Prolog–Blockly Playgrounds for Logical Reasoning through Natural Deduction

Joaquín Arias[1], Alberto Scocco[2]

[1]CETINIA, Universidad Rey Juan Carlos, Móstoles, 28933, Spain
[2]Università Politecnica delle Marche, Ancona, 60131, Italia

Abstract

Mathematical and computational logic are basic subjects in most university degrees related to Computer Science. Not only are they fundamental in the study of programming languages, artificial intelligence, and the design and verification of hardware and software systems, but they also constitute the basis for logical reasoning. The process of acquiring these logical skills also contributes to the development of fundamental metacognitive abilities, which have significant implications for one's approach to problem setting and problem solving, as well as for the structuring and expression of thought. However, students do not immediately recognize the connections between logic and what they will need to learn (and apply) in their future studies. To mitigate this feeling of disconnect between basic and more advanced content, we have developed DeduccionNatural[.pl, .blockly], two open-source tools built with the Ciao Prolog Playground and Google's Blockly library. Available online and at https://github.com/Xuaco/DeduccionNatural, these tools allow users to verify the correctness of natural deduction proofs in propositional logic. In DeduccionNatural, the proofs are programs, the inference rules are functions of a predefined library, and the derived rules are subroutines (to refactor the proof) defined by the users/students. We have used the Prolog version of DeduccionNatural in the Cybersecurity Engineering and Artificial Intelligence degrees at Universidad Rey Juan Carlos (URJC) since 2021, and since 2025 we have also incorporated the Blockly-based version. Statistical evaluations of teaching experiences using the Prolog version have been published in educational conferences and received the third prize in the XI Edición del Premio Docentes Innovadores 2024 at URJC, while the evaluation of the Blockly-based version is ongoing.

Keywords

Teaching tools, Logical Reasoning, Natural Deduction, Prolog, Blockly

1. Introduction

The teaching materials presented in this article are part of the subject of Logic, which deals with the science that studies the formal validity of reasoning. Through the formalization of language and its basic rules, it provides the necessary tools to be able to rigorously deal with and solve problems that have their origins and applications in all areas of the sciences [1, 2, 3].

Various studies in cognitive psychology have demonstrated a direct correlation between the use of formal logic (e.g., quantifiers, negations, and implications) and problem-solving skills. Compared to those who rely solely on informal reasoning, individuals who master formal logical language demonstrate superior abilities in defining problem boundaries, identifying relevant variables, and developing solutions [4]. Conversely, deficiencies in these skills (e.g., in the negation of quantifiers) are proven obstacles to analyzing causes and formulating correct solutions [5, 6] This confirms that logic is a prerequisite for the development of critical and strategic thinking. This also points to the fact that studying logic has implications well beyond STEM, as it serves as a cornerstone for developing cognitive faculties and refining how thought is structured and expressed [6, 7].

It is therefore natural that mathematical and computational logic are basic subjects in most university degrees related to Computer Science. Their study usually has the difficult task of compensating for possible deficiencies in the students' previous training. In addition, it serves to improve the three basic

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026), co-located with the Federated Logic Conference (FLoC 2026), July 18–19 and 24–25, 2026, Lisbon, Portugal.
joaquin.arias@urjc.es (J. Arias); a.scocco@pm.univpm.it (A. Scocco)
https://www.ia.urjc.es/GIA/joaquin-arias (J. Arias)
0000-0003-4148-311X (J. Arias); 0009-0001-7767-3452 (A. Scocco)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-03

CEUR-WS.org/Vol-4311/PEG_regular_14.pdf

skills of reading, mathematics and science (assessed in the Spanish context through the PISA 2018 report) and the capacity for analytical and critical reasoning/thinking. One of the main educational objectives of logic courses is "the ability to acquire, formalize and represent human knowledge in a computable form for problem solving by means of a computer system in any field of application".

From our point of view, a main objective of the subject of Logic is to introduce some systems of formal proof. For most students, it is not immediate to correctly justify the validity of a line of reasoning or the solution of a particular problem. The deductive methods of mathematical logic are at the basis of the automatic proof of theorems. However, in recent years we have observed that it is not easy to get first-year students to understand the importance of logic and its applications to programming.

Because of its more intuitive formulation, in our subject we decided to present Gentzen's method of Natural Deduction [8, 3, 1]. Natural deduction aims to capture the way in which people naturally reason when constructing mathematical proofs: starting from premises and applying the rules of inference, the validity of the corresponding conclusion is proved.

> Example 1. Adapted from [1, p. 46, Chapter 2], consider the following natural deduction problem:
> $$T[s \land p \lor q, p \to \neg r, q \to \neg r] \vdash s \land \neg r$$
> The exercise requires proving that the formula $s \land \neg r$ is a logical consequence of the premises $s \land (p \lor q)$, $p \to \neg r$, and $q \to \neg r$. Intuitively, the first premise establishes that s holds and that either p or q is true, while the remaining premises guarantee that, regardless of whether p or q holds, the conclusion $\neg r$ can be derived. Therefore, combining these results, the target formula $s \land \neg r$ follows by natural deduction.

As we will see later, inference rules can be interpreted as predefined functions with a particular API (application programming interface), and derived rules (proof patterns based on the inference rules) are then subroutines that the user can define for reuse in several proofs without having to repeat a particular proof pattern, e.g., *ModusTollens* from [1, pp. 51–52, Chapter 2].

In this context, the objective of the tools we present, called DeduccionNatural.pl (the Prolog version) and DeduccionNatural.blockly (the Blockly-based version), is to facilitate the learning of natural deduction considering that the proofs are programs that invoke predefined functions, while allowing students to become familiar with the Ciao Prolog playground and to explore natural deduction through a playful and interactive Blockly interface based on Google's Blockly library (https://blockly.com).

In Section 2 we present the theoretical framework and related work of this paper. In Section 3 and Section 4 we present the main contributions of this work, the tools DeduccionNatural.pl and DeduccionNatural.blockly:

- Although DeduccionNatural.pl could be implemented in any programming language, the use of Prolog visualizes its expressiveness.
- The use of Google's Blockly library provides an intuitive visual programming environment based on drag-and-drop blocks, which significantly reduces syntax errors and facilitates learning and interaction for non-expert with limited computing backgrounds.
- Being open-source, they can be adapted to suit the approach of other teachers.
- We have manuals and video tutorials to facilitate their use (at the moment only in Spanish).

In addition, we have been able to verify that it is possible to incorporate its use in the subject of Logic, and since it has a very low learning curve and does not require previous installation, its incorporation can be done without the need to alter the syllabus designed in the teaching guides. And finally, in Section 5 we present conclusions and activities where students have implemented logic programs related to the subject of Logic.

Previously, in JENUI'23 [9], and SIIE'23 [10] we presented preliminary results of the use of DeduccionNatural.pl in the development of practical activities in the Cybersecurity Engineering degree and in the Artificial Intelligence degree of the Universidad Rey Juan Carlos (during the academic year 2021–2022). Additionally, we received the third prize in the XI Edición del Premio Docentes Innovadores 2024 at URJC.

2. Theoretical framework

We assume that the reader has some familiarity with the contents of Logic subjects in the context of Computer Science related degrees. In Section 2.1 we will recall Gentzen's definition of Natural Deduction system for propositional logic. In Section 2.2 we give a brief description of related work and tools that have been or could be used in the teaching of Logic.

2.1. Gentzen's natural deduction

A formal demonstration (or proof) system is defined as $S = (A, F, X, R)$, where:

- A is the alphabet of the system: the set of symbols that can be used.
- F is the set of syntax rules: define well-formed formulas (i.e., syntactically correct formulas).
- X is the set of axioms: the formulas that are valid by definition.
- R is the set of inference rules: transformation rules that allow the derivation of a formula, the conclusion (or consequent) from a set of formulas (either premises or previously derived formulas).

Deductive reasoning is based on a set of formulas $\Phi = \{\varphi_1,\ \varphi_2,\ \ldots,\ \varphi_n\}$, called the premises, and of a formula φ, called the conclusion (or the consequent) of the deduction. A deduction is correct (or valid) if the conclusion φ is a logical consequence of the set of premises Φ, which is denoted as $\Phi \vdash \varphi$.

Gerald Gentzen's (1934-1935) system of natural deduction [8] is, from our perspective, the proof system that best models informal or intuitive reasoning. For propositional logic, this system is based on deductions and consists exclusively of inference rules, with no axioms. The system is defined by ten inference rules (one introduction rule and one elimination rule for each logical connective $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ in its alphabet). These rules are presented below:

(I $\neg$) **Rule of introduction of negation.**
Method of demonstration by contradiction.

$$\{\varphi \rightarrow \psi \wedge \neg\psi\} \vdash \neg\varphi$$

(E $\neg$) **Rule of elimination of negation.**

$$\{\neg\neg\varphi\} \vdash \varphi$$

(I $\wedge$) **Conjunction introduction rule.**
It is deduced from two formulas.

$$\{\varphi, \psi\} \vdash \varphi \wedge \psi$$

(E $\wedge$) **Conjunction elimination rule.**
From a conjunction we deduce the formula from the left (a) or from the right (b).

$$\{\varphi \wedge \psi\} \vdash \varphi \quad \text{(a)}$$
$$\{\varphi \wedge \psi\} \vdash \psi \quad \text{(b)}$$

(I $\vee$) **Rule of introduction of disjunction.**
The disjunction with ψ is deduced by adding it on the right (a) or from the left (b).

$$\{\varphi\} \vdash \varphi \vee \psi \quad \text{(a)}$$
$$\{\varphi\} \vdash \psi \vee \varphi \quad \text{(b)}$$

(E $\vee$) **Disjunction elimination rule.**
Demonstration method by cases.

$$\{\varphi \vee \psi, \varphi \rightarrow \chi, \psi \rightarrow \chi\} \vdash \chi$$

(I $\rightarrow$) **Implication introduction rule**.
1) We suppose the auxiliary premise φ.
2) If we deduce the formula ψ, by supposing φ, then we prove the validity of $\varphi \rightarrow \psi$.

$$\{\varphi(supuesto), \psi\} \vdash \varphi \rightarrow \psi$$

(E $\rightarrow$) **Implication elimination rule.**

$$\{\varphi \rightarrow \psi, \varphi\} \vdash \psi$$

(I $\leftrightarrow$) **Bi-Implication introduction rule.**

$$\{\varphi \rightarrow \psi, \psi \rightarrow \varphi\} \vdash \varphi \leftrightarrow \psi$$

(E $\rightarrow$) **Bi-Implication elimination rule.**

$$\{\varphi \leftrightarrow \psi\} \vdash \varphi \rightarrow \psi \quad \text{(a)}$$
$$\{\varphi \leftrightarrow \psi\} \vdash \psi \rightarrow \varphi \quad \text{(b)}$$

The exclusive use of deductions presupposes reasoning that depends on a set (which may be empty) of premises. Because of its simplicity, its syntactic character and because it is equivalent to the interpretive theory of propositional logic, the system of natural deduction represents a good example of how symbolic logic can be applied to computer science.

Example 2 (Cont. Example 1). A possible proof in natural deduction establishing the validity of $T[s \wedge p \vee q,\ p \rightarrow \neg r,\ q \rightarrow \neg r] \vdash s \wedge \neg r$ is the following:

1.	$s \wedge p \vee q$	$premisa$
2.	$p \vee q$	$E_{\wedge}(1)$
3.	$p \rightarrow \neg r$	$premisa$
4.	$q \rightarrow \neg r$	$premisa$
5.	$\neg r$	$E_{\vee}(2,3,4)$
6.	s	$E_{\wedge}(1)$
7.	$s \wedge \neg r$	$I_{\wedge}(5,6)$

Each line in the proof is numbered to identify the corresponding premise or derived formula. The keyword *premisa* introduces premises into the proof so they can subsequently be referenced by their corresponding index. E.g., the premise, $s \wedge p \vee q$, appears on line 1, therefore, in lines 2 and 6 the inference rule $I_{\wedge}(1)$ refers to that formula.

In a similar way, at line 5, the inference rule $E_{\vee}(2,3,4)$ applies to the formula $p \vee q$, together with the premises $p \rightarrow \neg r$, and $q \rightarrow \neg r$. From these, we can derive $\neg r$. Finally, at line 7, we infer the consequent $s \wedge \neg r$. Therefore, the validity of the argument is proved.

2.2. Related work

In the study of computational logic there are several papers showing the similarities between the process of natural deduction proof and programming [11]. In addition, in the context of natural deduction, tools with varying features and levels of complexity have been developed such as Taut-Logic (https://www.taut-logic.com), Fitch (http://logic.stanford.edu/logica), the Assistant for Natural Deduction (ADN) developed at the University of Alicante [12], the Symbolic Deductive Logic (LSD) developed at the University of Girona [13], and others, such as those described in [14].

However, these tools do not highlight the similarities between natural deduction and programming and, although most of them are open-source, their source code is not always easy to access (if they are operational, which not all of them are) to adapt them. Moreover, such adaptation would not be straightforward because the implementations include ad hoc user interfaces and modifications to the logic of the systems would involve adapting the interface.

Based on the analysis of these tools, we have implemented DeduccionNatural.pl with the following goals: (i) to create a simple program (it has less than 300 lines of code and demos can be written with few lines of code), (ii) to distribute it under an open-source license, (iii) to implement it in Ciao Prolog, an (open-source) Prolog interpreter, and (iv) to allow it to be executed directly in any web browser using the Ciao Prolog playground [15]. The following link, http://platon.etsii.urjc.es/~jarias/DN/DeduccionNatural.pl, gives direct access to the 2023 version in the browser, without prior installation.

Finally, we have consulted some of the existing proposals for formalizing proofs in natural deduction and examples of logical games used in the teaching of Logic:

- Pepa Hernández (http://www.dia.fi.upm.es/es/phernan, UPM), proposes the inference rules and the indentation-based format for the proofs used in this work.
- José A. Alonso Jiménez (https://www.cs.us.es/~jalonso, US), has a multitude of works on Logic, from notes to introductory exercises to Prolog.[1]
- José Morales (https://jfmc.github.io, UPM), uses theorem provers, for example, to prove the logic problem of the *wolf, sheep and lettuce* with Z3 [16].

As a result, we have decided to use indentation as opposed to other proposals (such as those based on the Huth/Ryan notation or the Fitch notation, with boxes or lines) because it is closer to the strategy used in different programming languages (e.g., C and Python). Note that while in C indentation is used to facilitate reading, in Python it is used to define code blocks (see https://www.w3schools.com/python/python_syntax.asp).

[1]The format for representing assumptions differs from the format we use (although they are equivalent). The introduction of constants and variables in first-order logic is also different, but is not relevant to this paper.

```
% Conjunction
'I' and (A, B) :-
    formula(A, FA),
    formula(B, FB),
    add_formula(FA and FB).
'E' and a(A) :-
    formula(A, FA and _FB),
    add_formula(FA).
'E' and b(A) :-
    formula(A, _FA and FB),
    add_formula(FB).
% Disjunction
'E' or (A, B, C) :-
    formula(A, FB or FC),
    formula(B, FB --> F),
    formula(C, FC --> F),
    add_formula(F).
'I' or a(A, Formula) :-
    formula(A, FA),
    add_formula(FA or Formula).
'I' or b(Formula, B) :-
    formula(B, FB),
    add_formula(Formula or FB).
% Negation
'I' ! (A) :-
    formula(A, FA --> B and ! B),
    add_formula(! FA).
'E' ! (A) :-
    formula(A, ! ! FA),
    add_formula(FA).
% Implication
'E' --> (A, B) :-
    formula(A, FB --> FC),
    formula(B, FB),
    add_formula(FC).
'I' --> (A, B) :-
    opened(A),
    formula(A, FA),
    valid(B),
    formula(B, FB),
    close_assumption(A),
    add_formula(FA --> FB).

'Supuesto'(FA) :-
    increase_tab,
    add_formula(FA),
    counter(C),
    assert(opened(C)).

% Bi-Implication
'I' <-> (A, B) :-
    formula(A, FA --> FB),
    formula(B, FB --> FA),
    add_formula(FA <-> FB).
'E' <-> a(A) :-
        formula(A, FA <-> FB),
        add_formula(FA --> FB).
'E' <-> b(A) :-
        formula(A, FA <-> FB),
        add_formula(FB --> FA).
```

Figure 1: Extract from the code of DeduccionNatural.pl

3. Design and implementation of DeduccionNatural.pl

In this section we explain the implementation and operation of DeduccionNatural.pl. First, in Section 3.1 we propose a translation and implementation of the operators, given a concrete precedence criterion. In Section 3.2 we show the interface and part of that of the inference rules. In Section 3.3 we explain how to use the program to check if a proof is correct, and in Section 3.4 how to define and use derived rules to factorize proofs.

3.1. Syntax and precedence

The alphabet of propositional logic is composed of proposition symbols, connectives and a precedence criterion (see [1, 3]). Details and the implementation of this alphabet in Prolog is explained below:

- Propositional symbols are represented directly as atoms (words beginning with a lowercase letter). For example, $p, q, r, p, p1, p2, \ldots$.
- Logical connectives are translated into operators:

Connective	$\neg$	$\vee$	$\wedge$	$\rightarrow$	$\leftrightarrow$
Operator	`!`	`or`	`and`	`-->`	`<->`

- The precedence criterion of the logical connectives, according to [1], establishes that $\neg$ has higher precedence than $\wedge$ and $\vee$, that these have higher precedence than $\rightarrow$ and $\leftrightarrow$, and that at equal precedence they are associated by the right. The implementation of the operators considering this criterion[2] is:

[2]There are authors who define different precedence criteria, but their adaptation is easy (see [9] for details).

```
1 :- op(200, fy, !).
2 :- op(400, xfy, [and, or]).
3 :- op(600, xfy, [-->, <->]).
```

where the first argument of `op/3` defines the precedence (the lower the index the higher the precedence), the second specifies the type of operator and how it is associated (the `f` determines its position, the `x` is a term of lower precedence and the `y` is a term of lower or equal precedence[3]), and the third defines the name of the operator or operators.

With this implementation, of only three lines of code, we can check if a formula is well formed and suppress redundant parentheses. For example, by invoking `?- X = s and (p or q).`, Prolog returns `X = s and p or q` because since `and` and `or` have the same precedence and are associated from the right, the parentheses are not necessary. Try this (and other examples) following this link.

3.2. The ten inference rules

Table 1 shows the interface of the ten inference rules in DeduccionNatural.pl. The inference rules `'I'`, `'E'` and `'Supuesto'` are enclosed in quotation marks so that Prolog treats them as atoms (and not as variables). In addition, to facilitate the readability of the rules, we have taken advantage of the infix definition of the connectives (overloading `!` as a binary operator).

Table 1
Inference rules interface.

Op	Introducción		Eliminación	
`and`	`'I' and (_,_)`		`'E' and a(_)`	`'E' and b(_)`
`or`	`'I' or a(_,F)`	`'I' or b(F,_)`	`'E' or (_,_,_)`	
`!`	`'I' ! (_)`		`'E' ! (_)`	
`-->`	`'Supuesto'(F)`	`'I' --> (_,_)`	`'E' --> (_,_)`	
`<->`	`'I' <-> (_,_)`		`'E' <-> a(_)`	`'E' <-> b(_)`

Figure 1 shows their implementation in Prolog. Note that the number of arguments (used to refer formulas' indexes or new formulas) in each rule corresponds to the number of premises of the inference rules defined in Section 2.1. For example, `'I' ! (_)`, has only one argument, and its implementation (lines 25-27 in Figure 1) maps the inference rule $\{\varphi \to \psi \wedge \neg\psi\} \vdash \neg\varphi$, while `'E' or (_,_,_)`, has three arguments, and its implementation (lines 13-17 in Figure 1) maps the inference rule $\{\varphi \vee \psi, \varphi \to \chi, \psi \to \chi\} \vdash \chi$.

For rules with two possible consequents, we use the functors `a/1` and `b/1` respectively. For example, `'E' and a(_)` represents $\{\varphi \wedge \psi\} \vdash \varphi$, while `'E' and b(_)` represents $\{\varphi \wedge \psi\} \vdash \psi$. And the rules `'I' or a(_,F)`, `'I' or b(F,_)`, and `'Supuesto'(F)` allow the user to enter new formulas by instantiating `F` with the desired formula.

Note that, thanks to the expressiveness of Prolog, the implementation presented in Figure 1 is a one-to-one translation of the inference rules listed in Section 2.1.

3.3. Instantiation (and verification) of proofs

To execute DeduccionNatural.pl and run the examples, we recommend using the Ciao Prolog playground by following the link http://platon.etsii.urjc.es/~jarias/DN/DeduccionNatural.pl. Figure 2 shows a screenshot of the execution of DeduccionNatural.pl in the Ciao Prolog Playground.

Example 3 (Cont. Example 2). Fig. 3b shows the instantiation of `main/3` following the proof proposed in Example 2 (which we reproduce again in Fig. 3a for the reader's convenience). The first argument of `main/3` is the list with the three premises, the second is the consequent and

[3]The position of the term `y` indicates the association criterion.

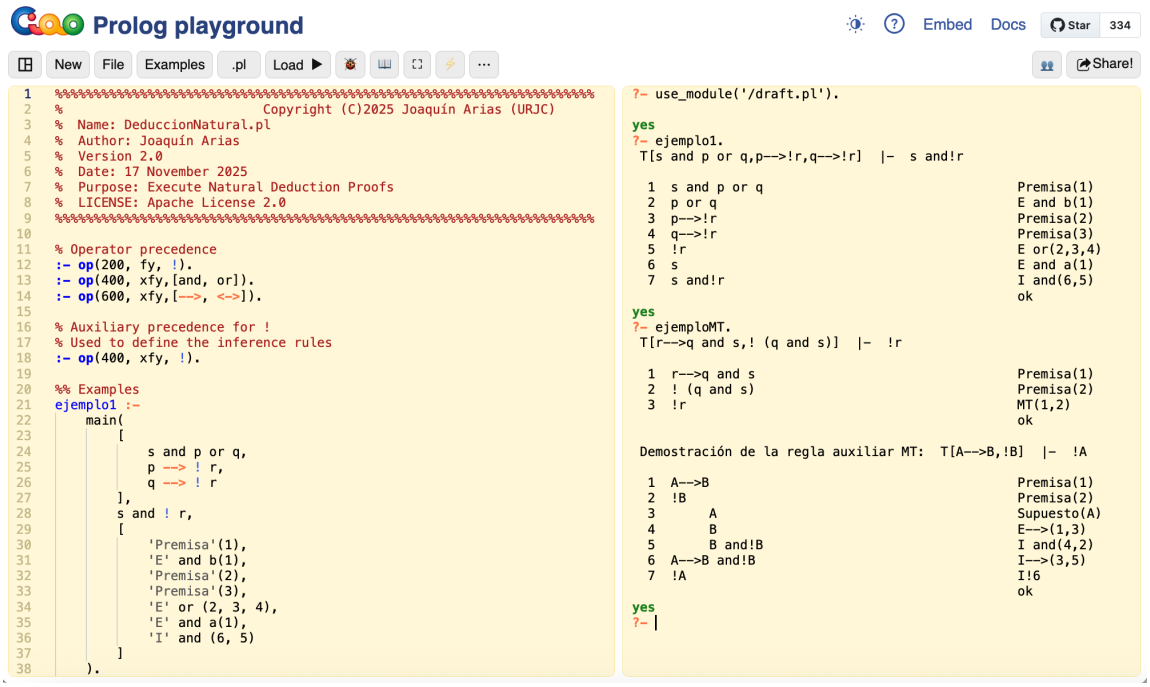


Figure 2: Screenshot with the execution of DeduccionNatural.pl.

$$T[\, s \wedge (p \vee q), p \to \neg r, q \to \neg r \,] \vdash s \wedge \neg r$$

1.	$s \wedge (p \vee q)$	$premisa$
2.	$p \vee q$	$E_{\wedge}(1)$
3.	$p \to \neg r$	$premisa$
4.	$q \to \neg r$	$premisa$
5.	$\neg r$	$E_{\vee}(2,3,4)$
6.	s	$E_{\wedge}(1)$
7.	$s \wedge \neg r$	$I_{\wedge}(5,6)$

(a) Proof presented in Example 2.

```
1  main( [ s and p or q,
2          p --> ! r,
3          q --> ! r ],
4        s and ! r,
5        [ 'Premisa'(1),     % 1
6          'E' and b(1),     % 2
7          'Premisa'(2),     % 3
8          'Premisa'(3),     % 4
9          'E' or (2,3,4),   % 5
10         'E' and a(1),     % 6
11         'I' and (6,5)     % 7
12       ] ).
```

(b) Proof using DeduccionNatural.pl.

Figure 3: Proofs using natural deduction for Example 1.

the third is the list of rules to apply. To incorporate a premise to the proof we have to indicate its index in the list of premises. For example, `'Premise'(1)` incorporates the first premise, `s and p or q`. As we mentioned before, the functors `a/1` and `b/1` select a concrete consequent, i.e., with `'E' and b(1)` we derive the formula on the right, `p or q`.
Note that, although in Example 2 we write $I_{\wedge}(5,6)$ (see line 7 in Fig. 3a), in DeduccionNatural.pl the user must provide the arguments in a specific order (see line 11 in Fig. 3b). That is using `'I' and (6,5)` it derives `s and !r`, whereas with `'I' and (5,6)` it would derive `!r and s`, which is not the desired formula.

The predicate `main/3` can be invoked directly in the Playground console (right panel) or through an auxiliary predicate defined as part of DeduccionNatural.pl. Figure 2 shows the implementation of `example1/0` as part of DeduccionNatural.pl and in the console shows its invocation and the generated proof (which matches the proof defined in Example 2).

To facilitate the learning of DeduccionNatural.pl we have created a manual and three videos, available, together with DeduccionNatural.pl at https://github.com/Xuaco/DeduccionNatural.

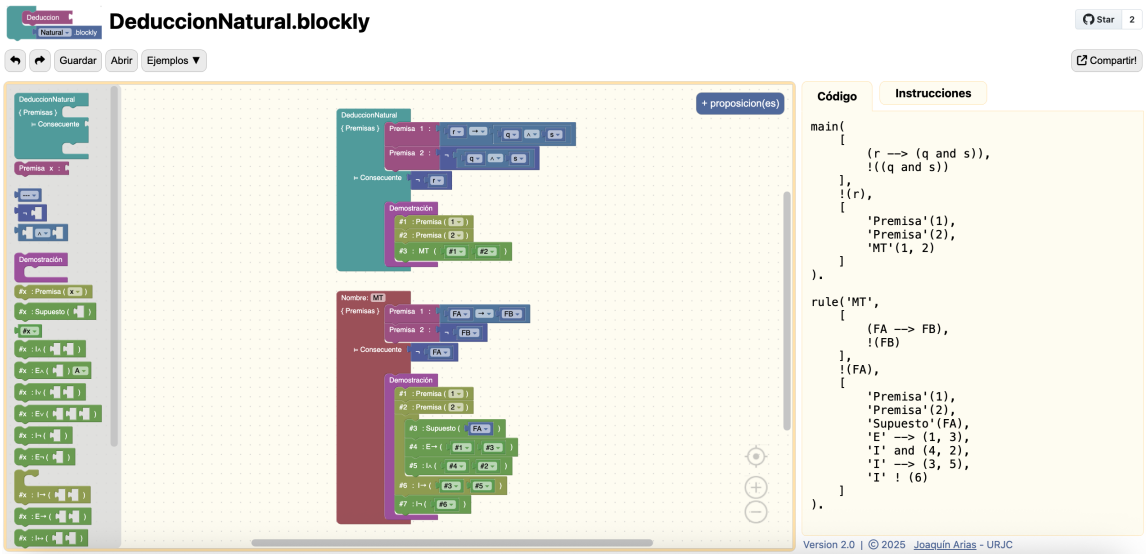


Figure 4: Screenshot with the execution of DeduccionNatural.blockly.

3.4. Implementation of derived rules

As mentioned above, one of the main advantages of DeduccionNatural.pl is the possibility of defining derived rules (for factoring proofs) using the predicate `rule/4`.

Example 4. Perhaps the best-known derived rule is *Modus Tollens*, $\{\varphi \to \psi, \neg\psi\} \vdash \neg\varphi$ whose implementation (which we include in DeduccionNatural.pl as an example) is as follows:

```
1  rule( 'MT',
2        [ FA --> FB,
3          !FB ],
4        !FA,
5        [ 'Premisa'(1),
6          'Premisa'(2),
7          'Supuesto'(FA),
8          'E' --> (1,3),
9          'I' and (4,2),
10         'I' --> (3,5),
11         'I' ! (6)
12       ]).
```

where the first argument defines the name of the derived rule, `'MT'`, the second one the list of premises, `[FA --> FB, !FB]`, the third the consequent of the derived rule, `!FA`, and the fourth the list of rules to apply. Note that we use meta-formulas, i.e., formulas that instead of proposition symbols have variables (words starting with capital letters) and that during the evaluation will be instantiated with the corresponding formulas.[a]

[a] If we implement other derived rules and these are not written following the rule *Modus Tollens*, the Ciao compiler will emit a `WARNING` but the program will work correctly.

To use a derived rule we can proceed as with a basic inference rule. In this case, the number of arguments and their order is determined by the number and order of the formulas in the list of premises defined in the implementation of the derived rule.

Example 5. Adapted from [1, pp. 51–52, Chapter 2], consider the deduction problem: $T[r \to q \wedge s, \neg(q \wedge s)] \vdash \neg r$ which using the derived rule `MT` (*ModusTollens*) introduced in Example 4, the proof can be written as part of DeduccionNatural.pl adding an auxiliary predicate, as:

```
1  ejemploMT :-
2      main( [ r --> q and s,
3              !(q and s) ],
4            !r,
5            [ 'Premisa'(1),
6              'Premisa'(2),
7              'MT'(1,2)
8            ]).
```

where the arguments we pass to `'MT'` match the premises defined in its implementation. Figure 2 shows the output obtained by invoking `?- ejemploMT`, the auxiliary predicate we have defined. Note that after the proof of the deduction problem, the correctness of the derived rules used are checked.

4. Design and implementation of DeduccionNatural.blockly

As we mentioned before, we have evaluated the use of DeduccionNatural.pl and concluded that, although the objective of facilitating the learning of natural deduction has been achieved, the design of the interface still needed to be improved [10].

The first action we implemented to improve the usability of DeduccionNatural.pl was the bachelor's thesis project of Marco, a student at the URJC, which involved developing a graphical interface that masked the Prolog syntax [9]. However, this approach did not satisfy us because we were losing the opportunity to introduce students to the use of Prolog.

Instead, we have developed a tool based on Google's Blockly library that facilitates the creation of natural deduction proofs using block-based construction in the style of Scratch (a tool developed by MIT for teaching programming to children), thereby avoiding syntax errors. Figure 4 shows the screenshot of the execution of DeductionNatural.blockly, displaying the proof from Example 5, in which a derived rule is used.

The interface is simple to facilitate usability, but it also provides undo/redo functionality, the ability to save and load files, and, more importantly, a collection of examples and exercises that allow users to explore the playground without needing to read the manual. Additionally, there is a share button that generates a link, which is automatically copied to the clipboard. When pasted into any browser, it opens DeduccionNatural.blockly with the proof exactly as it appears at that moment.

The implementation of DeduccionNatural.blockly automatically and on the fly generates, in the right-hand panel, the query for DeduccionNatural.pl. This panel also includes a second tab with instructions on how to copy the query into DeduccionNatural.pl. The generated query has the correct syntax thanks to, among others, the following features:

- Propositions are created/removed one at a time, without being restricted to predefined names.
- Premises are numbered automatically, and their numbering is updated whenever the user changes their order in the premises lists.
- Proof steps are also numbered automatically, with special attention given to blocks that combine assumptions with implication introduction.
- Each derived rule is automatically added to the block panel when the user defines it using a block structure that is almost identical to the one used for constructing the proofs.

5. Conclusions and future work

In this paper we have presented and described a preliminary version of DeduccionNatural.pl, a tool implemented in Prolog, for solving and correcting natural deduction proofs in propositional logic, and a visual interface to ensure syntax correctness of the proofs, called DeduccionNatural.blockly. With DeduccionNatural[.pl, .blockly] the proofs are programs, the inference rules are functions of a predefined library, and the derived rules are subroutines that allow refactoring the proofs.

An initial experience with students at the URJC has allowed us to (i) motivate some students to learn and apply Prolog in different use cases (indeed, some of them were able to detect a bug in the code of DeduccionNatural.pl), and (ii) identify aspects to improve. Overall, the results obtained encourage us to continue developing new functionalities. As future work we have already planned to extend the language to first-order logic, and we would be happy to collaborate in its adaptation to other notations, such as, for example, the already mentioned Huth/Ryan and Fitch notations. Additionally, we contemplate as future work the adaptation of DeduccionNatural[.pl, blockly] to automatically correct exercises and exams, as well as the implementation of a game in which each level is completed by solving a set of proofs that increase in complexity as the user progresses through the levels.

Acknowledgments

This work is supported by the projects Scaffolding Online University Learning: Support Systems

(2022-1-IT02-KA220-HED-000090206), funded by the European Commission (Erasmus+ 2022–2026), COSASS (PID2021-123673OB-C32), funded by MCIN/AEI/10.13039/501100011033, and EVASAI (PID2024-158227NB-C32), funded by MICIU/AEI/10.13039/501100011033 and by FEDER, EU, and the ERASMUS+ for international PhD internships.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] J. Arias, Lógica: desde Aristóteles hasta Prolog, Servicio de Publicaciones URJC, 2022. http://hdl.handle.net/10115/20014.

[2] J. E. Labra Gayo, ¿Hay Lógica en la situación actual de las titulaciones informáticas?, in: JENUI 2004, 2004, pp. 227–234.

[3] A. Gallinari, Apuntes y problemas de la lógica matemática, Dykinson, Madrid, 2009.

[4] J. Arias, A. Fernández, IA, Razonamiento Lógico y Lenguaje Natural, in: A. Hernando García-Cervigón, F. Martínez De Carnero, N. Candorcio Rodríguez, et al. (Eds.), Inteligencia Artificial, Tecnología y Discurso en la Educación Superior, Comares, 2025, pp. 13–31.

[5] Z. Fehér, L. Jaruska, K. Szarka, E. Tóthová Tarová, Students' propositional logic thinking in higher education from the perspective of disciplines, in: Frontiers in education, volume 8, Frontiers Media SA, 2023, p. 1247653. doi:10.3389/feduc.2023.1247653.

[6] M. Bouckaert, The assessment of students' creative and critical thinking skills in higher education across oecd countries: A review of policies and related practices, OECD Education Working Papers (2023). doi:10.1787/35dbd439-en.

[7] A. García-Carmona, Scientific thinking and critical thinking in science education: Two distinct but symbiotically related intellectual processes, Science & Education 34 (2025) 227–245. doi:10.1007/s11191-023-00460-5.

[8] G. Gentzen, Untersuchungen über das logische schließen. I., Mathematische Zeitschrift 35 (1935) 176–210.

[9] J. Arias, I. Ramírez, A. Gallinari, DeduccionNatural.pl: herramienta escrita en Prolog para el aprendizaje de la asignatura de Lógica, JENUI 2023 (2023) 129–136.

[10] J. Arias, Teach the importance of logic (programming) in Computer Science and why it is important, in: International Symposium on Computers in Education, SIIE 2023, Setúbal, Portugal, 2023, IEEE, 2023, pp. 1–6. doi:10.1109/SIIE59826.2023.10423714.

[11] F. Llorens Largo, R. Satorre Cuerda, F. Escolano, P. Arques Corrales, Deducción natural versus computación, JENUI 1999 (1999) 259–265.

[12] F. L. Largo, S. M. Cabrera, Herramienta para la enseñanza de la Deducción Natural, in: JENUI 2000, 2000, pp. 496–502.

[13] J. Humet, LSD, una herramienta didáctica para el aprendizaje de la lógica, JENUI 2001 (2001) 482–485.

[14] P. Manas, CLPractice 2.0. Tools for Learning: Computation and Logic, University of Edinburgh (2021). https://project-archive.inf.ed.ac.uk/ug4/20212198/ug4_proj.pdf.

[15] G. García-Pradales, J. F. Morales, M. V. Hermenegildo, J. Arias, M. Carro, An s(CASP) In-Browser Playground based on Ciao Prolog, in: ICLP Workshops, 3193, CEUR-WS, 2022.

[16] L. De Moura, N. Bjørner, Z3: An efficient SMT solver, in: TACAS, Springer, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3_24.