A Visual Improvement to the Pedagogy of Introductory Logic

Joshua Crotts*, Ali Altamimi, Harinderveer Badesha,
Christopher Brantley, Nadia Doudou

Department of Computer Science

Abstract:

Formal logic is considered by many philosophy majors to be challenging to overcome, exposing students to confusing symbols, rules, axioms, and other concepts that look similar to concepts from mathematics and computer science. We have seen students struggle with formal logic as a result of its conjectural and abstract nature, as well as the prolific and esoteric use of conceptual thought-processing. Figures or images (i.e., walking through the steps to solve a problem) frequently accompany textbooks and other sources when the need to demonstrate a problem or example arises. However, these often leave a lot to be desired - some concepts come easier than others, such as truth tables versus proof-based natural deduction, and it likewise depends on the person and their major/interests. Certain programming languages and websites exist that serve similar purposes but frequently do not provide user-friendly solutions to non-programmers and those that are not already experts at the material. Our current work is focused on building a visually appealing aid and tool to complement the traditional textbook and lecture pedagogy that provides beginner to intermediate students with a digital canvas to explore formal logic definitions, rules, and tools at their own pace in attempts to improve their overall understanding of the material.

Introduction:

The difficulty in learning formal logic comes primarily from the fact that many students simply have not seen the material before - the unfamiliarity is rather intimidating. Moreover, we have seen that, when students learn topics such as truth trees, truth tables, proofs, and other concepts related to first-order predicate and propositional logic, they want to understand the practicality of it. In addition, practice makes perfect when studying for these subjects, and having access to problems as well as solutions is often requested by the students. Of course, a professor/instructor can post these with step-by-step explanations, but even then, a student may be lost. We hope to solve the majority of these problems with our in-progress tool: LLAT (Logic-Learning Assistance Tool).

Tackling the issue of understanding the practicality is a challenge without directly exposing the students to the material, so that is what we have done. Within LLAT, there exist explanations for every concept and symbol that they can use, alongside axioms, definitions, symbol names, and formal names for that symbol. Further, to alleviate the fact that some logic courses use different symbols and notation, we give the ability to change the symbols to a set that they are familiar with. One issue with online solvers and programs comes from their restricting of the user to using one preset notation which most likely stems from either a “generalized” notation used in computer science and mathematics, or a hybrid of symbols from different styles. LLAT allows students to pick and choose which symbols they want from the various notations available.

Practice problems and exercises, alongside “live” explanations go a long way in improving the students’ overall comprehension of the material. This is why LLAT lets students insert any valid well-formed formula in propositional or first-order predicate logic, and the student can view different visual representations of said formula. Note that the validity of a well-formed formula varies from resource to resource, so we have standardized an approach used in the formal logic course offered at UNCG (PHI 310 - Introduction to Formal Logic). These explanations are offered in greater detail in the “Help” section of LLAT.

One final issue we wanted to address was translations. Sometimes, it may be difficult for a non-native English speaker to understand what this material means, so we allow them to translate the program into any alternative language.

Methods

LLAT offers three primary ways of viewing the results of a well-formed formula: parse trees, truth trees, and truth tables. After inputting a formula, the user may pick from several “algorithms” available depending on how many formulas are entered. Formulas are delimited by a comma, and the selection of available algorithms will vary based on what type of formula, and how many, are entered. Our algorithm choices come from what we believe are the most important things to know about a formula when learning formal logic. For example, identifying the main operator of a formula is crucial in being able to construct truth tables, truth trees, confirm the validity of an argument, and much more.

The following is a generalized list of all available algorithms to the user:

- Main operator detector
- Closed, open, and ground sentence determiners
- Truth table, truth tree and parse tree generators
- Closed and open tree determiners
- Logical relationship testers (equivalent, contradictory, consistent, contrary, implying, tautology, falsehood, and contingent)
- Argument validity tester
- Random propositional and predicate logic formula generators
- Export as .tex

Results:

Below are several screenshots of the program running with example inputs. We show different algorithms and features including a change of language, the login screen (to save preferences and last ten well-formed formulas used). The caption below each figure describes the algorithm used/task performed.

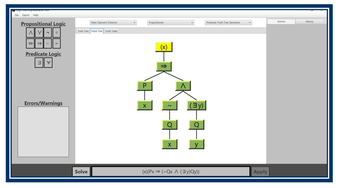

Figure 1: Parse tree of a predicate logic formula.

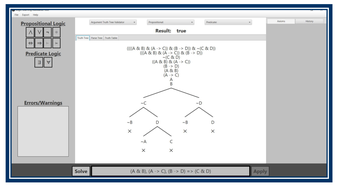

Figure 2: Determination of the validity of a propositional logic formula.

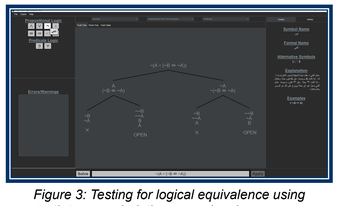

Figure 3: Testing for logical equivalence using truth trees, a dark theme, and an in-progress translation to Arabic.

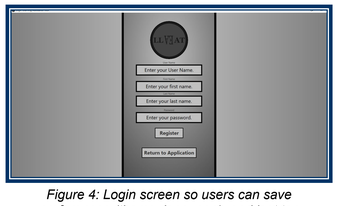

Figure 4: Login screen so users can save preferences (theme, language), and last ten wffs.

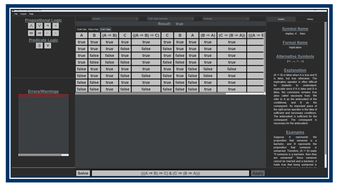

Figure 5: Truth table example, showing the implication axiom in right pane.

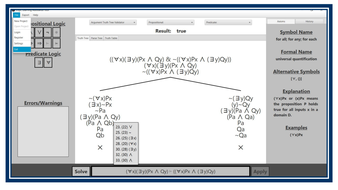

Figure 6: Argument validity check with result at the top in first-order predicate logic.

Discussion:

Many of the features we wanted to implement have incredibly challenging algorithms that do not have “fast’ solutions available; only approximations. To compensate, we prevent the user from entering too large of well-formed formulas. For instance, the complexity of generating a truth table is exponential, so after fourteen atoms, the program becomes too slow to run (also referred to as the boolean satisfiability problem in computational complexity). Likewise, while propositional logic is decidable, first-order predicate logic is only semi-decidable due to its use of the universal quantifier and the identity symbol. As a result, truth trees (and the steps within) may be reconstructed indefinitely. So, we enforce a timeout when generating truth trees - once a large-enough tree is entered, the program informs the user that the tree is too large to be computed.

Another challenge we encountered was the desire to feature as many languages as possible, so as to improve the user experience if they are a non-native English speaker. We utilized the Google Translate API, which only works so well.

Conclusions and Future Work:

Visual aids serve as an excellent alternative to a textbook or traditional lecturing, particularly for subjects which require much practice to perfect. In the future, we would like to add more algorithms, and improve on the ones we currently have. One example is to add support for natural deduction proofs as another method of determining the validity of an argument.

In addition, we would like to improve the performance of some algorithms, such as the truth table generator. Further, better warning and detailed error messages for the user could drastically improve their overall experience with the application. Also, a major part in designing this application was to add step-by-step explanations for many algorithms. While these are listed in detail in the “Help” menu, we plan to add detailed descriptions for each step in several algorithms including the truth tree generation, and argument validity proofs. Finally, because there is very little available research on the generation of random predicate and propositional logic formulas, our algorithm is subjective and arbitrary at best, and unsolvable/undecidable at worst. So, improving the generation capabilities by allowing the user to choose how complex they want the generated well-formed formula to be could serve them greatly.

In essence, LLAT is designed to improve the learning experience and process for students and teachers alike, and continued development and research on our part will refine the educational experience.

References:

[1] Hatcher, D. 1999. *Why Formal Logic is Essential for Critical Thinking.* Article in *Informal Logic*, Volume 12 (January 1999). 13 pages. DOI = 10.22329/il.v19i.2317
[2] Herman, G. L. et al., 2012. *Describing the what and why of students' difficulties in Boolean logic.* ACM Trans. Comput. Educ. 12, 1, Article 3 (March 2012), 28 pages. DOI = 10.1145/2133797.2133800
[3] Lawler, I. R., 2021. Introduction to Formal Logic Course Notes. Department of Philosophy. University of North Carolina at Greensboro.
[4] Priest, G. 2008. *An Introduction to Non-Classical Logic: From Ifs to Is*, 2nd ed. Cambridge: Cambridge University Press.

Acknowledgements:

We would like to acknowledge support through the UNCG Undergraduate Research, Scholarship and Creativity Office, as well as Dr. Minjeong Kim of the Computer Science department for mentoring this project.