symbols of a theory, by assigning the desired presentation with LaTeX; 2) an algorithm, based on Broda and Damas combinators, for generate monotonic or anti monotonic inferences in the Dijkstra-Scholten logic; and 3) an algorithm to generate the proofs of dual theorems, in Boolean Algebra theory.

2 Contribution

A proof theory was developed for the Dijkstra-Scholten system that explains, based on algorithms, three proof methods (direct method, starting from one side and weakening-strengthening), three metatheorems (metatheorem of true, parity metatheorem and duality metatheorem) and the process of printing a proof. Each user inference and metatheorem corresponds to an algorithm that generates a sequence of elementary inferences that are correctly combined to generate a more complex inference. Designing and proving that these algorithms always generate correct inferences is the goal of the Algorithmic Proof Theory field.

The development of these algorithms is a contribution to the area of Algorithmic Proof Theory for the Dijkstra-Scholten system. In particular, the classic Parity Metatheorem of [1] is presented as an algorithm that makes use of untyped Broda-Damas Combinators [5]. This is an example of how the untyped theories can be more useful than expected. All the theories developed in this work were done without a type system. The application could load untyped equational theories and propositional calculational logic. The untyped equational theories designed for students were Boolean algebras, Field theory (only terms equalities without notion of sets, ideals, quotients, homomorphism and isomorphism), Trigonometric, Finite difference and Summation theory. This variety of theories show an idea of how powerful untyped theories are for college courses.

On the other hand a configurable algorithm that prints formulas in LaTeX format has been implemented. Said algorithm prints the symbols depending on the configuration of the administrator. This paper explains the configuration made on this algorithm so that the formulas were printed in notation of the Discrete Mathematics course using [6] as the textbook. This is a verification that the application is versatile thanks to the design of this formula printing algorithm.

This paper presents CalcLogic functionalities that did not exist when the first work on this tool was written [3]. These are: 1) Direct Proof Method for Propositional Logic, 2) weakening-strengthening proof method, 3) window for show the instantiation introduced by the user, 4) automatic substitution for inferences, 5) auto-generation of proofs of the dual theorems for Boolean Algebra theory, 6) symbol configuration and 7) automatic monotonic or anti monotonic inferences.

CalcLogic's interface was designed to correct the student, so that when his one-step inference is not correct, the assistant notifies it, but without explaining the error. This work also studies to what extent this education system is effective and if it can be used as a form of assessment. In this way, to collect educational data, CalcLogic was used to teach Propositional Logic and Boolean Algebras to students at FIU.

The experiment that has been done in FIU was oriented to specifically answer the following **educational research questions**: 1) To what extent does CalcLogic's user interface allow the student to improve their competencies to proof Propositional Logic and Boolean Algebras theorems?. Since the prover doesn't construct the proofs, only said if there is an inference error, it is reasonable to ask, 2) to what extent the stored student proofs can be used for student assessment?.

To answer these questions, students were divided into groups depending on the performance obtained in CalcLogic. Then, written exams were made on the same content exercised with CalcLogic. Then, the performance with the application was compared by groups with respect to the written exams. The results show that those who perform well using the application also perform well in classroom assessments. If this experiment is repeated, and this correlation is always good, it could be argued that you can assess a large percentage using CalcLogic, because the expected result could be very similar in a written exam.

3 Related Jobs

Calculational Logic was first described in the book [1] and later in [2] which popularized it. The book [2] is used in undergraduate computer logic courses at many universities. However, from a rigorous point of view, the previous references left open theoretical aspects in the foundation of the system. Later work was done on the foundations, but they differ from each other, to the point that there have been misunderstandings that were cleared up in [7].

In this way there is not standard on which lies the rules of inference of the system. For example, [8] proposes a system with four rules: Transitivity, Substitution, Leibniz and Equanimity. In [9, 10, 11, 12] the system has several types of Leibniz rules. In [13] there is a discussion on the possible variants of Leibniz's rule. Leibniz's rule using in [14] is slightly different from the other works. Then in the attempt to add the everywhere operator (see [15, 16]) to the system, [17, 18] conclude that the Substitution rule should be removed. The inference rules that were chosen in CalcLogic's design are the ones from [18], because these rules allow the everywhere operator to be added in the future and are compatible with the untyped fragment of the logic used.

On the other hand, in the field of Algorithmic Proof Theory there are several algorithms for the Gödel T inference system, the Hilbert system and the Gentzel system [19, 20, 21]. No results has been found in the bibliography on this area for the Dijkstra-Scholten system, which justifies the development of the algorithms in this work. This work is a contribution to the field of Algorithm Proof Theory for the Dijkstra-Scholten system.

Algorithm Proof Theory is the base to implement proof assistants. However, the study of proof assistants is classified within another area called Automated Theorem Proving, which in turn is part of an area called Automated Reasoning. The Automated Reasoning area consists of the use of a computer to carry out reasoning according to inference rules and syntax of a Formal Logic inference system.

According to Krysia Broda's Automated Reasoning course at Imperial College London [22], the Automated Reasoning area is divided into the categories shown in Fig 1. Although Calculational Logic is not exactly Equational Logic (see section 4), CalcLogic fits best into the Equality Reasoning category.

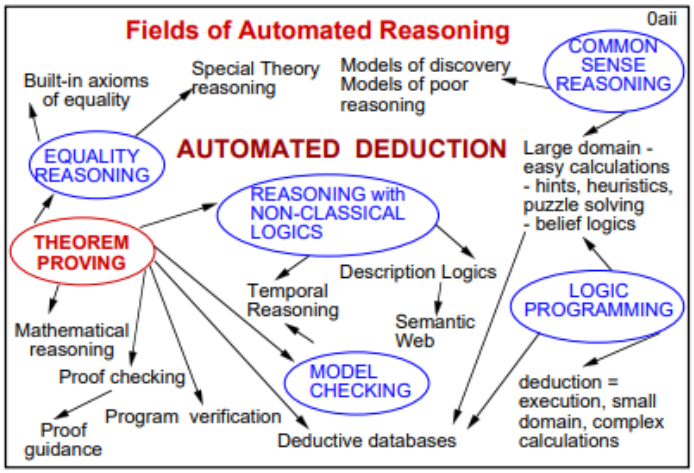


Figure 1: Sub areas of Automated Reasoning [22]

The application presented in this work is an interactive proof assistant. There are many, but the most famous interactive proof assistant are *Isabelle/HOL* [23] with its interactive window *PG Tips* [24] and *Coq* [25]. None of these assistants are based on Calculational Logic.

There are two proof assistants for Calculational Logic; CalcCheck [26], and CalcCheck web [27]. The last two applications are basically the same, the difference is that the first one requires installation and the second one runs with a browser and internet connection. There is an important difference between these applications and the one presented in this work. The previous ones require that the proof should be written in LaTeX in order to be processed and verified, while a CalcLogic user does not need to know any language in particular, since that everything is done by means of the mouse.

Calculational Logic additionally has ideal features to verify a proof automatically. For example, in [28], an application design is proposed, with the ability to verify handwritten calculation proofs, from its photograph. The authors explain how the theoretical characteristics of Calculational Logic are adequate to be able to carry out this verification process.

The first time CalcLogic was used in an educational setting was during the time of the COVID19 pan-

demic. In this period, it was decided to use CalcLogic as a mechanism that allowed Discrete Mathematics assessments, reducing cheating among students or making it evident. The results of this experiment are published in [3]. As a complement to [3], this paper studies how effective learning is with CalcLogic. It seeks to justify that the assistant can be used in the assessments, because it is safe (as indicated by [3]) and representative of the student's knowledge.

For avoiding cheating between students during remote assessments, at [29] a review was written about the secure assessment practices taken by various universities. One practice in [29] is about a software and services for detecting copies by scanning the answers delivered by students. Instead of images, the answers of the students delivered by CalcLogic were encoded in the tables of a database as a string. With this, it is easier to check for copies, with an automatic verbatim comparison of the answers stored in the database.

Other attempts to use proof assistants in education can be reviewed in [30], where the Coq assistant was used to help the student with the proofs. In that work, the student was exposed to learn a formal language to work with Coq, and then it was sought to transform the proofs into the language of mathematics books. Instead, with the CalcLogic interface, students can verify proofs directly in the same language that math books write their formulas.

In [31] a web interactive theorem prover is shown that assists natural deduction derivations. The users can proof theorems by means of the mouse like CalcLogic does, but the formulas are not displayed in the fancy style that is achieved with the CalcLogic interface. For example, in [31] both the predicates and the functionals are only printed with letters from A to z, whereas in CalcLogic any symbol can be printed with LaTeX style.

On the other hand combinatory logic (defined by Schönfinkel [32] and popularized by [33]) is a formal system of inference that uses an applicative language without variables, but with functions called combinators. Broda and Damas combinators [5] was use to implement the parity metatheorem of weakening-strengthening proof method using Disjktra-Scholten proof style. The Broda and Damas abstraction algorithm is a version of the lambda lifting technique [34] but in combinatory logic.

Finally this paper has the following structure: Section 4 shows the notation used in Calculational Logic proofs, the list of its inference rules, and the proof methods supported by CalcLogic. Section 5 explains the functionalities of CalcLogic and shows images of how its user interface works. The functionalities explained are: List Theorems, show saved proofs, symbol configuration, starting for one side method, direct method, weakening-strengthening method, auto-generating proof of dual statements, one step inference, statement instantiation, manual entry of formulas. Section 6 explains the printing algorithm for display formulas, and the algorithms used for the parity metatheorem and dual metatheorem. Section 7 the data resulting from the educational experiment is shown and a discussion of these is made. Finally, Section 8 contains the conclusions of the work.

4 Notations and Theoretical Aspects

This section presents the definitions and notations that have been used in selected works on Calculational Logic. The following rules and notations are valid for equality or for equivalence. Therefore, to generalize, we will write eq indicating that equality $=$ or equivalence $\equiv$ can be used.

A derivation in Calculational Logic is an inference tree whose root is the conclusion of the inference. The notation $\Gamma \vdash P$ means that there is a derivation assuming the list of premises Γ and concluding P. To build the derivation trees the following rules of inference are used:

Axiom: If P occurs in list Γ

$$\frac{}{\Gamma \vdash P}\ \text{Ax id}$$

Leibniz rule

$$\frac{\Gamma \vdash P\ eq\ Q}{\Gamma \vdash E_P^z\ eq\ E_Q^z}\ \text{L}$$

Symmetry

$$\frac{\Gamma \vdash P\ eq\ Q}{\Gamma \vdash Q\ eq\ P}\ \text{S}$$

Transitivity

$$\frac{\Gamma \vdash P \ eq \ Q \qquad \Gamma \vdash Q \ eq \ R}{\Gamma \vdash P \ eq \ R} \ \mathrm{T}$$

Equanimity

$$\frac{\Gamma \vdash P \equiv Q \qquad \Gamma \vdash P}{\Gamma \vdash Q} \ \mathrm{E}$$

The notation E^z denotes a formula scheme in which some of its subformulas or subterms may have been denoted by placeholder z [13]. The notation E^z_P denotes the result of replacing in E^z all occurrences of its placeholder z by the formula or term P [13]. Leibniz's rule, when eq is $=$, is also called "Replacement of equals by equals".

The list of premises Γ is usually infinite and is described by schemes, which are instantiated to pick a particular premise. The action of instantiating a scheme occurs at a meta level, and is not considered a rule of inference. The rule for instantiating is as follows:

If R is a scheme then

$R[\overline{x} := \overline{Exp}]$ is a premise in Γ

The notation $R[\overline{x} := \overline{Exp}]$ denotes the result of substituting in parallel each occurrence in R, of the variables in the list $\overline{x}$, by the respective expression in the list of expressions $\overline{Exp}$.

The Equanimity rule is generally used in a method of proof that in [2] was called the "Direct Method". The Equanimity rule is what differentiates this formal system from the Equational Logic described in books of Term Rewriting as [35] (Equational Logic is defined as Calculational Logic, where eq is just $=$, with the reflexivity rule and without Equanimity).

Neither [1] nor [2] write proofs in derivation tree notation, but instead use a vertical writing notation that abbreviates derivation trees. This notation is explained below.

If $P \ eq \ Q$ is a premise in Γ and it is labeled with the identifier id, then when we write:

A_1
eq ⟨st id⟩
A_2

It is referring to the following derivation:

$$\frac{}{\Gamma \vdash P \ eq \ Q} \ \text{Ax id}$$

when A_1 is P and A_2 is Q. On the other hand, if A_1 is Q and A_2 is P, then the notation is referring the derivation:

$$\frac{\dfrac{}{\Gamma \vdash P \ eq \ Q} \ \text{Ax id}}{\Gamma \vdash Q \ eq \ P} \ \mathrm{S}$$

The letters "st" are an abbreviation for the word "statement". If the statement annotated with the identifier id is $P \ eq \ Q$ and E is a scheme with placeholder z, then the notation

A_1
eq ⟨st id with $\overline{x} := \overline{Exp}$ and E⟩
A_2

is referring the derivation:

$$\frac{\dfrac{}{\Gamma \vdash P[\overline{x} := \overline{Exp}] \ eq \ Q[\overline{x} := \overline{Exp}]} \ \text{Ax id}}{\Gamma \vdash E^z_{P[\overline{x}:=\overline{Exp}]} \ eq \ E^z_{Q[\overline{x}:=\overline{Exp}]}} \ \mathrm{L}$$

when A_1 is the formula resulting from calculating $E^z_{P[\overline{x}:=\overline{Exp}]}$ and A_2 is the formula resulting from calculating $E^z_{Q[\overline{x}:=\overline{Exp}]}$. If it is not and $A_1 \ eq \ A_2$ is $E^z_{Q[\overline{x}:=\overline{Exp}]} \ eq \ E^z_{P[\overline{x}:=\overline{Exp}]}$, then a symmetry inference is added at the end, i.e. the notation:

A_1
eq ⟨st id with $\overline{x} := \overline{Exp}$ and E⟩
A_2

would mean:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P[\overline{x} := \overline{Exp}] \ eq \ Q[\overline{x} := \overline{Exp}]}\ \text{Ax id}}{\Gamma \vdash E^z_{P[\overline{x}:=\overline{Exp}]} \ eq \ E^z_{Q[\overline{x}:=\overline{Exp}]}}\ \text{L}}{\Gamma \vdash E^z_{Q[\overline{x}:=\overline{Exp}]} \ eq \ E^z_{P[\overline{x}:=\overline{Exp}]}}\ \text{S}$$

In the same way:

A_1
$eq \quad \langle$st id and $E\rangle$
A_2

is a compact notation of the derivation

$$\dfrac{\dfrac{}{\Gamma \vdash P \ eq \ Q}\ \text{Ax id}}{\Gamma \vdash E^z_P \ eq \ E^z_Q}\ \text{L}$$

when A_1 is the formula resulting from calculating E^z_P and A_2 is the formula resulting from calculating E^z_Q. On the other hand, if $A_1 \ eq \ A_2$ is the resulting formula of $E^z_Q \ eq \ E^z_P$, then the notation is understood as the derivation tree described above, to which a symmetry inference (S) is added at the end.

Analogously, it follows that

A_1
$eq \quad \langle$st id with $\overline{x} := \overline{Exp}\rangle$
A_2

is a compact notation of the derivation:

$$\dfrac{}{\Gamma \vdash P[\overline{x} := \overline{Exp}] \ eq \ Q[\overline{x} := \overline{Exp}]}\ \text{Ax id}$$

when A_1 is the formula resulting from calculating $P[\overline{x} := \overline{Exp}]$ and A_2 is the formula resulting from calculating $Q[\overline{x} := \overline{Exp}]$. If it is not and $A_1 \ eq \ A_2$ is $Q[\overline{x} := \overline{Exp}] \ eq \ P[\overline{x} := \overline{Exp}]$, then a symmetry inference is added at the end, i.e. the notation:

A_1
$eq \quad \langle$st id with $\overline{x} := \overline{Exp}\rangle$
A_2

would mean:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P[\overline{x} := \overline{Exp}] \ eq \ Q[\overline{x} := \overline{Exp}]}\ \text{Ax id}}{\Gamma \vdash Q[\overline{x} := \overline{Exp}] \ eq \ P[\overline{x} := \overline{Exp}]}\ \text{S}}$$

We have seen that

A_1
$eq \quad \langle \text{hint}_1 \rangle$
A_2

by itself, denotes a derivation $\Gamma \vdash A_1 \ eq \ A_2$ composed of several inferences. This is the kind of derivation that Dijkstra and Scholten considered to be atomic inference, although by definition it is not. Similarly, in CalcLogic, for the user point of view, this is the type of inferences that are considered one-step.

The symbol "$eq \quad \langle$hint$\rangle$" is metaconjunctive in the sense that the following vertical notation:

A_1
$eq \quad \langle \text{hint}_1 \rangle$
A_2
$eq \quad \langle \text{hint}_2 \rangle$
A_3
$eq \quad \langle \text{hint}_3 \rangle$
A_4

denotes the conjunction of the derivations "$\Gamma \vdash A_1 \ eq \ A_2$ and $\Gamma \vdash A_2 \ eq \ A_3$ and $\Gamma \vdash A_3 \ eq \ A_4$" where "and" is the conjunction of the metalanguage. The intention of this vertical multi-step "$eq \quad \langle$hint$\rangle$" notation is to write derivations of theorems more compactly than the tree.

A proof with the vertical notation of more than one step is not sufficient by itself to reconstruct a formal derivation in the form of a tree. The vertical notation only provides knowledge of a list of derivations (in the case of the example the list $\Gamma \vdash A_1 \ eq \ A_2$, $\Gamma \vdash A_2 \ eq \ A_3$, $\Gamma \vdash A_3 \ eq \ A_4$ of derivations). It is necessary the information of the proof method used to build the derivation tree, based on this list of derivations.

The proof method "Starting from one Side of Equation" will be the aim of the next section.

4.1 Starting from One Side of the Equation

This proof method is used only when the theorem to be proved is of the form $P\ eq\ Q$. It consists of using the vertical proof notation starting from P, to arrive at Q with several steps.

For example, a vertical proof declared with the method of starting from one side, has the following form:

$$\begin{array}{ll} & P \\ eq & \langle \text{hint}_1 \rangle \\ & A_1 \\ eq & \langle \text{hint}_2 \rangle \\ & A_2 \\ eq & \langle \text{hint}_3 \rangle \\ & A_3 \\ \vdots & \\ & A_{n-1} \\ eq & \langle \text{hint}_n \rangle \\ & Q \end{array}$$

and it is interpreted as the following derivation tree

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash P\ eq\ A_1 \qquad \Gamma \vdash A_1\ eq\ A_2}{\Gamma \vdash P\ eq\ A_2}\,\text{T} \qquad \Gamma \vdash A_2\ eq\ A_3}{\Gamma \vdash P\ eq\ A_3}\,\text{T}}{\begin{array}{c}\vdots \\ \Gamma \vdash P\ eq\ A_{n-1}\end{array} \qquad \Gamma \vdash A_{n-1}\ eq\ Q}\,\text{T}}{\Gamma \vdash P\ eq\ Q}\,\text{T}$$

Where each derivation $\Gamma \vdash A_{i-1}\ eq\ A_i$ is the derivation corresponding to

$$\begin{array}{ll} & A_{i-1} \\ eq & \langle \text{hint}_i \rangle \\ & A_i \end{array}$$

according to the notation described above.

4.2 Direct Method

This proof method consists of using the vertical notation starting from the theorem to prove and arriving, by means of equivalences, at some already proven theorem. It is also valid to start from an already proven theorem and arrive at the theorem to prove, by means of equivalences.

For example, if P is the theorem to be proved and Q is a proven theorem, then the following vertical proof using the direct method is

$$\begin{array}{ll} & P \\ \equiv & \langle \text{hint}_1 \rangle \\ & A_1 \\ \equiv & \langle \text{hint}_2 \rangle \\ & A_2 \\ \vdots & \\ \equiv & \langle \text{hint}_n \rangle \\ & Q \end{array}$$

and it is interpreted as the following derivation tree of P

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash P \equiv A_1 \qquad \Gamma \vdash A_1 \equiv A_2}{\Gamma \vdash P \equiv A_2}\,\text{T} \qquad \vdots}{\begin{array}{c}\vdots \\ \Gamma \vdash P \equiv A_{n-1}\end{array} \qquad \Gamma \vdash A_{n-1} \equiv Q}\,\text{T}}{\dfrac{\Gamma \vdash P \equiv Q}{\Gamma \vdash Q \equiv P}\,\text{S}}}{\Gamma \vdash P}\,\text{E}$$

Where each derivation $\Gamma \vdash A_{i-1} \equiv A_i$ is the derivation corresponding to

$$\begin{array}{ll} & A_{i-1} \\ \equiv & \langle \text{hint}_i \rangle \\ & A_i \end{array}$$

according to the notation described above.

Also in this method P can be a theorem already proven and Q the theorem to be proved. But in this case the derivation tree that is abbreviated with the vertical notation, does not have the symmetry rule before the equanimity.

4.3 True Metatheorem

In the area of Calculational Logic the following metatheorem is known (Metatheorem 2.2 in [36] or weakened form of Metatheorem 3.7 in [2]): If P is a theorem, then $true \equiv P$ is a theorem. The proof of the metatheorem is given by the following derivation tree scheme, where ∇ is the derivation tree of the theorem P.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash (true \equiv p)[p := P] \equiv p[p := P]}\,\mathrm{A}}{\Gamma \vdash P \equiv (true \equiv P)}\,\mathrm{S} \qquad \dfrac{\nabla}{\Gamma \vdash P}}{\Gamma \vdash true \equiv P}$$

For a fixed ∇ with root P, the previous tree will be denoted $MT(\nabla)$.

The previous scheme can also be understood as an algorithm that receives a concrete theorem T and its derivation ∇', to then replaces ∇ in the previous scheme by ∇' and each occurrence of P by T. This algorithm generates a concrete derivation tree with root $\Gamma \vdash true \equiv T$. From this point of view, this metatheorem is an algorithm that always generate correct derivation trees, which is convenient for implementing the proof assistant. As a design decision, in CalcLogic every metatheorem is an algorithm.

4.4 Modus Pones

In calculational logic the Modus Pones inference rule

$$\dfrac{\dfrac{\nabla_1}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\nabla_2}{\Gamma \vdash A}}{\Gamma \vdash B}$$

is a derivable rule from the other rules, since B can be proof by direct method as follows (when 3.39 and 3.66 are the identifiers of $true \wedge p \equiv p$ and $p \wedge (p \Rightarrow q) \equiv p \wedge q$, respectively, i.e.,

$$\begin{array}{ll} & B \\ \equiv & \langle \text{st 3.39 with } p := B \rangle \\ & true \wedge B \\ \equiv & \langle \text{true metatheorem and } E : z \wedge B \rangle \\ & A \wedge B \\ \equiv & \langle \text{st 3.66 with } p, q := A, B \rangle \\ & A \wedge (A \Rightarrow B) \\ \equiv & \langle \text{true metatheorem and } E : z \wedge (A \Rightarrow B) \rangle \\ & true \wedge (A \Rightarrow B) \\ \equiv & \langle \text{st 3.39 with } p := A \Rightarrow B \rangle \\ & A \Rightarrow B \\ \equiv & \langle \text{true metatheorem} \rangle \\ & true \end{array}$$

It is important to have the modus ponens rule in the inference system, since it is the fundamental piece of the proof method that is explained in next subsection.

4.5 Weakening-Strengthening Method

The Weakening-Strengthening method emphasizes the use of the transitivity of the $\Rightarrow$ and $\Leftarrow$ connectors. Therefore this method is applicable only to proving statements of the form $p\ op\ q$, where op is either $\Rightarrow$ or $\Leftarrow$.

In a proof of $p\ op\ q$ statement with the Weakening-Strengthening method, is allowed to place the operator op instead of $\equiv$, as the line separator in the vertical proof notation. For example, a vertical display consisting only of the following three lines

$$\begin{array}{ll} & A \\ op & \langle \text{st id} \rangle \\ & B \end{array}$$

it means the following derivation ∇

$$\dfrac{}{\vdash p\ op\ q}\,\text{Ax id}$$

where id is the identifier of the theorem $p\ op\ q$. In the same way the following derivation tree

$$\dfrac{}{\vdash p[\overline{x} := \overline{Exp}]\ op\ q[\overline{x} := \overline{Exp}]}\,\text{Ax id}$$

is printed as follows

$$\begin{array}{ll} & A \\ op & \langle \text{st id with } \overline{x} := \overline{Exp} \rangle \\ & B \end{array}$$

where A and B are the formulas resulting from calculating $p[\overline{x} := \overline{Exp}]$ and $q[\overline{x} := \overline{Exp}]$ respectively.

The Leibniz rule only can be applied to an equality or equivalence, however this method allows inferences in which some implication is used in a subformula. In those cases the same notation E as Leibniz is used for the hint. For example if formula $p \Rightarrow p \vee q$ has identifier 2 then we can write

$$\begin{array}{ll} & p \wedge q \Rightarrow p \\ \Rightarrow & \langle \text{st 2 and } E : p \wedge q \Rightarrow z \rangle \\ & p \wedge q \Rightarrow p \vee q \end{array}$$

However in [1] it is explained that depending on the place of a subformula, denoted by z in E, the implication can change the direction. For example if z is located in the antecedent of an implication the inference looks like this:

$$\begin{array}{ll} & p \wedge q \Rightarrow p \\ \Leftarrow & \langle \text{st 2 and } E : z \wedge q \Rightarrow p \rangle \\ & (p \vee q) \wedge q \Rightarrow p \end{array}$$

According to the terminology used in [1], formula scheme $E^z : p \wedge q \Rightarrow z$ defines a monotonic predicate transformer, while formula scheme $E^z : z \wedge q \Rightarrow p$ defines an anti monotonic one.

The last two vertical notation represent a derivation tree ∇ with root $(p \wedge q \Rightarrow p) \Rightarrow (p \wedge q \Rightarrow p \vee q)$ or $(p \wedge q \Rightarrow p) \Leftarrow ((p \vee q) \wedge q \Rightarrow p)$ respectively. The construction of ∇ is by means of an algorithm that is described in section 6. The rule that specifies the direction of the main logic connector ($\Rightarrow$ or $\Leftarrow$) depending of the position of z is a statement called metatheorem of Parity [1]. Our algorithm is another way to find the right logic connector ($\Rightarrow$ or $\Leftarrow$), therefore the rule described in the metatheorem of parity according [1] can be ignored in this work.

In this section the symbol ∇ denote a derivations with the form

$$\begin{array}{ll} & A \\ op & \langle \text{hint} \rangle \\ & B \end{array}$$

regardless of which hint is used. The one-step inference in this proof method is either ∇ or an user point of view's one-step inference (according section 4). If the proof had several one-step inferences the construction of the proof can be described recursively. The base case for a several steps derivation with Weakening-Strengthening method are of two kind: A derivation $MT(\nabla)$ or a derivation tree like this:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash (e \equiv f \Rightarrow E_f^z \Rightarrow (true \equiv E_e^z))[e, f, E := A_1, A_m, z \ op \ A_{m+1}]}\,\text{Ax} \quad \dfrac{\nabla'}{\Gamma \vdash A_1 \equiv A_m}}{\Gamma \vdash A_m \ op \ A_{m+1} \Rightarrow (true \equiv A_1 \ op \ A_{m+1})} \quad \dfrac{\nabla}{\Gamma \vdash A_m \ op \ A_{m+1}}}{\Gamma \vdash true \equiv A_1 \ op \ A_{m+1}}$$

where ∇' is a derivation of $A_1 \equiv A_2$ with the Starting From One Side method. The previous tree is printed to the user like this:

$$\begin{array}{ll} & A_1 \\ \equiv & \langle hint_1 \rangle \\ & \vdots \\ \equiv & \langle hint_m \rangle \\ & A_m \\ op & \langle hint_{m+1} \rangle \\ & A_{m+1} \end{array} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \nabla' \\ \left.\begin{array}{l} \\ \end{array}\right\} \nabla \end{array}$$

Now if ∇'' is a Weakening-Strengthening derivation tree, we can construct recursively more complex derivation trees like this:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash ((true \equiv p \ op \ q) \Rightarrow q \ op \ r \Rightarrow (true \equiv p \ op \ r))[p, q, r := A_1, A_k, A_{k+1}]}\,\text{Ax} \quad \dfrac{\nabla''}{\Gamma \vdash true \equiv A_1 \ op \ A_k}}{\Gamma \vdash A_k \ op \ A_{k+1} \Rightarrow (true \equiv A_1 \ op \ A_{k+1})} \quad \dfrac{\nabla}{\Gamma \vdash A_k \ op \ A_{k+1}}}{\Gamma \vdash true \equiv A_1 \ op \ A_{k+1}}$$

or like this:

$$\dfrac{\dfrac{\nabla''}{\Gamma \vdash true \equiv A_1 \ op \ A_k} \quad \dfrac{\nabla'''}{\Gamma \vdash A_1 \ op \ A_k \equiv A_1 \ op \ A_{k+1}}}{\Gamma \vdash true \equiv A_1 \ op \ A_{k+1}}$$

where ∇''' is an user point of view's one-step inference (according to section 4).

The first tree is printed to the user in this way:

$$\left.\begin{array}{l}\vdots\\ \end{array}\right\}\nabla'' \textit{ in vertical notation}$$
$$\left.\begin{array}{ll} & A_k\\ op & \langle hint\rangle\\ & A_{k+1}\end{array}\right\}\nabla$$

and the second one is printed in this way:

$$\left.\begin{array}{l}\vdots\\ \end{array}\right\}\nabla'' \textit{ in vertical notation}$$
$$\left.\begin{array}{ll} & A_k\\ \equiv & \langle hint\rangle\\ & A_{k+1}\end{array}\right\}\nabla'''$$

If the statement that has to be proved is $A_1\ op\ A_n$ and the root of the current derivation tree ∇_c is $true \equiv A_1\ op\ A_n$, then the proof ends adding an equanimity inference in this way:

$$\dfrac{\dfrac{\nabla_c}{\Gamma \vdash true \equiv A_1\ op\ A_n} \qquad \dfrac{}{\Gamma \vdash true}\,\mathrm{A}}{\Gamma \vdash A_1\ op\ A_n}$$

A proof is a derivation tree in the CalcLogic Backend and it is stored in the database as string. Derivation trees are not visible to the user, for greater usability, only the previous vertical notation is visible. To print a proof in the frontend, the application performs an algorithm that constructs a string with the vertical notation while traveling on the tree.

In the next section there is a description of CalcLogic's functionality and how the interface looks like for each proof method in vertical notation.

5 CalcLogic Description

The application manages a session for each student and teacher, therefore it is necessary to create an account in it. The main textbook for the Discrete Mathematics course at FIU is [6]. Although the Propositional Logic chapter of this book uses a different formal system than Calculational Logic, the "Direct" and "Starting from One Side" proof methods are valid in both systems. In this way CalcLogic was configured so that the math language displayed and manipulated by the user was exactly the same as the math language of [6].

To display the language as [6] a configuration module was implemented with an algorithm that allows presentation changes of the symbols, using LaTeX codification. For example the symbol used in [6] for implication is $\rightarrow$ instead of $\Rightarrow$. With our algorithm, the application can display $\rightarrow$ by setting in the configuration module a LaTeX code `\rightarrow` for the implication symbol.

5.1 Symbols Settings

Application administrators can create the symbols and edit their attributes. The attributes to be edited are: the number of arguments, the precedence, LaTeX notation and the location of the arguments. This allows the application to be versatile and extended to different mathematical theories with the diversity of symbols that LaTeX allows to write. Figure 2 shows the table of symbols that administrator can edit.

All the symbols of the application are listed in the table, and each one is associated with a unique integer identifier (*id*). Formulas are printed to the browser in LaTeX and then rendered in the client using MathJax [37], a translator from LaTeX to HTML written in Javascript.

The display depends on the fields that are configured by the administrators for each symbol. This is why it is important that the administrator understand the printing algorithm explained in section 6.

5.2 My Theorems

The "My Theorems" option on the navigation bar displays a view with the complete list of theorems that the user needs to prove. This view allows the user to inspect his progress. In the list are all the statements: the axioms, the proven theorems, and the unproved theorems. The theorems that have already been proved and the axioms are labeled by an open padlock symbol, indicating that the statement can be used to make another proof. Theorems that have not yet been proved are labeled by a closed padlock, indicating that the statement cannot be used in another proof yet. The padlock of a theorem is opened when its proof is done.

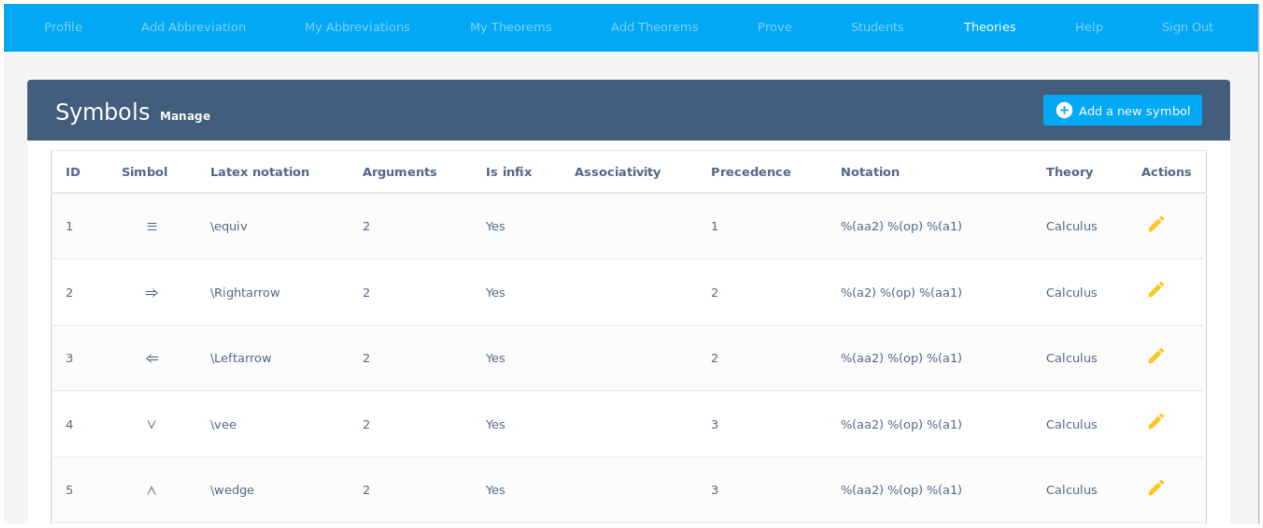


Figure 2: Table with the editable attributes of each symbol of the application

The student's goal is to open as many locks as possible (prove as many theorems as possible), adding a ludic component to the learning process.

The statements are classified by categories. For example, in the CalcLogic instance with Propositional Logic content, the categories are: "primary identities" with 19 identities available on equivalences, "implication identities" with 5 available and 4 pending to prove, "if and only if identities" with 4 available and 2 pending to prove and "Exercises" with a list of theorems to prove. Analogously, the CalcLogic instance with Boolean Algebra content contains the categories: "Axioms and Commutative Identities" with 10 axioms and 6 commuted versions of the axioms pending to prove, "Primary Identities" with 4 pending identities to prove and "Exercises" with a list of theorems to prove.

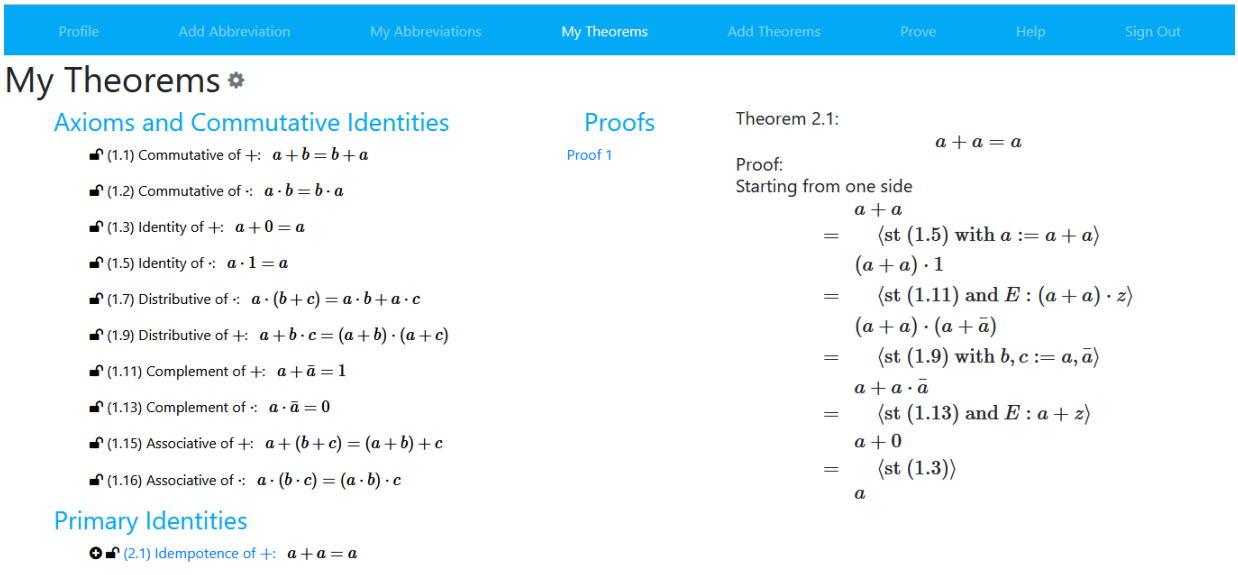


Figure 3: Inspection the Theorem 2.1 proof made by the user

The student can inspect the proofs already done by selecting an open-padlock theorem. Likewise, the teacher can see the proofs made of all the students. For example, Figure 3 shows the proof of Theorem 2.1 that the user had done. For the CalcLogic instance with the Boolean Algebras content, each proven theorem has a $\oplus$ symbol to the left of it. Clicking on it displays the option to auto-generate the proof of the dual theorem. For example, Figure 4 shows the auto-generated proof of the dual theorem based on the proof of Figure 3.

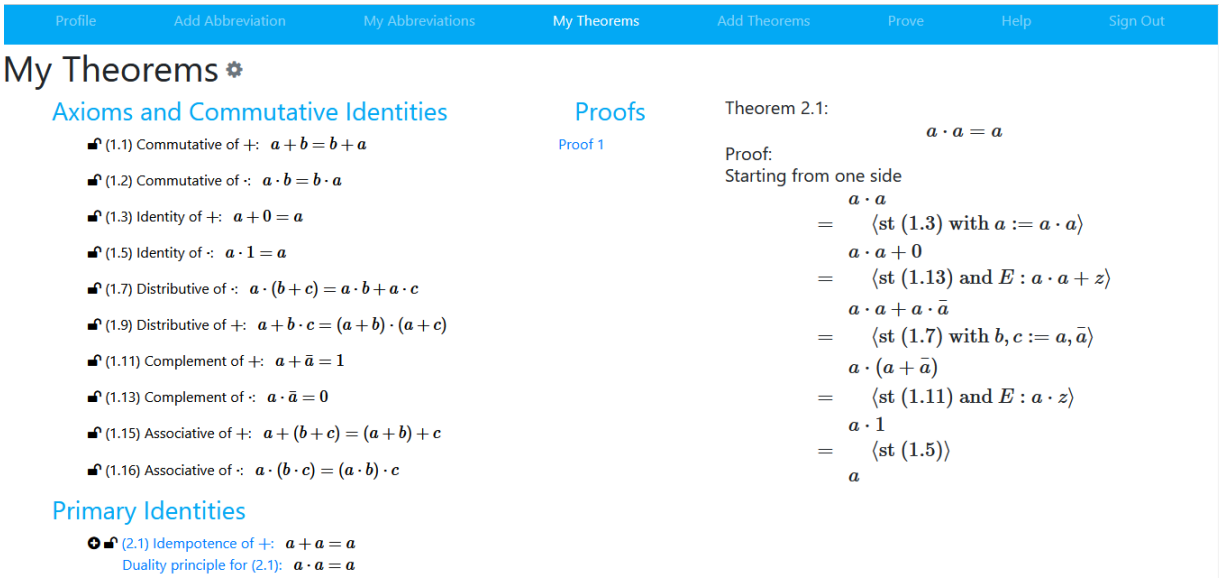


Figure 4: Auto generated proof of Theorem 2.1 dual

5.3 Prove

The "Prove" option on the navigation bar displays a view to start a proof. In this view there are the categories of theorems in blue that display the selectable statements. Selecting one of them, its proof starts.

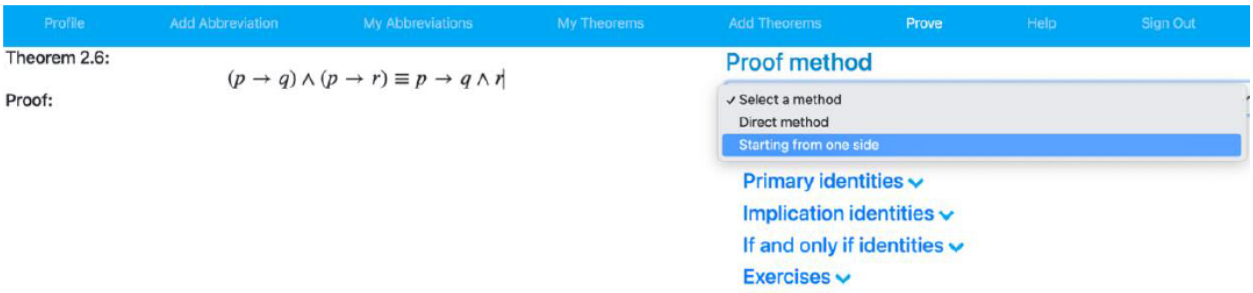


Figure 5: Start of Theorem 2.6 proof with method "starting from one side"

Immediately after starting the proof of a theorem, the proof method to use must be selected. For example, in Figure 5 the proof of Theorem 2.6 is being started by choosing the "Starting from one side" method.

5.3.1 Performing One-Step Inference

Once a proof is started, a one-step inference can be performed. For this, one of the available statements must be selected, which are grouped by categories on the right side. When one is selected, the fields of substitution (A in Figure 6) and Leibniz rule of inference (B in Figure 6) appear. Once the details of the substitution and Leibniz rule to be performed have been entered correctly the inference is made by pressing the Infer button.

The forms for indicating the substitution and Leibniz rule can be filled by placing the cursor on them and using the buttons, to build formulas, (C in Figure 6). However, there are automated mechanisms to perform this task.

The expression E for the Leibniz rule of inference is automatically calculated by the application if the subformula to be replaced is underlined on the last line of the proof. For example in Figure 6 the user performed the following steps to make a one-step inference: 1) selected the identity 2.1. 2) Underlined the subformula $p \to q$ to have CalcLogic automatically filled in the Leibniz field labeled with B. 3) Pressed the "infer" button. The result of this inference is found in Figure 7. As it was described above, CalcLogic can be operated only with the mouse.

If the inference made by the user is not correct, CalcLogic does not allow you to proceed. For example, Figure 8 shows the error message "No valid inference" after incorrectly attempting to use De Morgan's Law

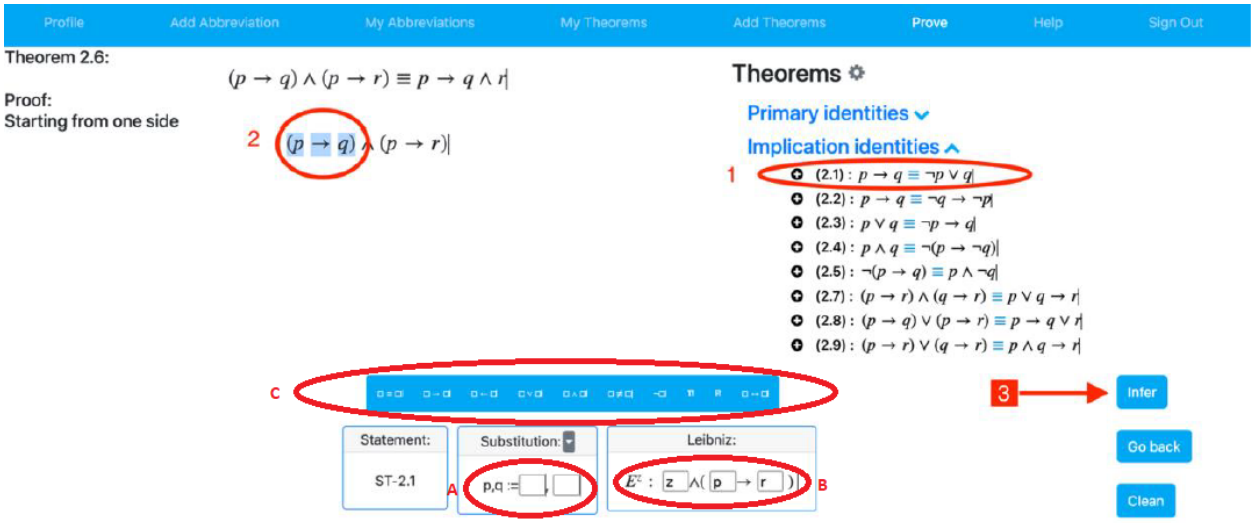


Figure 6: Process and forms for performing one-step inference

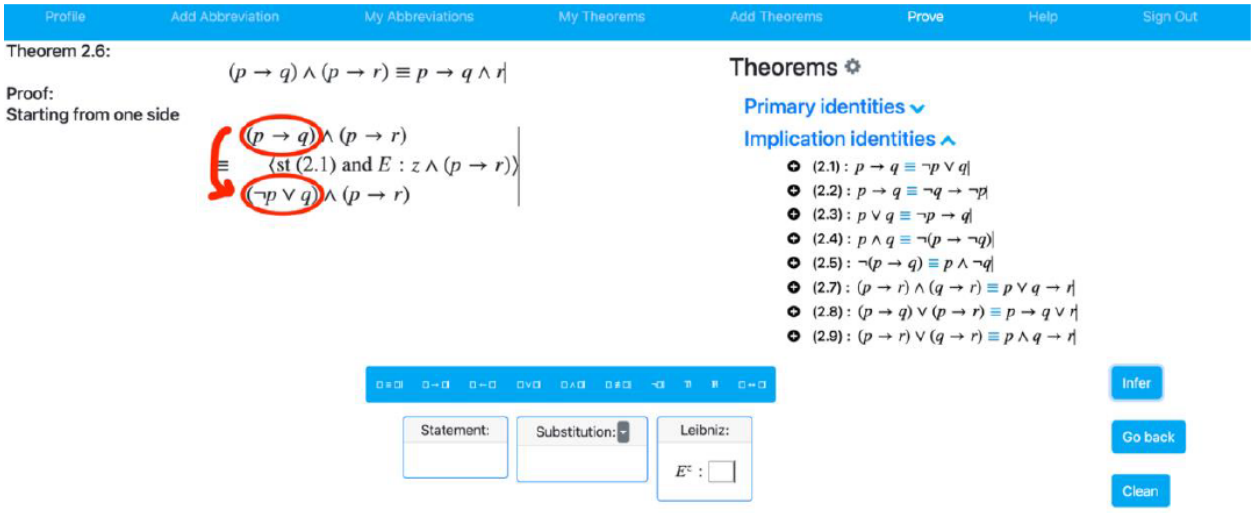


Figure 7: Result of the Figure 6 inference

1.13 $(\neg(p \wedge q) \equiv \neg p \vee \neg q)$ in the subformula $\neg p \vee q$.

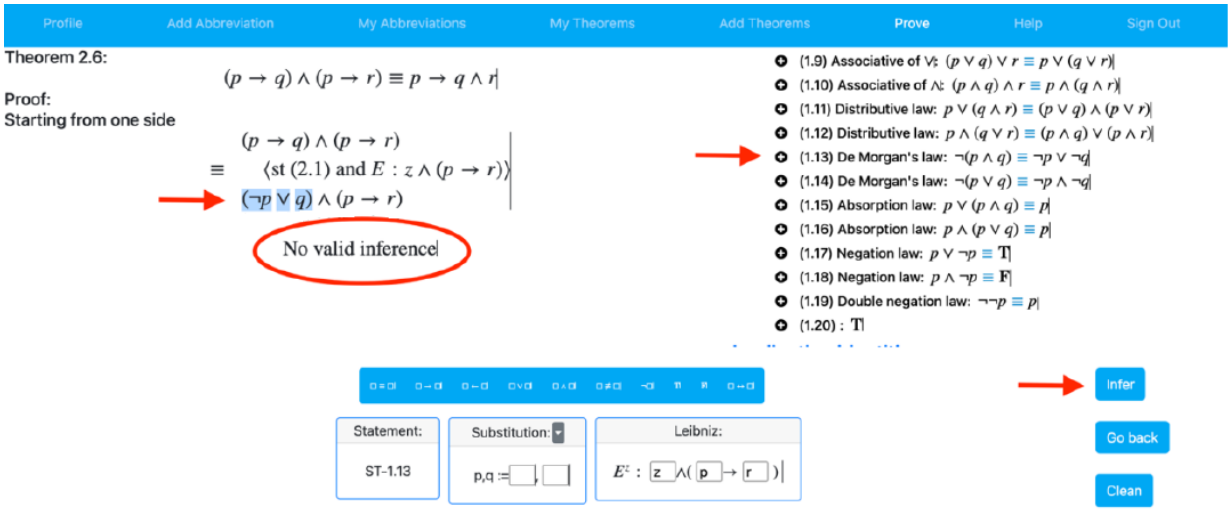


Figure 8: Wrong inference

5.3.2 Statement Instantiation

The instantiation of a statement is done, as explained in the introduction, by means of the substitution operator $[\overline{x} := \overline{Exp}]$. The user selects which operator will be used by filling out the forms found in the substitution field (marked with A in Figure 6). In order for these forms to appear, we must first select the statement to be instantiated.

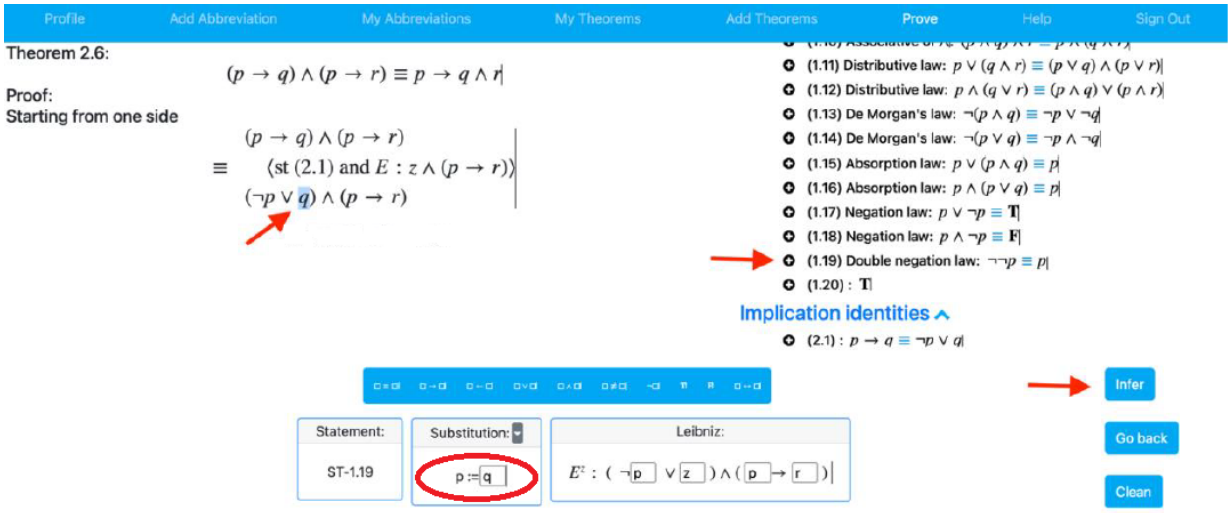


Figure 9: Use of the substitution field

For example, to correctly continue the proof of Figure 8, the double negation law 1.19 must be selected on the proposition q, as indicated in Figure 9. Note that the substitution field in Figure 9 means that the p of 1.19 is to be replaced by q. The effect of making this inference is visualized in Figure 10.

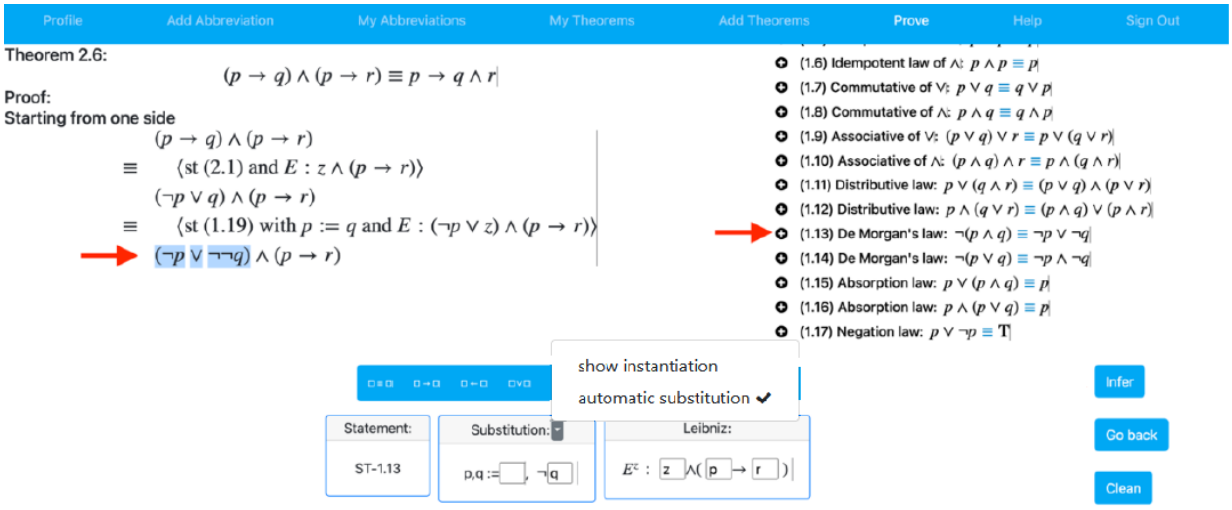


Figure 10: Second inference from the proof of 2.6

Figure 10 also shows that the button above the substitution forms displays two options: "show instantiation" and "automatic substitution".

If the "automatic substitution" option is on, after selecting the statement and underlining what you want to replace, CalcLogic automatically fills in the substitution form. The calculation of the substitution is done with a first-order unification algorithm internally. For example, in Figure 10 the user selected 1.13 and underlined $\neg p \vee \neg\neg p$, so the form is automatically filled in the way that the q in 1.13 should be replaced by $\neg q$.

The "show instantiation" option displays the result of applying the user-entered substitution operator to the selected statement, before making an inference. This is useful for checking if a statement is being

instantiated as desired. For example, Figure 11 shows the result of instantiating statement 1.13 ($\neg(p \wedge q) \equiv \neg p \wedge \neg q$) substituting q for $\neg q$

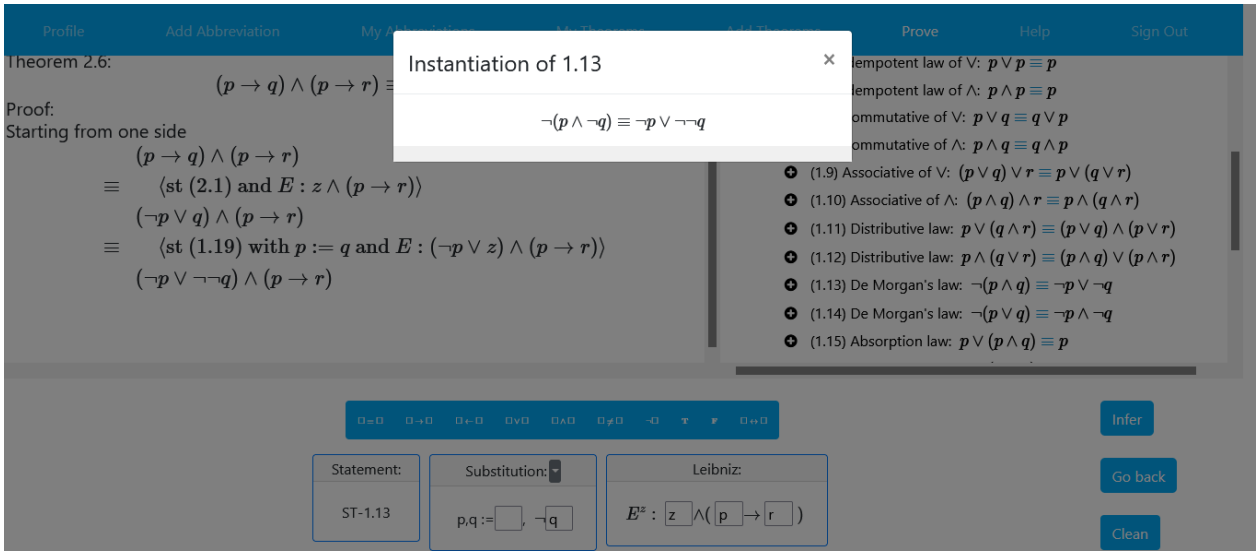


Figure 11: Showing instantiation of statement 1.13 selected by the user

5.3.3 Manual Entry of Formulas

In every field of a form where it is necessary to enter a formula or mathematical expression, the application has a table of buttons for assisted entry of them. In Figure 12, you can see a table where each blue button corresponds to a symbol. These symbols belong the CalcLogic instance for Boolean Algebras: the equality, the constant 0, the constant 1, the $+$, the $'\cdot'$ and the complement $\bar{}$.

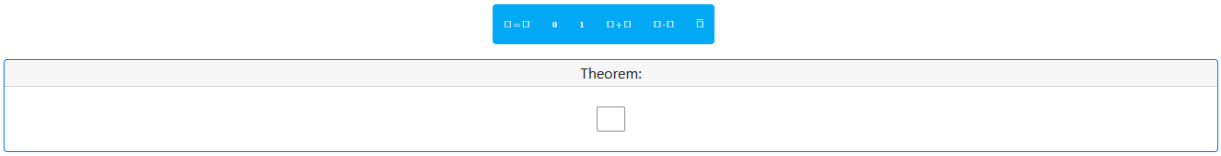


Figure 12: Button table with the symbols of the theory of Boolean Algebras

The cursor must be placed in a form and then use the button table to enter a formula in this form. If the button corresponds to an n-arity operation, when pressing it, the operator is printed on the form and n forms are generated, located in the position where the operator arguments should be (For example the Figure 13 shows the effect of pressing the operator button $'\cdot'$). To continue entering a desired expression, the cursor must be placed in one of the forms generated before and repeat the process recursively (For example, Figure 14 shows the effect of pressing the button $+$ when the cursor was on the second argument of the $'\cdot'$ operator).

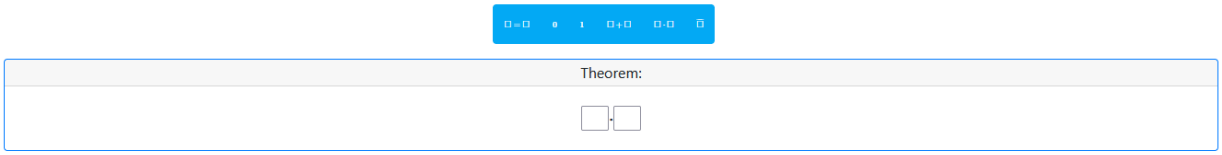


Figure 13: Effect of using the $\cdot$ button with the cursor on an initial form

To reverse the effect of pressing one of the buttons in the table, you must place the cursor in one of the forms generated by the button and press Backspace on the keyboard (Alt+Delete in some browsers). The formula or expression insertion process can be reversed to its initial state by pressing the Clean button.

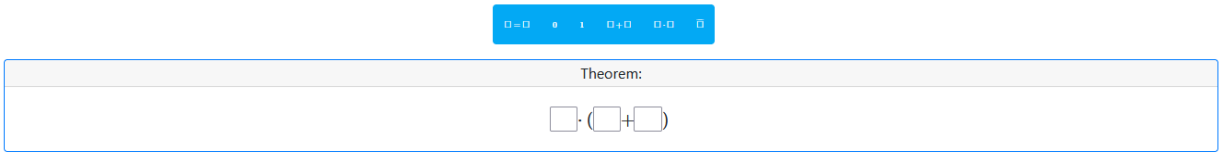


Figure 14: Effect of using the + button with the cursor on the second argument of ·

5.3.4 Starting for One Side

This method can be used only if the formula to be proved is of the form $Exp_1 = Exp_2$ or $Exp_1 \equiv Exp_2$. We call the formulas Exp_1 and Exp_2 the left side of the formula and the right side of the formula respectively. When the method of starting from one side is selected, the application asks you to select which side of the formula you want to start from. For this, left and right sides of the formula are colored blue and selection is made by the user by clicking on one of the sides. Figure 15 shows the user selecting the right hand side of the formula to start the proof from there.

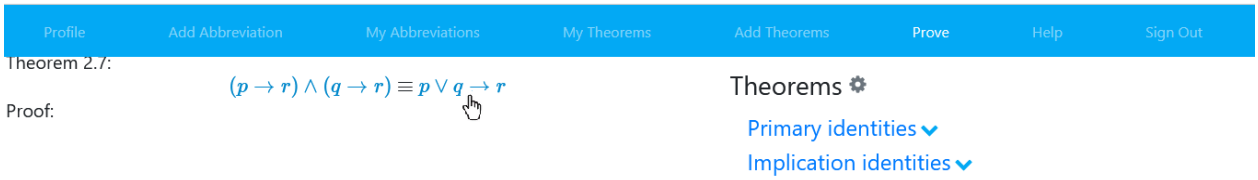


Figure 15: Selecting the right side to start the proof of 2.7

A proof with this proof method ends automatically when, by a sequence of equivalences or equalities, the other side of the formula is reached.

5.3.5 Direct Method

When the direct method is selected to start a proof, the application asks you to select the first line of the proof. The statement that is being proved can be selected, or any other theorem already proven (on the right of the screen). For example, the Figure 16 shows the user selecting the statement 4.1 as the first line of the proof of statement 4.1.

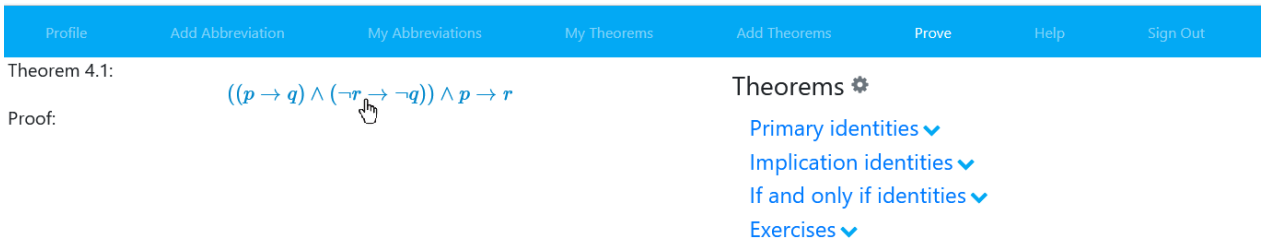


Figure 16: Selecting 4.1 as the first line of the proof of 4.1

The rest of the proof is done by making several one-step inferences to arrive at a goal. If the proof started from the theorem to be proved, the goal is any theorem already proven. If the proof starts from any theorem already proven, the goal is the theorem to be proved. In Figure 17 a proof from the theorem to be proven reaches to an axiom 1.20, and the proof finished.

5.3.6 Weakening-Strengthening

This proof method is not found in the FIU Discrete Mathematics program or in [6]. Then it was not available in the CalcLogic version for educational experiments. However, the functionality of proving using weakening-strengthening is implemented, therefore will be presented below.

Since this method was designed for students who do not follow the book [6], we have preferred to present this section with the original notation of [1], which is how the subsection 4.5 was written, i. e., $\Rightarrow$ and $\Leftarrow$ will be used instead of $\to$ and $\leftarrow$. These notation changes can be easily done in CalcLogic using the symbol configuration module described in Section 5.1.

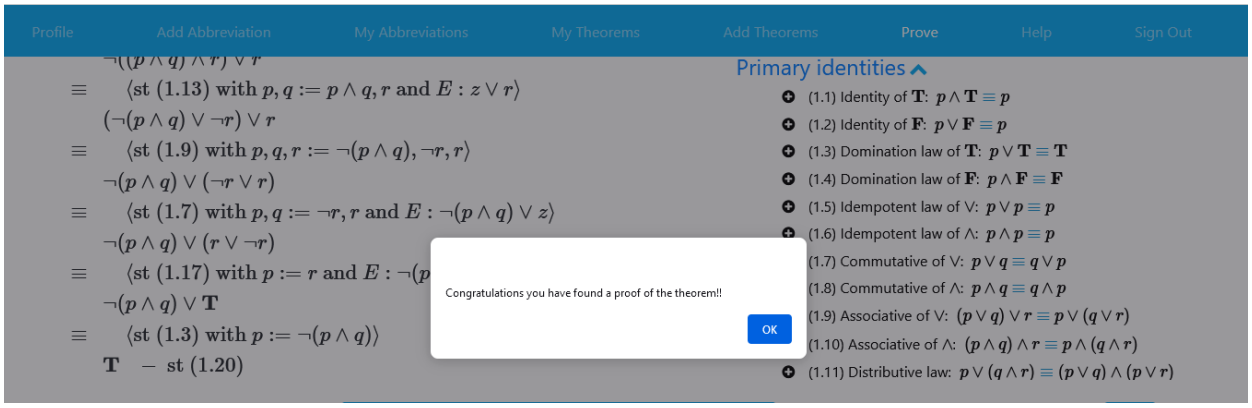


Figure 17: Proof of 4.1 reach to the axiom 1.2

Weakening-strengthening can only be used for formulas of the form $P \Rightarrow Q$ or $P \Leftarrow Q$. When the weakening-strengthening method is selected, the application asks you to select the antecedent or the consequent of the formula, to start the proof from the selected one.

In this proof methods, the $\Rightarrow$ or $\Leftarrow$ inferences (monotonic or anti monotonic inferences) is enabled, that are used by selecting statements of the form $P \Rightarrow Q$ or $P \Leftarrow Q$ respectively, and then filling in, the substitution and Leibniz rule fields in the usual way.

After starting the proof, inferences can be done sequentially as explained in the other methods. Implications can also be inferred as indicated in the previous paragraph. Figure 18 shows the order in which the selections must be done on the interface to execute the inference: 1) Select the theorem $p \Rightarrow p \vee q$ by clicking on the symbol $\Rightarrow$. 2) Underline q to specify the Leibniz expression to use. 3) Click on the infer button to run the inference shown in Figure 19. By doing these steps, CalcLogic automatically performs an anti monotonic inference using the Parity Metatheorem algorithm described in Subsection 6.2.

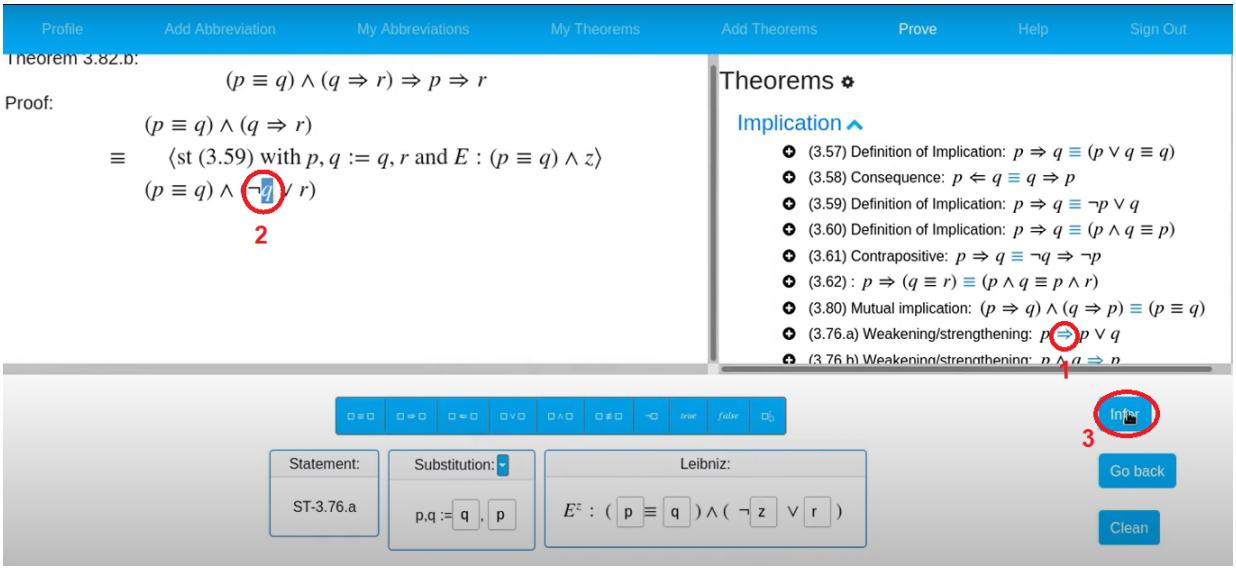


Figure 18: Process for performing an anti monotonic inference step

By doing the process described above several times, you can make nested inferences until you get to prove the theorem. For example, Figure 20 shows two nested monotone inferences.

6 Algorithms

In this section, it is explained the printing algorithm for displaying the formulas and the algorithmic versions of the parity metatheorem and duality metatheorem.

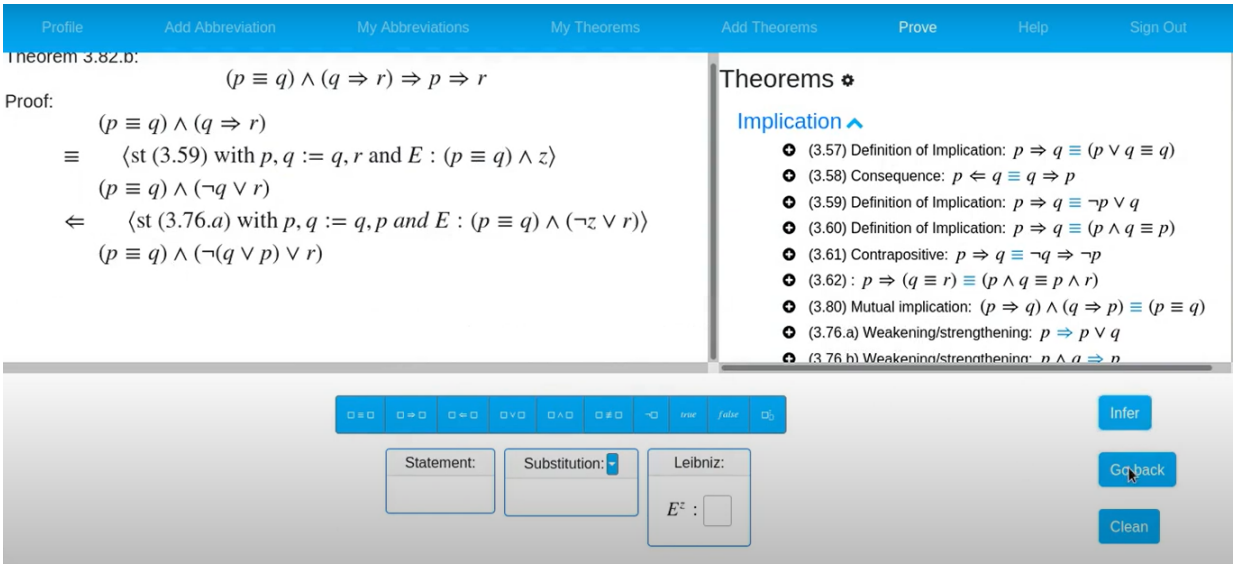


Figure 19: Result of performing an anti monotonic inference step

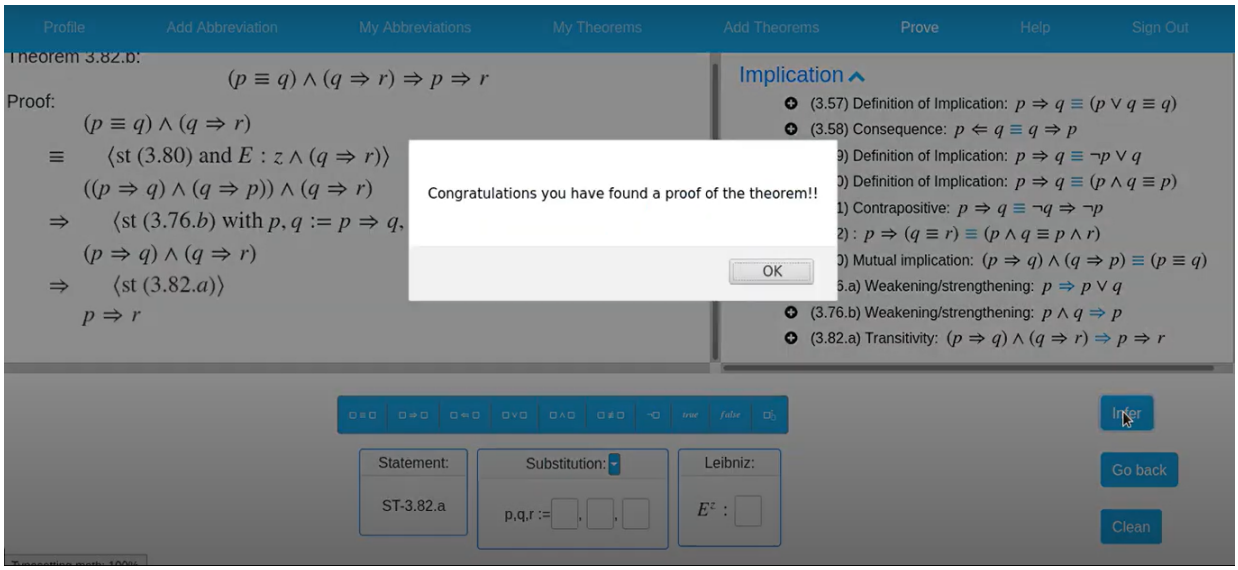


Figure 20: Proof of Theorem 3.82.b using weakening

6.1 Printing Algorithm

Formulas in CalcLogic are created from a set of variables and a set of non-variable symbols $\{C_i\}_{i=1}^{n}$. The non-variable symbols are those listed in the table on the Figure 2. The application internally represents each formula in applicative notation $F\ a_1\ a_2 \ldots a_m$ with $m \geq 0$, where a_j could be a lambda term and F is a symbol C_i or a variable. For each symbol C_i, the administrator must specify the arity $m \geq 0$, the precedence $pre(C_i)$, the LaTeX code L_i in which the symbol displays and the position in which its arguments will be printed. The last attribute can be set by writing a phrase N_i in LaTeX but with the occurrence of the placeholders

`%(op)`, `%(a1)`, `%(aa1)`, `%(na1)`, ..., `%(am)`, `%(aam)`, `%(nam)`.

The above list of placeholders will be abbreviated by $\overline{a}$, and by $\overline{L_i, f(a, \%a)}$ will be abbreviated the list

$$L_i,\ f(a_1, \texttt{\%(a1)}),\ f(a_1, \texttt{\%(aa1)}),\ f(a_1, \texttt{\%(na1)}), \ldots, f(a_m, \texttt{\%(am)}),\ f(a_m, \texttt{\%(aam)}),\ f(a_m, \texttt{\%(nam)}),$$

with f a two arguments function. The notation $N_i\langle \overline{a} := \overline{Exp}\rangle$, means the parallel replacement on the placeholders $\overline{a}$ of N_i by the list of phrases $\overline{Exp}$.

The printing of the formulas in LaTeX notation is done recursively. Each variable X is identified by a single character *char* in ASCII, so if $print(.)$ is the function to print formulas, and $+$ is the String concatenation, then:

- $print(X) := char$,
- $print(X\ a_1 \dots a_m) := "char(" + print(a_1) + "," + \dots + "," + print(a_m) + ")"$ when $m > 0$,
- $print(\lambda x.T) := print(T)$
- $print(C_i\ a_1 \dots a_m) := N_i\langle \overline{a} := \overline{L_i, pAux_{C_i}(a, \%a)}\rangle$ when $m \geq 0$,
  where
  - $pAux_{C_i}(a_j,\texttt{\%(aj)}) := \begin{cases} print(a_j) & if \quad (\exists k, a'_1, \dots, a'_m | : a_j = C_k\ a'_1 \cdots a'_m \wedge \\ & \quad pre(C_k) > pre(C_i)) \vee \\ & \quad a_j \in Var \vee \\ & \quad (\exists k | : a_j = C_k \wedge arity(C_k) = 0) \\ "(" + print(a_j) + ")" & \text{otherwise} \end{cases}$
  - $pAux_{C_i}(a_j,\texttt{\%(aaj)}) := \begin{cases} print(a_j) & if \quad (\exists k, a'_1, \dots, a'_m | : a_j = C_k\ a'_1 \cdots a'_m \wedge \\ & \quad pre(C_k) > pre(C_i)) \vee \\ & \quad (\exists a'_1, \dots, a'_m | : a_j = C_i\ a'_1 \cdots a'_m) \vee \\ & \quad a_j \in Var \vee \\ & \quad (\exists k | : a_j = C_k \wedge arity(C_k) = 0) \\ "(" + print(a_j) + ")" & \text{otherwise} \end{cases}$
  - $pAux_{C_i}(a_j,\texttt{\%(naj)}) := print(a_j)$

The idea of the above algorithm is that variables and constants of arity 0 are always printed without parentheses. The `%(aj)` and `%(naj)` notations means that the j-th argument is always printed with parentheses, or without parentheses respectively, unless it is an operation with higher precedence than C_i. In other hand, `%(aaj)` means that the operation C_i associates in the position of the argument a_j, so the parentheses on a_j are not printed when a_j is another application of the operation C_i.

For example if the arity of C_4 is 2, its LaTeX code L_4 is + and N_4 is `%(aa2) %(op) %(a1)`, then the symbol + is a left-associating infix binary operator. This can be shown by calculating,

$$print(C_4\ y\ x) = (\texttt{\%(aa2) \%(op) \%(a1)})\langle \overline{a} := \overline{L_4, pAux_{C_4}(a, \%a)}\rangle$$

where the operator $\langle \overline{a} := \overline{L_4, pAux_{C_4}(a, \%a)}\rangle$ replace `%(op)`, `%(a1)` and `%(aa2)` for +, $pAux_{C_4}(y,\texttt{\%(a1)})$ and $pAux_{C_4}(x,\texttt{\%(aa2)})$ respectively, and as

$$pAux_{C_4}(y,\texttt{\%(a1)}) = print(y) = y,$$

$$pAux_{C_4}(x,\texttt{\%(aa2)}) = print(x) = x$$

then

$$print(C_4\ y\ x) = x + y.$$

The associativity of + can be seen by nesting the previous term with another C_4 operator and calculating,

$$print(C_4\ z\ (C_4\ y\ x)) = (\texttt{\%(aa2) \%(op) \%(a1)})\langle \overline{a} := \overline{L_4, pAux(a, \%a)}\rangle.$$

In this case the operator $\langle \overline{a} := \overline{L_4, pAux(a, \%a)}\rangle$ replace `%(op)`, `%(a1)` and `%(aa2)` for +, $pAux_{C_4}(z,\texttt{\%(a1)})$ and $pAux_{C_4}(C_4\ y\ x,\texttt{\%(aa2)})$ respectively, and as

$$pAux_{C_4}(z,\texttt{\%(a1)}) = print(z) = z,$$

$$pAux_{C_4}(C_4\ y\ x,\texttt{\%(aa2)}) = print(C_4\ y\ x) = x + y$$

then

$$print(C_4\ z\ (C_4\ y\ x)) = x + y + z.$$

The display contains parentesis if the nested operation is at the first argument. For example

$$print(C_4\ (C_4\ z\ y)\ x) = (\texttt{\%(aa2) \%(op) \%(a1)})\langle \overline{a} := \overline{L_4, pAux(a, \%a)}\rangle = x + (y + z)$$

because at `%(a1)` placeholder the algorithm replace $pAux_{C_4}(C_4\ z\ y,\texttt{\%(a1)}) = "(" + print(C_4\ z\ y) + ")" = "(y + z)"$.

These examples have shown that a symbol in CalcLogic can be set to be left or right associative by placing in its N_i notation the string `%(aa2) %(op) %(a1)` or `%(a2) %(op) %(aa1)` respectively.

As a second example, suppose that C_5 is a operator of arity 2 with N_5 as `%(aa2) %(op) %(a1)`, L_5 as $*$ and $pre(C_5) > pre(C_4)$. The term $C_5\ z\ y$ would be printed as $y * z$ and the term $C_4\ (C_5\ z\ y)\ x$ would be printed as

$$print(C_4\ (C_5\ z\ y)\ x) = (\texttt{\%(aa2) \%(op) \%(a1)})\langle \overline{a} := \overline{L_4, pAux(a, \%a)}\rangle = x + y * z$$

because at `%(a1)` placeholder the algorithm replaces $pAux_{C_4}(C_5\ z\ y, \texttt{\%(a1)}) \overset{pre(C_5)\geq pre(C_4)}{=} print(C_5\ z\ y) = y * z$.

As a third example, suppose that we have the constant symbols C_1, C_2 and C_3 of arity 2, 2 and 0 respectively, with L_1, L_2 and L_3 as `\displaystyle\lim`, `\frac` and `1` respectively and N_1, N_2 and N_3 as `%(op)_{x\to %(na1)} %(na2)`, `%(op){%(na1)}{%(na2)}` and `%(op)`, respectively. The term C_3 would be printed as `1`, the term $C_2\ x\ y$ would be printed as `\frac{x}{y}`, and the term $C_1\ C_3\ (\lambda x.C_2\ x\ y)$ would be printed as `\displaystyle\lim_{x\to 1} \frac{x}{y}`. The display with MathJax would look like $\lim\limits_{x\to 1} \frac{x}{y}$.

As shown in the previous example, the abstraction λx is not printed to the user, but is useful for binding the variable x in some operators. The operators that bound the variable x with λx are $\lim\limits_{x\to a}$, Δ (finite difference) and Σ (Summation). In this way, the user cannot instantiate the variable x of a theorem, if it occurs inside these operators.

The variables of type function can be displayed in CalcLogic. For example the aplicative term $f\ x$ is printed in this way

$$print(f\ x) = "f(" + print(x) + ")" = "f(x)"$$

and $f\ (C_4\ C_3\ x)$ is printed in this form

$$print(f\ (C_4\ C_3\ x)) = "f(" + print(C_4\ C_3\ x) + ")" = "f(x+1)".$$

With this algorithm the finite difference theory could be displayed in the same style of several books like Figure 21.

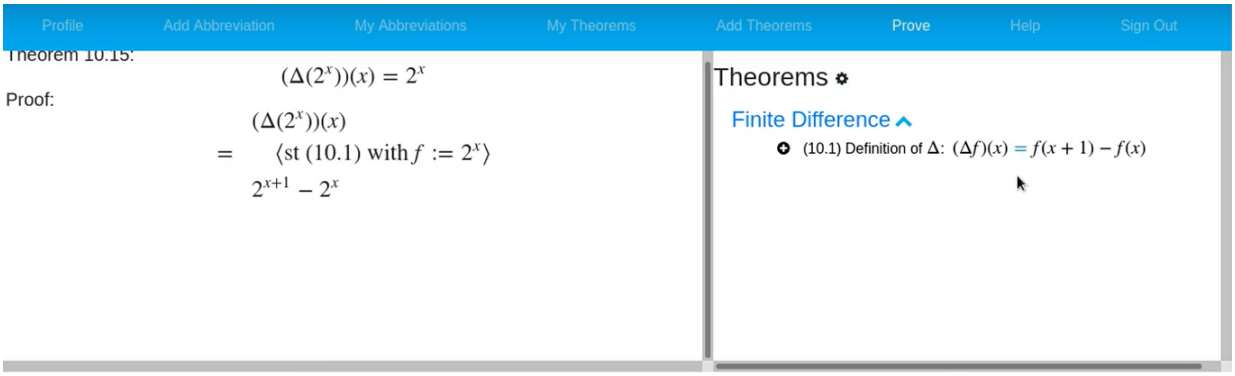


Figure 21: Display of formulas in finite difference theory

CalcLogic configuration is such that all binary operators have as N_i notation the string `%(a2) %(op) %(a1)`, `%(aa2) %(op) %(a1)` or `%(a2) %(op) %(aa1)`. With this configuration a term in applicative notation $C_i\ a_1\ a_2$ is always printed infix by placing the first argument to the right of the operator and the second to the left. This means that when the formula $p \Rightarrow q$ is displayed on the screen, then CalcLogic internally handles it in applicative notation as $\Rightarrow q\ p$ (with the arguments swapped). The reason for this decision comes from the fact that by abstracting the variable x into an expression with a non commutative operator like $x - 3$, the resulting anonymous function $\lambda x.x - 3$, can be given the name $subtractThree(x)$ (subtract 3 to x). The currying expression for $subtractThree(x)$ is $-\ 3$ and when it receives the argument x the resulting expression is $-\ 3\ x$. With this reasoning the arguments of $-\ 3\ x$ occur swapped with respect to the original expression $x - 3$. It was decided that CalcLogic displays formulas consistently with this reasoning.

6.2 Parity Metatheorem

Every metatheorem in CalcLogic is implemented as an algorithm that generates correct derivation trees. The parity metatheorem algorithm runs automatically when the Leibniz field is filled in a proof using the weakening-strengthening method. The algorithm generates derivation trees that are combined with the

current proof, generating a new proof that uses monoticity or anti monoticity. All this happens automatically and transparently to the user.

According to [1] some logical connectors are monotonic with respect to one of their arguments and anti monotonic with respect to others. For this reason, depending on where the z is in Leibniz's expression E, when an op inference is made, it can be of the form $\overset{P}{q}\overset{\Rightarrow\langle\text{hint}\rangle}{}$ or $\overset{P}{q}\overset{\Leftarrow\langle\text{hint}\rangle}{}$ regardless of whether op is $\Rightarrow$ or $\Leftarrow$. However, CalcLogic calculates and displays automatically the correct choice.

This subsection explains the algorithm that the application uses to build the correct derivation tree. For this, the following trees are defined, where op_1 is any of the operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftarrow$. It is taken that $op \in \{\Rightarrow, \Leftarrow\}$ and op' is $\Leftarrow$ if op is $\Rightarrow$ or op' is $\Rightarrow$ if op is $\Leftarrow$:

WSL_1^{P,Q,op,R,op_1} with op_1 any operator between $\wedge$, $\vee$ and $\Leftarrow$

$$\frac{}{\Gamma \vdash ((p\ op\ q) \Rightarrow ((p\ op_1\ r)\ op\ (q\ op_1\ r)))[p,q,r := P,Q,R]}\ \text{Ax id}$$

WSR_1^{P,Q,op,R,op_1} with op_1 any operator between $\wedge$, $\vee$ and $\Rightarrow$

$$\frac{}{\Gamma \vdash ((p\ op\ q) \Rightarrow ((r\ op_1\ p)\ op\ (r\ op_1\ q)))[p,q,r := P,Q,R]}\ \text{Ax id}$$

$WSR_2^{P,Q,op,R}$

$$\frac{}{\Gamma \vdash ((p\ op\ q) \Rightarrow ((r \Leftarrow p)\ op'\ (r \Leftarrow q)))[p,q,r := P,Q,R]}\ \text{Ax id}$$

$WSL_2^{P,Q,op,R}$

$$\frac{}{\Gamma \vdash ((p\ op\ q) \Rightarrow (p \Rightarrow r\ op'\ q \Rightarrow r))[p,q,r := P,Q,R]}\ \text{Ax id}$$

$Neg^{P,Q,op}$

$$\frac{}{\Gamma \vdash ((p\ op\ q) \equiv (\neg p\ op'\ \neg q))[p,q := P,Q]}\ \text{Ax id}$$

If ∇ is a derivation where the root is of the form $P\ op\ Q$, then instead of placing the arguments P, Q, op in the abbreviations of the above trees, we can place ∇ directly to simplify the notation. For example WSL_1^{∇,R,op_1}, WSR_1^{∇,R,op_1}, $WSR_2^{\nabla,R}$, $WSL_2^{\nabla,R}$ and Neg^{∇} will denote WSL_1^{P,Q,op,R,op_1}, WSR_1^{P,Q,op,R,op_1}, $WSR_2^{P,Q,op,R}$, $WSL_2^{P,Q,op,R}$ and $Neg^{P,Q,op}$ respectively.

If ∇' and ∇ are derivation trees where the root of ∇' is of the form $P \equiv Q$ or $P \Rightarrow Q$, then $(\nabla'\nabla)$ is denoted to a new derivation, where equanimity or modus ponens is applied between the two derivations ∇' and ∇. Additionally, it is denoted as $[z]$ the bracket abstraction to abstract the variable z in applicative languages, which in [38] is denoted as $[z]_{BD_{-\eta}}$ (Definition 3.10 of [38]). The bracket abstraction $[z]$ implements the operator λz within combinatory logic. That is, given an expression Exp, the result of $[z]Exp$ is an anonymous function that depends on z. This function is implemented with combinators and the functional application with terms rewriting.

The combinators used for $[z]$ are those of [5]. These combinators are functions whose purpose will be explained in the paragraph below. The combinators of [5] are syntactically of the form Φ_α, where α is a list of symbols. These symbols are b, c and ordered pairs (α_1, α_2) with α_1, α_2 lists built recursively in the same way that α. The definition of $[z]$ is the following:

- $[z]p = \Phi$ if $z = p$,
- $[z]p = \Phi_K p$ if $z \notin p$
- $[z]pq = t_1(q, [z]p)$ if $z \in p \wedge z \notin q$,
- $[z]pq = t_2(p, [z]q)$ if $z \notin p \wedge z \in q$,
- $[z]pq = t_3([z]p, [z]q)$ if $z \in p \wedge z \in q$.

where

$$t_1(q,r_1)=\begin{cases}\Phi_{c\alpha}qw_1\dots w_{\#\alpha} & if \quad r_1=\Phi_\alpha w_1\dots w_{\#\alpha}\\ \Phi & otherwise\end{cases}$$

$$t_2(p,r_2)=\begin{cases}\Phi_{b\alpha}pw_1\dots w_{\#\alpha} & if \quad r_2=\Phi_\alpha w_1\dots w_{\#\alpha}\\ \Phi & otherwise\end{cases}$$

$$t_3(r_1,r_2)=\begin{cases}\Phi_{(\alpha_1,\alpha_2)}w_1\dots w_{\#\alpha_1}w'_1\dots w'_{\#\alpha_2} & if \quad r_1=\Phi_{\alpha_1}w_1\dots w_{\#\alpha_1}\ and\ r_2=\Phi_{\alpha_2}w'_1\dots w'_{\#\alpha_2}\\ \Phi & otherwise\end{cases}$$

The previous bracket abstraction $[z]$ is the same as in [5], but without the recursive rule η and defined with Bunder's algorithm [39], which is more efficient than [5]. The $[z]$ operator converts an expression or formula into a function, for example applying $[z]$ to the formula $\wedge\ z\ \mathbf{T}$ results in $[z](\wedge\ z\ \mathbf{T}) = t_1(\mathbf{T},[z](\wedge\ z)) = t_1(\mathbf{T},t_2(\wedge,[z]z)) = t_1(\mathbf{T},t_2(\wedge,\Phi)) = t_1(\mathbf{T},\Phi_b\ \wedge) = \Phi_{cb}\ \mathbf{T}\ \wedge$, where the combinator Φ_{cb} represents the function $\lambda x,y,z.y\ z\ x$. In Johnsson's technique [34], the action of converting the formula $\wedge\ z\ \mathbf{T}$ into the functional version $(\lambda x,y,z.y\ z\ x)\ \ \mathbf{T}\ \wedge$ is called Lifting and the function $(\lambda x,y,z.y\ z\ x)$ Lifter. The bracket abstraction $[z]$ of [5] applied to an expression or formula Exp, performs a lambda lifting but in combinatory logic, i.e. $[z]Exp = \Phi_\alpha t_1\cdots t_n$ where $t_1,\dots,t_n$ are subterms of Exp and Φ_α is a Lifter (but in combinatory logic).

The term rewriting system that describes the behavior of the functional application to a combinator Φ_α, is defined in [5]. With this nomenclature the following theorem determines the algorithm used to implement the parity metatheorem.

Theorem 1. *Let op be $\Rightarrow$ or $\Leftarrow$, let ∇ be a derivation tree of $\Gamma\vdash P\ op\ Q$ and a term E, with z a variable occurring in E only once, without it being in the scope of a $\equiv$'s arguments. Then a derivation tree $f([z]E,\nabla)$ is recursively constructed whose root is $E[z:=P]\ op_2\ E[z:=Q]$ with op_2 some operator between $\Rightarrow$ and $\Leftarrow$, as follows:*

$f(t,\nabla)=\nabla$ *if t is atomic*

$f(t\ \neg,\nabla) = f(t, Neg^\nabla\nabla)$

$f(t\ (\Rightarrow R),\nabla) = f(t, WSL_2^{\nabla,R}\nabla)$

$f(t\ (o\ R),\nabla) = f(t, WSL_1^{\nabla,R,o}\nabla)$

$f((t\ R)\Leftarrow,\nabla) = f(t, WSR_2^{\nabla,R}\nabla)$

$f((t\ R)\ o,\nabla) = f(t, WSR_1^{\nabla,R,o}\nabla)$ *where t and R are formulas written in applicative language and $o\in\{\Rightarrow,\Leftarrow,\wedge,\vee\}$*

For example, if ∇ derives the axiom $p\Rightarrow p\vee q$, and $E: z\wedge q\Rightarrow p$ which in the CalcLogic backend is written as $\Rightarrow p\ (\wedge\ q\ z)$, then $f([z]E,\nabla) = f([z](\Rightarrow p\ (\wedge\ q\ z)),\nabla) = f(\Phi_{bb}(\Rightarrow p)(\wedge\ q),\nabla) = f(\Phi_{bb}(\Rightarrow p), WSL_1^{\nabla,q,\wedge}\nabla) = f(\Phi_{bb}, WSL_2^{WSL_1^{\nabla,q,\wedge}\nabla,p}(WSL_1^{\nabla,q,\wedge}\nabla)) = WSL_2^{WSL_1^{\nabla,q,\wedge}\nabla,p}(WSL_1^{\nabla,q,\wedge}\nabla)$.

In tree notation the previous result is the following:

$$\dfrac{WSL_2^{WSL_1^{\nabla,q,\wedge}\nabla,p} \qquad \dfrac{\dfrac{}{\Gamma\vdash(p\Rightarrow p\vee q)\Rightarrow(p\wedge q\Rightarrow(p\vee q)\wedge q)}\,WSL_1^{\nabla,q,\wedge} \qquad \dfrac{}{\Gamma\vdash p\Rightarrow p\vee q}\,\nabla}{\Gamma\vdash p\wedge q\Rightarrow(p\vee q)\wedge q}}{\Gamma\vdash(p\wedge q\Rightarrow p)\Leftarrow((p\vee q)\wedge q\Rightarrow p)}$$

where $WSL_2^{WSL_1^{\nabla,q,\wedge}\nabla,p}$ is the tree

$$\dfrac{}{\Gamma\vdash((p\Rightarrow q)\Rightarrow((p\Rightarrow r)\Leftarrow(q\Rightarrow r)))[p,q,r:=p\wedge q,(p\vee q)\wedge q,p]}\,\text{Ax}$$

A derivation like the previous one is considered as an atomic inference from the user's point of view, when doing a weakening-strengthening proof. The above derivation tree was named ∇ in the 4.5 section and is printed to the user with the following vertical notation:

$p\wedge q\Rightarrow p$

$\Leftarrow\quad\langle$st 2 and $E: z\wedge q\Rightarrow p\rangle$

$(p\vee q)\wedge q\Rightarrow p$

when the user selects theorem $p\Rightarrow p\vee q$ with identifier 2 and Leibniz expression $E: z\wedge q\Rightarrow p$. If the user enters an E that does not fulfill the hypothesis of the Theorem 1, the application returns an error message.

In the proof of Theorem 1, the notation $|p|q$ from Definition 2.4 of [38] is used, which is defined recursively as $|p|(qt) := |pq|t$, $|p|q := pq$ and $|p|\epsilon := p$. Also the notation $z\in E$ will denote "z occurs in E".

Proof. If E is an atom then $E=z$ because $z\in E$, in this case $t=\Phi=[z]z$ and therefore $f(t,\nabla)=\nabla$, so that the root of $f(t,\nabla)$ is $P\ op\ Q = E[z:=P]\ op\ E[z:=Q]$.

On the other hand, if $E\neq z$, then the smallest well-formed expressions E where z occur only once without being under the scope of a $\equiv$'s arguments, are the following: $\neg z$, $o\ R\ z$ and $o\ z\ R$ where $o\in\{\Rightarrow,\Leftarrow,\wedge,\vee\}$ and R is a formula. Let's split by cases.

- If $E = \neg z$.

  $t = [z]E = \Phi_b\neg$ so that $f(t, \nabla) = f(\Phi_b, Neg^{\nabla}\nabla) = Neg^{\nabla}\nabla$ and this derivation by equanimity has root $\neg P\ op'\ \neg Q = E[z := P]\ op'\ E[z := Q]$

- If $E = o\ R\ z$ let's split by cases: o is $\Rightarrow$ or $o \in \{\Leftarrow, \wedge, \vee\}$

  - If o is $\Rightarrow$ then $t = [z](\Rightarrow R\ z) = \Phi_b(\Rightarrow R)$, so that $f(t, \nabla) = f(\Phi_b, WSL_2^{\nabla,R}\nabla) = WSL_2^{\nabla,R}\nabla$ and this derivation by modus ponens has root $(P \Rightarrow R)\ op'\ (Q \Rightarrow R)$, that in applicative notation is equals to $(\Rightarrow R\ P)\ op'\ (\Rightarrow R\ Q) = (\Rightarrow R\ z)[z := P]\ op'\ (\Rightarrow R\ z)[z := Q] = E[z := P]\ op'\ E[z := Q]$
  - If $o \in \{\Leftarrow, \wedge, \vee\}$ then $t = [z](o\ R\ z) = \Phi_b(o\ R)$, so that $f(t, \nabla) = f(\Phi_b, WSL_1^{\nabla,R,o}\nabla) = WSL_1^{\nabla,R,o}\nabla$ and this derivation by modus ponens has root $(P\ o\ R)\ op\ (Q\ o\ R)$, that in applicative notation is equals to $(o\ R\ P)\ op\ (o\ R\ Q) = E[z := P]\ op\ E[z := Q]$

- If $E = o\ z\ R$ let's split in two cases: o is $\Leftarrow$ or $o \in \{\Rightarrow, \wedge, \vee\}$

  - If o is $\Leftarrow$ then $t = [z](\Leftarrow z\ R) = \Phi_{cb}R \Leftarrow$, so that $f(t, \nabla) = f(\Phi_{cb}, WSR_2^{\nabla,R}\nabla) = WSR_2^{\nabla,R}\nabla$, and this derivation by modus ponens has root $(R \Leftarrow P)\ op'\ (R \Leftarrow Q)$ that in applicative notation is equals to $(\Leftarrow P\ R)\ op'\ (\Leftarrow Q\ R) = E[z := P]\ op'\ E[z := Q]$.
  - If $o \in \{\Rightarrow, \wedge, \vee\}$ then $t = [z](o\ z\ R) = \Phi_{cb}R\ o$, so that $f(t, \nabla) = f(\Phi_{cb}, WSR_1^{\nabla,R,o}\nabla) = WSR_1^{\nabla,R,o}\nabla$, and this derivation by modus ponens has root $(R\ o\ P)\ op\ (R\ o\ Q)$ that in applicative notation is equals to $(o\ P\ R)\ op\ (o\ Q\ R) = E[z := P]\ op\ E[z := Q]$

Any other term $E \neq z$ can be decomposed in the form $E = t_1[z := t_2]$ with t_2 any of the forms $\neg z$, $o\ R\ z$ and $o\ z\ R$ and z occurring in t_1 only once. The proof is concluded by doing structural induction on t_1.

- If t_1 is atomic.

  Then $z = t_1$ because $z \in t_1$, therefore $E = t_2$ and the proof consists of the verification made previously.

- If t_1 is not atomic.

  Is easy to proof that $t = [z]E = ||\Phi_{\alpha\alpha'}|w|w'$ where $[z]t_1 = |\Phi_\alpha|w$ and $[z]t_2 = |\Phi_{\alpha'}|w'$, where $[z]t_2$ is one of the expressions t previously calculated, for which it has already been shown that f constructs a derivation ∇' such that the root is $t_2[z := P]\ op_2\ t_2[z := Q]$. In this way $f(t, \nabla) = f(||\Phi_{\alpha\alpha'}|w|w', \nabla) = f(|\Phi_{\alpha\alpha'}|w, \nabla') \overset{f\ not\ depend\ of\ \Phi_{\alpha\alpha'}}{=} f(|\Phi_\alpha|w, \nabla')$ and the latter by inductive hypothesis is a derivation whose root is $t_1[z := t_2[z := P]]\ op_2\ t_1[z := t_2[z := Q]] = t_1[z := t_2][z := P]\ op_2\ t_1[z := t_2][z := Q] = E[z := P]\ op_2\ E[z := Q]$.

□

6.3 Duality Metatheorem

The duality metatheorem of Boolean Algebra theory says that if P is a theorem of Boolean Algebra then $P[\cdot, + := +, \cdot]$ is also a theorem. The notation $P[\cdot, + := +, \cdot]$ refers to the statement of the theorem P in which all occurrences of the operators "$\cdot$" and "$+$" are replaced to "$+$" and "$\cdot$" respectively.

The statement $P[\cdot, + := +, \cdot]$ is called the dual theorem of P. The dual theorem of P is always an axiom or a theorem that can be proved by generating a derivation tree with the following recursive algorithm f:

Input: a derivation tree ∇ with root P
Output: a derivation tree with root $P[\cdot, + := +, \cdot]$

- For each leaf A^E of ∇ do:
  - If $E[\cdot, + := +, \cdot]$ is an axiom or theorem already proven then $\nabla := \nabla[A^E := A^{E[\cdot,+:=+,\cdot]}]$
  - If not, and ∇' is the derivation tree of E, then $\nabla := \nabla[A^E := f(\nabla')]$

Where A^E denotes the inference rule "Axiom" selecting the premise E and $\nabla[A^E := A^{E'}]$ denotes the tree resulting from replacing in ∇ all the occurrences of the inference rule A^E by the inference rule $A^{E'}$.

The proof that $f(\nabla)$ is a derivation tree with root $P[\cdot, + := +, \cdot]$ can be done by induction on the structure of the tree ∇.

Group	CalcLogic score	Average	Passing Rate
All	any	65.69	51.7%
open	$x = 1$	72.79	55.3%
try	$x = 2$	74.38	58.6%
a chain	$3 \leq x \leq 5$	78.60	70.0%
solve a few	$6 \leq x \leq 7$	81.00	81.3%
good performance	$x > 7$	82.90	90.0%

Table 1: Average and Passing Rate of exam 1

Group	CalcLogic score	Average	Passing Rate
All	any	66.17	53.4%
open	$x = 1$	68.92	60.5%
try	$x = 2$	70.97	65.5%
a chain	$3 \leq x \leq 5$	77.15	75.0%
solve a few	$6 \leq x \leq 7$	81.63	87.5%
good performance	$x > 7$	85.10	90.0%

Table 2: Average and Passing Rate of exam 2

7 Assessment Experience

The homework that was sent through CalcLogic represented an extra credit of 10 points out of 100. The score was as follows:

- 0 points: Never registered in CalcLogic
- 1 point: Only registered and did not go beyond selecting the proof method to use.
- 2 points: Only one line of the proof appeared in some of the exercises.
- 3-5 points: Made a chain of inferences but did not solve any exercises.
- 6-8 points: Solved more than one exercise.
- 9 points: Solved all but the most difficult.
- 10 points: Solved all the problems.

From a total of 58 students, the average of the grades of the written exams was taken by groups, depending on how much their CalcLogic score was. These groups are the students such that: only opened CalcLogic (1 point), tried to prove (2 points), made a chain of inferences (3 to 5 points), solved some exercises (6 to 7 points), performed well (greater than 7 points) and all students.

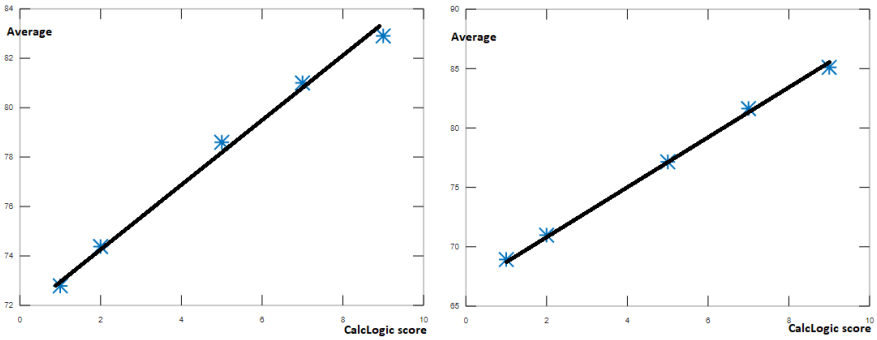
The first exam was about Propositional Logic and the second about Boolean Algebras, with exercises similar to those found in CalcLogic. The average and Passing Rate for the first exam can be found in Table 1.

The average and Passing Rate for the second exam can be found in Table 2.

The “Group” column shows the description of each group of students based on their performance in CalcLogic. The “CalcLogic score” column shows the score obtained by each group in CalcLogic. The “Average” column shows the average of the exam scores by groups. The “Passing Rate” column shows the percentage of students in the group whose exam score was 70 or higher (passing).

The results show a high correlation between the performance in CalcLogic and the written exams. The better the performance in CalcLogic, the better the average in the written exams.

Calculating the line of best fit using the root mean square, it can be seen that the correlation is linear. Figures 22a and 22b show the correlation graph of the score obtained in CalcLogic vs. the average of exam 1 and 2, respectively. In both figures the line of best fit is drawn.



(a) CalcLogic score vs average of exam 1 (b) CalcLogic score vs average of exam 2

8 Conclusions

CalcLogic started as an experiment with students of courses at the Simon Bolivar University [3]. The experiments could be exported to other institutions such as FIU, thanks to the adaptability of the application. By editing configurations, CalcLogic can be presented with distinct notations according to the corresponding literature. Classroom experience has shown that students adapt quickly. Surely this is due to the fact that the formulas are being shown with identical notation to that of the university course books.

For example, in the experiments carried out in [3], a field theory was loaded with division and exponentiation symbols that allowed the use of properties such as: double C, addition and subtraction of exponents, notable products, etc. The formulas for this theory were familiar to the students because they looked identical to the high school textbook [40]. In the same experiment, the axioms of the trigonometric functions were loaded and the formulas looked identical to the high school book [41]. The formulas of the theory of finite differences were identical to the book [42] and those of Boolean Algebra to the book [6]. Two versions of the propositional logic were set up, one using the notation of [6] (used at FIU) and one using the notation of [2] (for a future experiment). Students have become familiar with the notation in these books in high school and college, so CalcLogic is often received with familiarity due to its interface.

On the other hand, the theoretical development that has been exposed here has an algorithmic approach. The translation of formulas in applicative notation to infix notation is an algorithm (Subsection 6.1), the translation of a derivation tree to the vertical notation of a proof is an algorithm, which depends on the method, described in Section 4. In Subsection 4.3 it was explained that the True metatheorem can be understood as an algorithm. The duality metatheorem in Boolean algebra is an algorithm. The parity metatheorem is an algorithm described in Subsection 6.2.

Each classical metatheorem in the Dijkstra-Scholten bibliography has been understood as an algorithm that generates correct derivation trees. With this approach, proving a metatheorem consists in proving that the associated algorithm is correct. Said proof is done in the metalanguage and therefore it is impossible to verify it using CalcLogic.

In other deductive systems there are two inference rules useful for proofs with quantifiers, these are the rules called $\exists$ elimination and $\forall$ introduction. In [2] there are two metatheorems (metatheorem 9.16 and 9.30) that show that both rules are derivable from the direct method. We assert without proof that these metatheorems can also be understood as algorithms. It is proposed for future work to develop these algorithms and implement them on CalcLogic.

The Dijkstra-Scholten system is defined on a typed language. This is because without a type system you cannot distinguish between a predicate and a term. For this reason CalcLogic should have a type system, however the simplest viable product does not need this.

Untyped theories cover the contents of college courses more than expected. Most of the high school math is within field theory (adding some symbols) which is equational. Trigonometry was loaded in CalcLogic as an equational theory without the need to check types. We could write without doing type checking, the theory of finite differences and summations as an equational theory. Boolean Algebra is an equational theory that does not require type checking. Although the design of limit theory in CalcLogic is not yet complete, preliminar experimentation has shown that some theorems can be written within an equational theory without type checking. Finally, the formulas of Propositional Logic can be written without type checking.

With the limitations explained, a large amount of content has been successfully covered. It is hoped that more content will be covered when CalcLogic has a type system. This is because formulas may combine predicates and logical connectors. This work is expected to be carried out in the future and thus cover the contents of Set Theory, Arithmetic, Hoare's Logic, etc.

On the educational side, the results suggest that training using CalcLogic becomes an improvement in the skills to prove Propositional Logic and Boolean Algebra theorems. The results suggest that the answer to research question 1 is: there is an apparent improvement directly proportional to the effort made by the student in the application.

Responding to research question 2. Since the described linear correlation exists, written exams could be reduced and more assessments could be done using CalcLogic. This is because if a student has already performed well in CalcLogic, it can be inferred (because of the correlation) that its performance will be proportional in the written exams, so the need to take them would be reduced. This argument is another reason to keep what was studied in [3], where CalcLogic was recommended as a tool to make remote assessments.

CalcLogic can be understood as a repository of student-done exercises that was automatically checked. This repository is a certificate of the number of correct exercises that the student has done during their studies, which at the same time certifies that the student has been studying correctly. That is why a successful written exam is expected from him.

There is a difference with respect to the traditional learning process and what is said in the previous paragraph. The exercises that the student does at home to study in a traditional way, were invisible to the teacher's eyes. With CalcLogic they become visible to the teacher, with guarantees of correctness, without the need for the teacher to correct them.

References

[1] E. W. Dijkstra and S. C. Scholten, *Predicate calculus and program semantics.* New York: Texts and Monographs in Computer Science, Springer-Verlag, 1990.

[2] D. Gries and F. B. Schneider, *A logical approach to discrete math.* New York, New York: Springer, 1993.

[3] F. Flaviani, "Interactive theorem prover based on calculational logic to assist finite difference and summation learning," *In: Uskov V.L., Howlett R.J., Jain L.C. (eds) Smart Education and e-Learning 2021. Smart Innovation, Systems and Technologies*, vol. 240, pp. 161–172, 2021.

[4] F. Flaviani and W. Carballosa, "Proof assistant based on calculational logic to assist the learning of propositional logic and boolean algebras," in *Proceedings 2022 XLVIII Latin American Computing Conference (CLEI).* Danvers, MA 01923, USA: IEEE, Nov. 2022, pp. 1–9.

[5] S. Broda and L. Damas, "Compact bracket abstraction in combinatory logic," *The Journal of Symbolic Logic*, vol. 62, no. 1, pp. 729–740, 1997.

[6] K. Rosen, *Discrete Mathematics and Its Applications (7 ed.).* New York: McGraw-Hill Higher Education, 2012.

[7] L. Bijlsma and R. Nederpelt, "Dijkstra-scholten predicate calculus: Concepts and misconceptions," *Acta Informatica*, vol. 35, pp. 1007–1036, 1998.

[8] D. Gries and F. B. Scheneider, "Equational propositional logic," *Information Processing Letters*, vol. 53, no. 3, pp. 145–152, 1995.

[9] G. Tourlakis, "A basic formal equational predicate logic - part i," *Bulletin of the Section of Logic*, vol. 29, no. 1, pp. 43–56, 2000.

[10] ——, "A basic formal equational predicate logic - part ii," *Bulletin of the Section of Logic*, vol. 29, no. 3, pp. 75–87, 2000.

[11] ——, "On the soundness and completeness of equational predicate logics," *Journal of Logic and Computation*, vol. 11, no. 4, pp. 623–653, 2001.

[12] ——, "A new foundation of a complete boolean equational logic," *Bulletin of the Section of Logic*, vol. 38, no. 1, pp. 13–28, 2009.

[13] D. Gries and F. Schneider, "Formalizations of substitution of equals for equals," Cornell University, NY, Tech. Rep. TR98-1686, May 1998.

[14] V. Lifschitz, "On calculational proofs," *Annals of Pure and Applied Logic*, vol. 113, no. 1, pp. 207–224, 2001.

[15] E. W. Dijkstra, "The everywhere operator once more," Austin University, TX, Tech. Rep. EWD1086, Nov. 1990.

[16] R. Dijkstra, ""everywhere" in predicate algebra and modal logic," *Information Processing Letters*, vol. 58, no. 5, pp. 237–243, 1996.

[17] D. Gries and F. B. Scheneider, "Adding the everywhere operator to propositional logic," *Journal of Logic and Computation*, vol. 8, no. 1, pp. 119–130, 1998.

[18] D. Gries, "Foundations for calculational logic," *Mathematical Methods in Program Development, NATO ASI Series F: Computer and System Sciences*, vol. 8, pp. 83–126, 1997.

[19] J. Girard, P. Taylor, and Y. Lafont, *Proof and types.* Cambridge, London: Cambridge university press, 1989.

[20] H. Simmons, *Derivation and Computation: taking the Curry-Howard correspondence seriously.* Cambridge, London: Cambridge university press, 2000.

[21] M. SØrensen and P. Urzyczyn, *Lectures on the Curry-Howard isomorphism.* Amsterdam, The Netherlands: Elsevier, 2006.

[22] K. Broda. (2009) slides of automated reasoning course. imperial college, london. [Online]. Available: https://www.doc.ic.ac.uk/~kb/MACTHINGS/SLIDES/0LIntro4up.pdf

[23] T. Nipkow, L. Paulson, and M. Wenzel. (2015) Isabelle/hol: A proof assistant for higher-order-logic. [Online]. Available: http://isabelle.in.tum.de/dist/Isabelle2015/doc/tutorial.pdf

[24] A. Mercer, A. Bundy, H. Duncan, and D. Aspinall, "Pg tips: A recommender system for an interactive theorem prover," in *In Mathematical User-Interfaces Workshop (MathUI 2006)*, Oxford, London, Aug. 2006, pp. 290–294.

[25] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development:Coq Art: The Calculus of Inductive Constructions.* Berlin, Heidelberg: Springer-Verlag, 2004.

[26] W. Kahl, "The teaching tool calccheck: A proof-checker for gries and schneider's "logical approach to discrete math"," *Certified Programs and Proofs, LNCS, Springer*, vol. 7086, pp. 216–230, 2011.

[27] ——, "Calccheck: A proof checker for teaching the "logical approach to discrete math"," *in: Avigad J., Mahboubi A. (eds) Interactive Theorem Proving. Proc. 9th ITP Oxford. Lecture Notes in Computer Science*, vol. 10895, pp. 324–341, 2018.

[28] A. Mendes and J. F. Ferreira, "Towards verified handwritten calculational proofs," *in: Avigad J., Mahboubi A. (eds) Interactive Theorem Proving. Proc. 9th ITP Oxford. Lecture Notes in Computer Science*, vol. 10895, pp. 432–440, 2018.

[29] K. A. A. Gamage, E. K. D. Silva, and N. Gunawardhana, "Online delivery and assessment during covid-19: safeguarding academic integrity," *Educ. Sci.*, vol. 10, no. 11, pp. 301–324, 2020.

[30] S. Böhne and C. Kreitz, "Learning how to prove: From the coq proof assistant to textbook style," in *P. Quaresma and W. Neuper (Eds.): 6th International Workshop on Theorem proving components for Educational software (ThEdu 17) EPTCS 267*, Gothenburg, Sweden, Aug. 2017, pp. 1–18.

[31] J. Villadsen, F. Halkjaer, and A. Schlichtkrull, "Natural deduction assistant (nadea)," in *In Pedro Quaresma & Walther Neuper, editors: Proceedings 7th International Work-shop on Theorem proving components for Educational Software (ThEdu), EPTCS 290*, Oxford, UK, Jul. 2019, pp. 14–29. [Online]. Available: https://doi.org/10.4204/EPTCS.290.2

[32] S. Schönfinkel, "über die bausteine der mathematischen logik," *Mathematische Annalen*, vol. 92, pp. 305–316, 1924.

[33] H. Curry, F. R., and C. W., *Combinatory Logic, volume 1.* North-Holland, 1958.

[34] T. Johnsson, *Lambda lifting: transforming programs to recursive equations.* In Conference on Functional Programming Languages and Computer Architecture, Nancy. Jouannaud (editor). LNCS 201. Springer Verlag, 1985.

[35] F. Baader and T. Nipkow, *Term Rewriting and All That.* Cambridge: Cambridge University Press, 1998.

[36] J. Bohórquez, "Intuitionistic logic according to dijkstra's calculus of equational deduction," *Notre Dame Journal of Formal Logic*, vol. 49, no. 4, pp. 361–384, 2008.

[37] Mathjax documentation. [Online]. Available: https://docs.mathjax.org/en/latest

[38] F. Flaviani and E. Tahhan, "Criteria for bracket abstractions design," *Electronic Notes in Theoretical Computer Science*, vol. 349, pp. 25–48, 2020.

[39] M. Bunder, "Expedited broda-damas bracket abstraction," *Journal of Symbolic Logic*, vol. 64, no. 4, pp. 1850–1857, 2000.

[40] A. Baldor, *Algebra de Baldor.* Publicaciones Cultural, 2002.

[41] J. Hoffmann, *Selección de Temas de Matemáticas.* Sphinx, 1995.

[42] F. Scheid and R. Di Costanzo, *Métodos Numéricos.* Mexico: McGraw-Hill, 1991.