Comparison of Natural Deduction Theorem Provers used in Electronic Tutoring Systems

L. JOSHUA CROTTS, Indiana University Bloomington, USA
STEPHEN R. TATE, University of North Carolina Greensboro, USA

In this paper we describe a methodology for comparing the efficacy of openly-available natural deduction tutoring software for students. Our proposed evaluation metrics are objective and can be applied to systems during development, before student-based assessments are feasible. We apply our assessment to five natural deduction tutoring systems, including two currently under development, and report the results of these evaluations and comparisons.

CCS Concepts: • **Theory of computation → Automated reasoning**; • **Applied computing → Computer-assisted instruction**.

Additional Key Words and Phrases: Natural deduction, formal logic, logic pedagogy, automatic theorem prover

ACM Reference Format:
L. Joshua Crotts and Stephen R. Tate. 2022. Comparison of Natural Deduction Theorem Provers used in Electronic Tutoring Systems. In *6th International Conference on Education and E-Learning (ICEEL), November 23–25, 2022, Tsuru, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The benefits of individually tailored tutoring as compared to group instruction are widely known, and have been the subject of many studies. The most influential research report along these lines is Bloom's 1984 paper [1] in which he provided evidence that a combination of individual tutoring and mastery learning led to improvements measured at two standard deviations from the mean. While the exact gain has been questioned, particularly the portion related to just one-on-one instruction [31], the impact of this paper led to the search for other instructional techniques that could match the resource-intensive one-on-one instruction gains, and this problem became known as "Bloom's two-sigma problem." Bloom's focus was primarily on instructor-led group instruction, but later advancements in both ubiquity and capability of technology gave rise to the use of cognitive automated tutoring systems as a way to address the two-sigma problem [3]. While cognitive tutors have been applied across a range of instructional levels, from introductory reading up through advanced mathematics, our work considers tutoring systems for natural deduction proofs in logic, which is central to disciplines from philosophy to mathematics to computer science, and which can be an important aspect of developing argumentation and critical thinking skills [13].

While a summative assessment of fully-developed tutoring systems must involve student testing, the design and development process requires the use of concrete measures that can be used to refine under-development systems into a final product. In natural deduction tutoring systems, a flexible system must be able to generate and use of novel problems rather than relying on limited hand-coded problem sets, and so tutoring systems must be able to not only guide students through proof construction, but the system must be able to derive proofs itself. Along these lines, we propose two metrics for developmental evaluation of natural deduction tutoring systems: the success rate in producing proofs for novel problems, and the simplicity of proofs produced. The latter metric is unique to tutoring systems: while a general "theorem prover" for technical problems simply needs to produce a sound proof, a tutoring system must provide something simple enough to be digested and understood by a student who is just learning about natural deduction. A proof engine that creates a 200-step proof for a simple exercise is not helpful in a tutoring system, even if it is a logically sound and correct proof. Our focus in this paper is to evaluate a collection of publicly available natural deduction tutoring systems along these lines, and compare to a system currently under development, and is the first such work applied to a range of logic tutoring systems. As the system development is guided by these metrics, it will lead to a future summative assessment of effectiveness with students. This paper is derived from the first author's masters thesis work [5].

2 LOGIC AND TUTORING SYSTEMS

Classical formal logic is a subset of philosophy that branches into related disciplines such as computer science, statistics, mathematics, and similar sciences. Logic is also taught in non-science fields like communication studies to reinforce critical thinking and improve deductive skills for argumentation. Per the Stanford Encyclopedia on Classical Logic [25], logic is a tool used for studying correct and analytical reasoning in both formal and informal languages. Its existence spawned questions ranging from its use in mathematics as an aid to disambiguate problems and proofs to considering it as an extension to natural language. Hatcher [13] posits that, due to an increased viewing of rhetoric and opinion versus factual knowledge in modern social media, the need for strong logical thinking abilities is crucial for evaluating, analyzing, and debating against arguments and claims. To address this need, Hatcher claims that standard logical deductive forms such as methods of inference and syllogisms serve as critical components for a student's ability to determine the validity of an argument and the relation (or lack thereof) of premises to conclusions. A desire for valid and sound arguments from students constitutes and contributes to wider adoption of formal logic classes in universities, or at the very least, the pedagogy of invalid arguments on how to refute incorrect and sometimes egregious contentions. Formal logic's relation to computer science, in particular, is prevalent when dealing with Boolean logic via sequential and iteration statements, mathematical proofs, set theory, reasoning about program correctness, and much more. The relevance of logic in our daily lives is unprecedented and often understated.

We first review some terms frequently used in formal logic-related work [26].

Definition 2.1 (Logic). A philosophical definition of *logic* is that it is a scheme of deductive reasoning defined by the laws of validity. In computer science, its definition extends this idea to electrical circuitry (i.e., how a computer ought to perform a given task).

Definition 2.2 (Well-Formed Formula). A *well-formed formula*, according to [23] is a string (formula) of syntactically-correct characters which conform (well-formed) to a language grammar.

Definition 2.3 (Proof). A *proof* is a style of argument which establishes the truth of a proposition.

Definition 2.4 (Natural Deduction). *Natural deduction* [11] is a form of proof where the method of reasoning stems from human, or natural, reasoning.

Extending formal logic pedagogy into educational technology is not a new idea — there exist many online solvers, provers, and programming languages designed to suit the needs of logic students, or those that use formal logic in some manner. In addition to these, we will also mention more powerful theorem provers that are aimed at experts/more experienced users.

Many of the systems we describe below, including our own, are a type of intelligent interactive tutor, which mimic the relationship between instructors and their student(s). Correspondingly, if a student starts to struggle with any given topic, the tutor ought to recognize the mistakes made, correct them, and then lead them down the intended path. Similarly, it should provide accurate and customized hints to a problem when requested, even if the student has not necessarily yet gone astray. Taking this a step further may call for dynamic generation of questions tailored to the individual student based on current progress. Intelligent interactive tutors have a host of benefits, including reduced stress on teachers by not having to create unique content for all students (a feat almost impossible unless the class size is appropriately small), and research has demonstrated that students perform better on assessments in one-on-one tutoring sessions [33] as demonstrated famously by Bloom [1].

Because of the reducible nature of propositional logic to simple structures and binary representations, there exist plentiful online truth table generators that provide detailed and immediate feedback for users while solving problems and well-formed formulas. Further, such generators work well not only for formal logic, but also computer science, mathematics, and electrical engineering, allowing students to enter a Boolean truth value (i.e., true/false) for an operand or proposition and the computer will determine if it is valid or invalid for an arbitrary cell [9]. An apparent drawback is that they require a student to have prior experience with the underlying logic or preexisting knowledge of entering values into a truth table [17], a sometimes undesired prerequisite. The problem is that many systems are not aimed at teaching, but rather serve as a solution or complementary aid to students or others who have a full understanding of the material.

Lukins et al. [19] developed the P-Logic Tutor: a propositional logic tutor with several key functions including a truth table generator, parse tree viewer, tautology/satisfiability determiners, a built-in theorem prover, all of which are supplemented with learning adaptability that generates feedback for students. Unfortunately, their provided link is now offline, meaning there is no way to investigate either the source code or even use the application in attempts to test it against modern alternatives.

Another stand-alone software-based solution (i.e., executable outside the browser) is LEGEND by Vlist [29]. LEGEND is untestable as it is closed-source and unavailable to the public, but it allows the user to prove and generate proofs from a (simple) given propositional formula.

Lodder et al. [18] created LOGAX, which is a tool for students to learn and construct axiomatic proofs with feedback and hints. While useful in its specific domain, we focus on natural deduction proofs instead of axiomatic proofs for simplicity and approachability for non-computer science students.

Mostafavi et al. [21] have written several papers on their Deep Thought tool with several iterations of improvement ranging from on-demand step hint generation to data-driven approaches to problem generation. Their work largely aims to improve the pedagogy of deductive logic via mastery learning leveling components which dynamically change based on student performance, with higher proficiency in problem solving leading to harder problems e.g., longer proofs, different axioms, etc. Their system also modifies the difficulty of problems to fit a student's mastery level (e.g., if a student has trouble answering a particular proof, they are given an easier alternative).

In [27], Siev describes the Automated Proof Search system AProS: a proof generation system which works in tandem with their proof tutor. Their system and pedagogical pipeline strays away from only propositional and first-order logic in favor of theory of computing topics such as Turing machines, computability, set theory, and others.

In [32], Verwer et al. briefly describe their propositional logic tutor Bop: a Fitch-style proving system. As an aside, Fitch-style calculus is a common method of proof for natural deduction, using lines to separate premises, derivations, and sub-derivations [10]. Like many other systems, their provided link is also offline, so we had no way of assessing its performance.

Cerna et al. [2] developed **AX**olotl: a unique tutor due to its Android implementation. **AX**olotl includes several types of proofs and tutorials, though its reliance on a file protocol to load problem sets or questions is a bit cumbersome for the non-savvy student or instructor as they describe. Plus, its curriculum is aimed at computer scientists with what appears to be a strong background in logic, lambda calculus, and type theory.

In [8] and [20], Mauco and Felice et al. show educational software for increasing the appeal of first-order logic to introductory students. With this idea, they developed FOLST and LogicChess. The former is a graphical application that shows an illustrated scene with predicates that define said scene, e.g., isSleeping(x), isOnTheGrass(x) when referring to entities on a farm. Students must enter facts that describe the illustration using these given predicates. LogicChess is similar in that students are provided predicates to describe a scene, but with the difference that it the scene is modeled as a chess board with chess pieces.

Perikos et al. in [22] developed a web-based system to help students translate sentences from natural language to first-order logic. Specifically, their approach is to ask students to find verbs, adjectives, and nouns from the text. Then, find connectives, predicates, and quantifiers. Finally, create groups of related predicates and repeatedly connect them until a final formula is achieved. They conclude with a suggestion to the broader research community that announces the problem of dynamically-generated hints.

Dostálová et al. [7] describe ORGANON: a web-based tutoring system for both propositional and predicate logic with randomly-generated mutations from a database of problems. Unfortunately, from the time that their paper was published, the system had very limited functionality, only supporting conjunctive/disjunctive/prenex normal form conversion exercises, with others in the works.

3 CHALLENGES FOR LOGIC TUTORING SYSTEMS

While plenty of research papers and projects on formal logic tutors and natural deduction proving systems exist, they are few and far between when viewed from the public, non-academic eye. That is, many only remain relevant in their academic research circle, and have either little purpose or minimal exposure outside to a "real audience." Further, current research efforts focus more on improving their current tool rather than performing direct comparisons with others. The issue with head-to-head comparisons is the mystery of a reliable metric: how do we measure "success" in a formal logic tutor without user evaluation? In other words, what metric is viable for determining the efficiency, or effectiveness, of a tutoring/proving/assistant system for formal logic? All students are not the same, and a formal logic class may contain students across different disciplines. As a result, what is difficult to one student who has yet to see logic may be easier to another student who programs or otherwise works with Boolean operators on a daily basis. Determining what exactly constitutes difficulty in a logic class is tricky for the same reason. In other words, it is a non-trivial job to evaluate whether one problem or topic is inherently more challenging than another.

Another associated problem of tutoring systems comes through quality and accurate problem generation. If a system is incapable of creating a natural deduction problem for the student to solve, then an expert, e.g., instructors, must manually write proofs and their solutions. The disadvantages of this approach are obvious: first, problems or propositions typically have multiple valid proofs, so if a logic system misinterprets a student's correct solution because it does not match the instructor's, it will certainly cause confusion. Secondly, while not as common as the former, an instructor may make a mistake when entering a solution or proof for the student to solve. Thus, when a student encounters this, it may take hours to fix the issue, assuming the student discovers the typo and does not question their sanity and intellect.

Finally, as we will discuss in sections 4 and 5, there is a significant burden placed on the evaluator of a logic system—such systems are rarely identical in terms of used syntax as there is no agreed-upon standard for logic. As an example, if an evaluator tests fifty formulas on five systems which each use a different logic language syntax, even if the difference is seemingly insignificant, it will require 250 variations in formulas. One way to minimize the cumbersome effort is to test two (or more) systems with identical input syntax. Another proposed method is to convert all formulas to an intermediate representation, then translate them into the appropriate language supported by the system (or outright use the intermediate representation). The former method reduces the testability of systems as a result of its rarity, whereas the latter requires adoption from the wider logic community. We develop and advocate for such a "gold standard" intermediate form in separate work [6].

4 STUDY DESIGN

Making comparisons between systems that perform related tasks appears easy at first glance, but quickly diverges into chaos if a consensus on differentiation is not reached. Our focus in this paper is on system evaluation of under-development systems, with a goal of leading to a well-designed system that can be assessed with students in the near future. We consider two systems under internal development (one desktop and one web-based), and compare against three freely-available outside systems. All systems are aimed at improving the pedagogy of introductory formal logic, and we briefly describe each system later in this section.

We first consider the methods used to investigate several natural deduction systems via head-to-head comparisons against one another. Systematically evaluating natural deduction software is complicated. For starters, as we previously mentioned, we need a metric of analysis: how to directly compare one system to another. The best and systematic way to do this, excluding the possibility of a human study, is via the overall size of a proof (i.e., the number of lines a generated proof contains). Beginning students that receive a large and unwieldy result are likely to ignore its significance out of frustration. Furthermore, because the length of a proof does not strictly correlate with its complexity, we consider the number of rules/axioms a system used to solve a proof compared to its base set size. The idea behind this decision is to provide insight into how often a system takes advantage of its axiom set, instead of trying to strictly use direct proofs (e.g., in Hilbert systems) or proofs by contradiction.

We collected and created 288[1] well-formed formula propositional and first-order logic proofs from various logic and computer science textbooks [14] [16] [30]. We manually converted each proof to the required syntax for each system, and then recorded the number of lines in the annotated proof as well as how many axioms/rules were used. Before we present the results, we will briefly review each investigated system.

4.1 FLAT: Formal Logic Aiding Tutor

Our first project, FLAT, began as a collaborative project which extended LLAT: the Logic Learning Assistance Tool [4]. This extension brought along a core component to formal logic proofs: the ability to prove or disprove a proposition via natural deduction. Not only does the system have an algorithm for automatically proving a clause of formulas, it also includes a tutoring system allowing students to, step-by-step, write a proof. FLAT supports both zeroth and first-order

[1] https://github.com/JoshuaCrotts/MastersThesis

Table 1. Subset of Implemented Algorithms in FLAT

Algorithm	Definition
Truth Tree	A truth tree is a description of the truth interpretations of a logic formula F.
Truth Table	A truth table is a sequence of true and false values evaluated for all models of a PL formula F.
Quine Tree Constructor	Quine's method [14] analyzes a PL formula F to determine if it is a tautology, contradiction, or contingency via heuristics to close branches.
Free Variable Detector	Finds all free variables in a FOPL formula F. An occurrence of a variable $v \in F$ is free iff there is no quantifier Q that binds v in its scope.
Bound Variable Detector	Finds all bound variables in a FOPL formula F. An occurrence of a variable $v \in F$ is bound if there is a quantifier Q that binds v in its scope.
Open Sentence Determiner	A FOPL formula F is open if $\exists v \in F$ such that v is free.
Closed Sentence Determiner	A FOPL formula F is closed if $\forall v \in F$, v is bound.
Ground Sentence Determiner	A FOPL formula F is ground if F does not contain any variables.
Main Operator Detector	A unary or binary connective c is the main operator of a logic formula F if it is the first-parsed operator when recursively evaluating F. If F contains no connectives, then there is no main operator.
Vacuous Quantifier Detector	A quantifier q in a FOPL formula F is vacuous if it does not bind any variable v in its scope.
Logical Tautology Determiner	A logic formula F is a logical tautology if it is true in every interpretation/model.
Logical Falsehood Determiner	A logic formula F is a logical falsehood if it is false in every interpretation/model.
Logical Contingency Determiner	A logic formula F is a logical contingency if it is neither a logical tautology or logical falsehood.
Logically Consistent Determiner	Two logic formulas F, F' are logically consistent if there a shared model $\mathcal{M}$ such that F and F' are true.
Logically Contradictory Determiner	Two logic formulas F, F' are logically contradictory if there is no model $\mathcal{M}$ such that $F_{\mathcal{M}} = F'_{\mathcal{M}}$.
Logically Contrary Determiner	Two logic formulas F, F' are logically contrary if there is at least one model $\mathcal{M}$ where F and F' are false and there is no model where both F and F' are true.
Logically Implied Determiner	Two logic formulas F, F' are logically implied if there does not exist a model $\mathcal{M}$ such that F is true and F' is false.
Logically Equivalent Determiner	Two logic formulas F, F' are logically equivalent if there does not exist a model $\mathcal{M}$ such that $F_{\mathcal{M}} \neq F'_{\mathcal{M}}$.

Table 2. Natural Deduction Axioms

Axiom Name	Definition
Modus Ponens (MP)	$(\phi \to \psi) \land \phi \therefore \psi$
Modus Tollens (MT)	$(\phi \to \psi) \land \neg\psi \therefore \neg\phi$
Hypothetical Syllogism (HS)	$(\phi \to \psi) \land (\psi \to \gamma) \therefore (\phi \to \gamma)$
Disjunctive Syllogism (DS)	$(\phi \lor \psi) \land \neg\phi \therefore \psi$
Disjunction Introduction (∨I)	$\phi \therefore (\phi \lor \psi)$
Conjunction Elimination (∧E)	$(\phi \land \psi) \therefore \phi$
Conjunction Introduction (∧I)	$\phi \land \psi \therefore (\phi \land \psi)$
Material Implication (MI)	$(\phi \to \psi) \leftrightarrow (\neg\phi \lor \psi)$
Biconditional Breakdown (BCB)	$(\phi \leftrightarrow \psi) \therefore (\phi \to \psi) \land (\psi \to \phi)$
Biconditional Introduction (BCI)	$(\phi \to \psi) \land (\psi \to \phi) \therefore (\phi \leftrightarrow \psi)$
Double Negation Introduction (DNI)	$\phi \therefore \neg\neg\phi$
Double Negation Elimination (DNE)	$\neg\neg\phi \therefore \phi$
Transposition (TP)	$(\phi \to \psi) \leftrightarrow (\neg\psi \to \neg\phi)$
Constructive Dilemma (CD)	$(\phi \lor \psi) \land (\phi \to \gamma) \land (\psi \to \omega) \therefore (\gamma \lor \omega)$
Destructive Dilemma (DD)	$(\neg\gamma \lor \neg\omega) \land (\phi \to \gamma) \land (\psi \to \omega) \therefore (\neg\phi \lor \neg\psi)$
DeMorgan's Law (DeM)	$\neg(\phi \land \psi) \leftrightarrow (\neg\phi \lor \neg\psi)$
	$\neg(\phi \lor \psi) \leftrightarrow (\neg\phi \land \neg\psi)$
	$\neg(\phi \leftrightarrow \psi) \leftrightarrow \neg((\phi \to \psi) \land (\psi \to \phi))$
	$\neg\exists\phi \leftrightarrow \forall\neg\phi$
	$\neg\forall\phi \leftrightarrow \exists\neg\phi$
Existential Elimination (∃E)	$\exists x Px \therefore P\alpha$ where α is not previously-used
Existential Introduction (∃I)	$P\alpha \therefore \exists x Px$
Universal Elimination (∀E)	$\forall x Px \therefore P\alpha$ where α is previously-used

logics, with heuristics to prevent infinite proofs that comes with the semi-decidability of first-order logic as proven by Turing [28].

Table 1 provides a list of a subset of the FLAT algorithms/subsystems that students can use to practice. Table 2 shows the axiom base set for natural deduction proofs in general, all of which are supported in FLAT.

4.2 ANDTaP: Automatic Natural Deduction Tutor and Prover

ANDTaP is a smaller, web-based version of FLAT's natural deduction implementation. It supports a subset of its proving capabilities, but a superset of the tutoring functionality. Some natural deduction rules, e.g., associativity and commutativity, that are not present in FLAT work as intended in ANDTaP. The choice to use a web-based client for ANDTaP rather than a desktop application was highly influenced by the desire to allow students to use it wherever they want instead of being restricted to a locally-installed (desktop) program.

4.3 TeachingLogic

The first external tool we analyzed was a web application by Laboratoire d'Informatique de Grenoble (LIGLAB) for proving propositional logic natural deduction formulas [12]. While their tool includes a few other argument verification tools, e.g., semantic tableaux solver for modal logic S4, a resolution prover for first-order logic, we focused only on its propositional logic proving ability. Users enter premises as a series of conjunctions followed by an implication to the conclusion. This syntax follows the standard proof idea which says if all premises are true, then the conclusion must be true (in other words, the premises logically imply the conclusion). Beginning students or those using an ever-so-slightly different notation may be frustrated to discover that they have to convert their entire input to this rigid standard to parse it correctly. Such restrictions mean that users must focus on formatting their input to what the system requires rather than what it outputs as a result. We did find that their prover solved every propositional logic proof in our suite, but we found that because the system has a small base set of theorems/axioms, almost every proof is a proof by contradiction, resulting in several nested proofs which can be hard to decipher.

4.4 NaturalDeduction

The second system we investigated is a Windows 10 application designed by Jukka Häkkinen named NaturalDeduction [15], and it includes both a proof generator and a proof checker. While it primarily focuses on modal logic (specifically, the modal logic system S5), it has a propositional logic prover because modal logic natural deduction semantics are a superset of those of propositional logic. Its interface is clean and very elegant to use. Likewise, its ability to prove both theorems and premise-conclusion style proofs is helpful. We also found its performance on par with, if not faster than, other similar software. However, a drawback of NaturalDeduction is that while it generates short and simple proofs for a subset of our test suite, for others the proofs were so unmanageably long and cumbersome that a student would, realistically, never look through them. In addition to this significant issue, we discovered that the system places an arbitrary limit on the length of a premise set and its corresponding conclusion. Along a similar vein, the system refuses any proof that contains more than seven propositions/atoms, even if the proof contains no connectives — only atoms (e.g., $A, B, C, D, E, F, G, H, \therefore H$). Lastly, the system automatically converts connectives to a recognizable format, e.g., > to $\rightarrow$, and upper-cases any entered propositions. It gets confused, though, if the user enters a symbol it does not recognize (e.g., & instead of $\wedge$), and erroneously replaces symbols that otherwise together represent one into two separate symbols, such as => into $\equiv\supset$. These restrictions do not entirely detract students from the application; however, they exemplify the types of downfalls that other systems do not have.

4.5 TAUT-Logic

TAUT-Logic is a web application designed by Ariel Roffé, and assisted by the Buenos Aires Logic Group [24]. Unlike the first two, TAUT-Logic supports first-order predicate logic, and is the only easy-to-use system of its kind that we found which is not out of commission or abandoned. In addition to its natural deduction toolset, it supports basic set theory, truth table generation, and model truth.

One awkward characteristic of this system is that its propositional logic application only supports lowercase letters for atoms. Additionally, similar to NaturalDeduction, TAUT-Logic supports a total of nine different atoms, ranging from o to w. Supporting only nine atoms may be enough for some problems/proofs, but this restriction seems rather arbitrary for natural deduction, whose problem complexity should not grow strictly in terms of the number of atoms. Another odd design choice is the preference of English phrases for connectives instead of symbols such as, for example, "implies" versus $\rightarrow$ or $\supset$. Because all other systems we tested only utilize symbols (with the exception of FLAT, as it supports both), the process of converting each formula and symbol to the appropriate format was tiresome. Lastly, for some reason, the biconditional connective is not supported, requiring students to either convert them to a conjunction of implications, or omit the proof altogether. We found that TAUT-Logic's performance is roughly on par in terms of speed, but fails on moderately complex propositional and first-order logic proofs. We also saw a general increase in the produced proof size compared to the other systems.

5 RESULTS AND DISCUSSION

Building on system overview and first-impressions in the previous section, in this section we present results of our quantitative testing of these systems.

5.1 System Comparison Results

Table 3 shows the tabulated results, including the average number of applied distinct rules compared to the base set size, the average length of all proofs, and the average success rate. Note: systems that did not support first-order logic contributed to a lower overall success rate. Consequently, we further subdivided the data into two groups: one with only propositional logic formulas ($n = 203$) in table 4, and another with only predicate logic formulas ($n = 85$) in table 5. Also note that, when a system cannot solve a proof, this, in turn, lowers the average number of steps and applied distinct rules as those count as zero towards the average. Thus, we created another table to show non-zero total averages in table 6.

Table 3. Data Analysis of Propositional and First-Order Logic Proofs

System	Avg. Success Rate	Avg. No. Steps	Avg. No. Distinct
FLAT	**84.72%**	**5.86**	**2.79**
TeachingLogic	70.49%	12.41	3.77
NaturalDeduction	56.60%	13.50	3.38
TAUT-Logic	73.96%	13.34	4.46

Table 4. Data Analysis of Propositional Logic Proofs

System	Avg. Success Rate	Avg. No. Steps	Avg. No. Distinct Rules
FLAT	**85.22%**	**5.94**	**2.67**
TeachingLogic	99.01%	17.48	5.30
NaturalDeduction	79.31%	19.03	4.73
TAUT-Logic	76.35%	15.59	4.71

Table 5. Data Analysis of Predicate Logic Proofs

System	Avg. Success Rate	Avg. No. Steps	Avg. No. Distinct Rules
FLAT	**83.53%**	**5.62**	**3.05**
TAUT-Logic	68.24%	7.46	3.76

Table 6. Non-Zero Data Analysis of All Proofs

System	Avg. No. Steps	Avg. No. Distinct Rules
FLAT	**6.92**	**3.30**
TeachingLogic	17.61	5.35
NaturalDeduction	23.85	5.96
TAUT-Logic	18.04	6.03

5.2 Analysis of Results

As the preceding tables demonstrate, FLAT has a higher percentage of generated proofs that are what we consider digestable for a beginning logic student. It is desirable for a natural deduction proof to be understandable, comprehensible, and condensed enough where details are not omitted, but rather are fleshed out to provide a clear path to the solution.

While TeachingLogic's system has a greater percentage of solved proofs than FLAT, NaturalDeduction, and TAUT-Logic, we found its user interface to be unintuitive, particularly when entering a formula with unrecognizable syntax according to their language grammar. This odd requirement strongly deters those who wish to use a natural deduction proving software. Instead, it forces users to focus more time modifying their input to fit the system specifications than that spent understanding the generated proof(s). Moreover, its limited axiom base set upper-bounds the kinds of produced proofs—resulting in several nested subproofs and proofs by contradiction. Importantly, the generated proofs are syntactically and semantically correct, but to someone with little prior experience in formal logic or mathematical proofs, it may come across as overwhelming.

A severe problem with NaturalDeduction, as we previously mentioned, is its unnecessary inconsistency — the generated proofs range from perfect (in that they match the intended expert solution) to out of control and incoherent. As an illustration, suppose we want to prove $P = \{(A \leftrightarrow B), (C \leftrightarrow B)\}$, $c = A \leftrightarrow C$. FLAT's solution, which we consider the optimal solution under our axiom set has twelve lines including the premises and conclusion. NaturalDeduction, on the contrary, has 247 lines. The absurd length we discovered is not exclusive to this proof; we found several other sets of premises and conclusions which caused the system to generate unjustifiably long and incomprehensible proofs. Furthermore, its input requirements are unreasonably restrictive: only allowing eight unique atoms and fourteen identical atoms combined via binary connectives e.g., $A \land A \land \ldots \land A$. We are unsure if these limitations are due to the proof generation system that NaturalDeduction uses, but because the instructions do not provide this information, we can, at best, only speculate. One additional problem we discovered is that NaturalDeduction omits premises that do not impact the final proof (i.e., premises that it deems are unnecessary), even if they were listed as premises in the original premise set P. We feel that this omission is problematic due to the relative importance of initial premises to an argument, and believe that it would lead to confusion and thoughts of "What happened to premise X?" from students.

Moreover, neither NaturalDeduction or TeachingLogic support first-order predicate logic. Oddly enough, NaturalDeduction, instead, supports S5 Modal Logic.

Lastly, TAUT-Logic was, as mentioned before, the only system that we tested which supports both zeroth and first-order logics. However, it too comes with its share of disadvantages. It does not support the biconditional operator, requiring instead to write it as a conjunction of implications. As each biconditional operator doubles the length of a formula in this translation, a formula using many biconditionals would be exponentially larger after being transformed into a formula that TAUT-Logic could process. Its performance is similar in that of FLAT, but we found it times out on both zeroth and first-order logic proofs for sometimes no discernible reason with an error stating that it simply could not solve the proof.

5.3 Problem of Testability

While each system that we evaluated has its share of advantages and disadvantages, the most profound problem was the inconsistency in entering proofs and well-formed formulas. Each system uses a slight variation of logic syntax which in turn means we had to manually convert each formula from the original format into four separate standards (FLAT, TeachingLogic, NaturalDeduction, TAUT-Logic), thus producing a total of 1,152 formulas. As a means of rectifying the problem, we propose the idea of a gold standard logic syntax, one that allows translations to an intermediary language for simplified parsing and evaluation of natural deduction proof systems. In other words, the goal is to create a language pipeline that allows for easy conversion between one language $\mathcal{L}$, the gold standard $M(\mathcal{L}, w)$, and another arbitrary language of the same class $\mathcal{L}'$. Symbolically, $\mathcal{L} \Leftrightarrow M(\mathcal{L}, w) \Leftrightarrow \mathcal{L}'$. While it is relatively unrealistic to expect current state-of-the-art systems to rewrite their input pipeline to only support the gold standard, this is not necessarily our goal. Instead, we aspire to have the gold standard be an intermediate representation for formulas which are fed, or piped, to and from testable systems. These intermediary representations may be stored in a collective database for the performance and automatic efficacy evaluation of logic tutoring systems. In separate work, we fully define a syntax for such a gold standard which is flexible enough to represent all systems we discuss here, as well as many others.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed metrics for evaluating interactive tutors for logic and natural deduction, which we applied in the comparison of five tutoring systems. Two of the five systems are being internally developed, and the proposed metrics expose some weakness in existing systems that can be corrected in the under-development systems. Our experiments used head-to-head comparisons of these natural deduction systems, using 288 examples that were hand-translated to work with the various systems under evaluation. In addition to quantitative results from our metrics, the evaluation included qualitative observations and discussion of how observed problems could be avoided in a new system. We believe that these results show an innovative method for evaluation and refinement of the developed tutoring systems, and as a result the under-development systems have improved as a result of this head-to-head comparison.

There is still a significant amount of work to be done on the intelligence of logic tutoring systems. We, in particular, would like to completely move FLAT to the browser so students do not have to download an application on their computer. This transition is in-progress via ANDTaP, but it has much room for improvement. A responsive and clean UI is all but mandatory in today's world of web and mobile applications, and while ANDTaP is clean, it does not scale well to the mobile platform (e.g., sizing issues, button clicks). Further, we wish to improve the tutoring system by supplementing the user with hints similar to other tutoring systems across different disciplines. Unlike many other systems which specialize in one area of logic, e.g., natural deduction, FLAT has several tools wherein each could use a pedagogical upgrade via hints, automatic (satisfiable) problem generation, and classroom integration. We plan to continuously update and upgrade both systems, as well as extend research into improved methods of teaching logic with special emphasis on non-computer science or mathematics students.

Finally, we wish to formally evaluate the efficacy of FLAT/ANDTaP on a collection of students across multiple disciplines. Student data can further push the effort towards finding an improved difficulty metric, as well as provide a means for determining whether the objective metrics proposed in this paper align with student impressions.

REFERENCES

[1] Benjamin Samuel Bloom. 1984. The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher* 13 (1984), 16 – 4.

[2] David M. Cerna, Rafael P.D. Kiesel, and Alexandra Dzhiganskaya. 2020. A Mobile Application for Self-Guided Study of Formal Reasoning. *Electronic Proceedings in Theoretical Computer Science* 313 (Feb 2020), 35–53. https://doi.org/10.4204/eptcs.313.3

[3] Albert Corbett. 2001. Cognitive Computer Tutors: Solving the Two-Sigma Problem. In *User Modeling 2001*. Springer, 137–147.

[4] Joshua Crotts, Ali Altamimi, Harinder Badesha Christopher Brantley, and Nadia Salou-Doudou. 2021. A Visual Improvement to the Pedagogy of Introductory Logic.

[5] Larry Joshua Crotts. 2022. *Construction and Evaluation of a Gold Standard Syntax for Formal Logic Formulas and Systems.* Master's thesis. University of North Carolina Greensboro, 1400 Spring Garden St., Greensboro, NC 27412.

[6] L. Joshua Crotts and Stephen R. Tate. 2022. Promoting a Common Testbed for Natural Deduction Tutoring Systems. (2022). In preparation.

[7] Ludmila Dostálová and Jaroslav Lang. 2007. ORGANON — The Web Tutor for Basic Logic Courses. *Logic Journal of the IGPL* 15, 4 (2007), 305–311. https://doi.org/10.1093/jigpal/jzm021

[8] Laura Felice and Maria Carmen Leonardi. 2017. Motivating Students through Educational Software Development.

[9] Bastion Fennell, Eysa Lee, and Thomas Kim. 2020. Truth Table Creator. https://www.cs.utexas.edu/~learnlogic/truthtables/ Accessed: 2021-07-04.

[10] Frederic B. Fitch. 1952. *Symbolic Logic: An Introduction.* New York, Ronald Press Co.

[11] Gerhard Gentzen. 1964. Investigations Into Logical Deduction. *American Philosophical Quarterly* 1, 4 (1964), 288–306.

[12] Grenoble Computer Science Laboratory. 2021. Natural Deduction. http://teachinglogic.liglab.fr/DN/

[13] Donald L. Hatcher. 1999. Why Formal Logic is Essential for Critical Thinking. *Informal Logic* 19 (1999).

[14] James L. Hein. 2002. *Discrete Structures, Logic, and Computability* (2nd ed.). Jones and Bartlett Publishers, Inc., USA.

[15] Jukka Häkkinen. 2017. NaturalDeduction. http://naturaldeduction.org/

[16] Rodger L. Jackson and Melanie L. McLeod. 2014. *The Logic of our Language: An Introduction to Symbolic Logic.* Broadview Press.

[17] Kenneth R. Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew Hockenberry. 2004. Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. *Lester J.C., Vicari R.M., Paraguaçu F. (eds) Intelligent Tutoring Systems* 3220 (2004). https://doi.org/10.1007/978-3-540-30139-4_16

[18] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. 2017. Generating Hints and Feedback for Hilbert-Style Axiomatic Proofs. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 387–392. https://doi.org/10.1145/3017680.3017736

[19] Stacy Lukins, Alan Levicki, and Jennifer Burg. 2002. A Tutorial Program for Propositional Logic with Human/Computer Interactive Learning. *SIGCSE Bull.* 34, 1 (February 2002), 381–385. https://doi.org/10.1145/563517.563490

[20] Virginia Mauco, Enzo Ferrante, and Laura Felice. 2014. Educational Software for First Order Semantics in Introductory Logic Courses. In *Information Systems Education Journal*, Vol. 12. 15–23.

[21] Behrooz Mostafavi and Tiffany Barnes. 2016. Evolution of an Intelligent Deductive Logic Tutor Using Data-Driven Elements. *International Journal of Artificial Intelligence in Education* 27 (04 2016). https://doi.org/10.1007/s40593-016-0112-1

[22] Isidoros Perikos, Foteini Grivokostopoulou, and Ioannis Hatzilygeroudis. 2011. Teaching Assistant Tools for NL to FOL Conversion.

[23] Anthony Ralston, Edwin D. Reilly, and David Hemmendinger. 2003. *Encyclopedia of Computer Science.* John Wiley and Sons Ltd., GBR.

[24] Ariel Roffé. n.d. Natural Deduction. https://www.taut-logic.com/

[25] Shapiro, Stewart, and Teresa Kouri Kissel. 2021. Classical Logic. *The Stanford Encyclopedia of Philosophy (Spring 2021)* (2021).

[26] Theodore Sider. 2010. *Logic for Philosophy.* Oxford, England: Oxford University Press.

[27] Wilfried Sieg. 2007. The AProS Project: Strategic Thinking and Computational Logic. *Logic Journal of the IGPL* 15, 4 (2007), 359–368. https://doi.org/10.1093/jigpal/jzm026

[28] Alan M. Turing. 1936. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* 2, 42 (1936), 230–265. http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf

[29] Christiaan van der Vlist. 2019. A Solver and Tutoring Tool for Logical Proofs in Natural Deduction. Bachelor's Thesis.

[30] William van Orman Quine. 1950. *Methods of Logic.* Harvard University Press.

[31] KURT VanLEHN. 2011. The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist* 46, 4 (Oct. 2011), 197–221.

[32] Sicco Verwer, Mathijs Weerdt, and Jonne Zutt. 2005. A Tutoring System to Practice Theorem Proving in Fitch.

[33] Beverly Park Woolf. 2008. *Building Intelligent Interactive Tutors: Student-Centered Strategies for Revolutionizing e-Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.