# A Solver and Tutoring Tool for Logical Proofs in Natural Deduction

Christiaan van der Vlist

Bachelor of Artificial Intelligence

Examinors:
Tomas Klos
Benjamin Rin

Utrecht University
The Netherlands
4th of April, 2019

7.5 ECTS

**Abstract**

This paper introduces LEGEND, an interactive tutoring system which provides formal proofs in natural deduction and allows users to construct their own proofs. I describe the way LEGEND can be used, and I explain the algorithm it utilizes to construct proofs or create graphs of paths from premises to a conclusion.

## Contents

# 1 Introduction

Formal logic is of considerable importance within several subfields of artificial intelligence. It is used for knowledge representation, as a tool for analyzing techniques, and sometimes as a tool for solving problems (Thomason, 2018). It makes sense, then, that logic is frequently considered an important part of education in AI. Part of this logic education focuses on constructing formal proofs. An analysis has shown that a large number of people taking a logic course drop out when the construction of formal proofs is introduced (Lodder et al., 2016). To construct formal, syntactic proofs, three systems are commonly taught: Hilbert calculus, sequential calculus, and natural deduction. Of these three, natural deduction may be the most popularly used. It is for this reason that I have set out to make an intelligent tutoring system, called LEGEND (Logic Education Guide for E-learning Natural Deduction) which, first and foremost, uses Fitch's natural deduction rules to generate a proof of a given conclusion from given set of premises. Furthermore, LEGEND allows the user to make a proof themselves, and can suggest hints when prompted by the user.

The rest of this introduction will 1) give an explanation of what intelligent tutoring systems are and why they are important, 2) give an overview of other intelligent tutoring systems that focus on formal proofs and algorithmic systems capable of providing proofs in using natural deduction, 3) and finally give a short introduction of LEGEND.

## 1.1 Intelligent Tutoring Systems

The aim is for LEGEND to be an intelligent tutoring system (ITS) for education in constructing formal proofs in proposition logic with natural deduction. ITS are defined by Steenbergen-Hu & Cooper (2013) as interactive computer-assisted learning environments which often incorporate cognitive learning theories. Meta-analyses show that ITS are as effective as, or slightly less effective than, human tutors but more effective than traditional classroom education (VanLehn, 2011, Steenbergen-Hu & Cooper, 2013, Ma et al., 2014). These analyses also found that integrating ITS into existing curricula boosts the effectiveness of the education. Additionally, while ITS may be less effective than human tutors, the fact that they are software means that they can be used at any time, wherever there is a computer available, regardless of whether a human tutor can be present or not.

More specific to the topic of constructing logical proofs is the research of Mostafavi and Barnes (2016). They found that the performance of students doing formal proof problems improves considerably when they receive individual instruction, even when this instruction is given by a computer system. They further show that the performance increases and the time it takes students to complete problems decreases when these problems are tailored to the individual abilities of the students. These effects are further augmented with the addition of on-demand hints.

Although the current iteration of LEGEND was not built with the idea of suggesting problems to students in mind, I think that an intelligent tutoring system which can provide hints may already be of value, especially considering the recent rise in popularity of artificial intelligence.

## 1.2 Similar Tools and Systems

The works that played the largest role in building LEGEND have been those of Lodder et al. (2016) and Bolotov et al. (2005). However, these are far from the only works that relate to the subject of electronic logic education tools. I mention a few others in this section to provide a scope of what is out there, but the list is by no means meant to be exhaustive.

A non-interactive system which can algorithmically provide natural deduction proofs in first-order logic is described by Bolotov et al. (2005). The proof searching approach of this algorithm is goal-directed in that it keeps a list of derived formulae (the proof list) and a list of goals (the goal list), which can be either be a single formula or a pair of contradictory formulae. The first items in the list of derived formulae are the premises, and the first goal is the desired conclusion. When a derivation is made, it is checked against the current goal (which is the latest goal to be added to the list). If the goal has been reached, the appropriate introduction rule is applied. The algorithm then terminates if the current goal was the initial goal, and otherwise continues searching for new derivations. If no new derivations can be made, the algorithm searches for new goals, and checks if they can be reached. If no more new goals can be found, the algorithm terminates having found no proof. The way in which the algorithm searches for derivations and goals is determined by the current goal and the type of formulae in the proof list. Finally, this algorithm is correct and terminates for any decidable input.

Another system is LOGAX, which is a tool with a similar aim to that of LEGEND's. LOGAX was made by Lodder et al. (2016), and is an interactive tutoring tool which can provide hints and feedback to students who are constructing a proof in Hilbert-style. LOGAX uses an edited version of the algorithm made by Bolotov et al. (2005) to build labeled directed acyclic multi graphs (DAM) which represent the proofs. One DAM can represent multiple ways a given conclusion can be proven from the same set of premises. While a student is making a proof and asks for hints, LOGAX uses the respective DAM to see which route toward the conclusion the student is farthest along at, and suggests the next step on that route.

Bop, a system introduced by Verwer et al., (2005), is a web-based proof editor for Fitch-style natural deduction. It allows students to construct their own proofs for a given problem similar to how LEGEND does this. Unlike an ITS, however, it is unable to provide hints when a student gets stuck.

Perkins (2007) introduces three dynamic strategy proof tutors for constructing proofs with natural deduction: the Explanation Tutor, the Walkthrough Tutor, and the Completion Tutor. These tutors are based on the Carnegie Proof Lab, a computer-based proof construction environment.

The Explanation Tutor shows students who are constructing a proof the four possible strategies they can follow to reach a certain goal formula with hyperlinks that can be clicked for further information. This information contains an explanation of how the strategy is employed and whether it is applicable or not, and if following it will lead to a proof.

The Walkthrough Tutor guides a student along the path of this proof in a step by step manner, along with examples and extra explanation for the more complex steps should the student ask for them. The downside of the Walkthrough Tutor is that it only allows for one path to be followed. If the student wishes to deviate from this, the Tutor can no longer offer assistance.

The Completion Tutor is most like LOGAX in that it can suggest a step to a student who is stuck based on the partial proof that they have already constructed. If there is no partial proof, the Completion Tutor will suggest the first step that the Walkthrough Tutor would have suggested.

The last tutoring system I introduce here is Deep Thought, created by Mostafavi and Barnes (2016). What sets Deep Thought apart from the other tutoring systems I have discussed is that it offers problems to students instead of allowing them to enter their own. Another major difference is that it uses data to reach its goal rather than relying on an algorithm which constructs proofs. This means that Deep Thought is not bound to the limitations of an algorithm and that it can learn new solutions to old problems, as well as new problems altogether. Another feature of Deep Thought is that it is able to search its database for problems suitable to the abilities of individual students after an evaluation period. On the other hand, Deep Thought can only tutor students in problems that it has in its database, and only in ways that it has seen before.

### 1.3 This Paper's Tool

The algorithm LEGEND uses is a form of bidirectional search. One direction goes from the premises down, making derivations using only elimination rules. This is similar to the algorithm of Bolotov et al. The other direction is goal-driven and works from the conclusion up. The conclusion is considered the initial goal, which can then be divided up into subgoals that prove the conclusion when they themselves are proven. Those subgoals can then be further divided up, and so on. The idea of searching for a proof using goals is inspired by both Bolotov et al. and Velleman (2006). When these derivations and goals meet, a possible path to a proof is found.

Of the tutoring systems mentioned in section 1.2, LEGEND resembles LOGAX the most. Both use variations of the algorithm of Bolotov et al. to construct proofs. The underlying idea for finding suitable hints, i.e. constructing a graph to represent the ways the conclusion can be reached, is also the same. Yet, there are also several differences. First and foremost, it does not use the same method of constructing proofs; LOGAX uses Hilbert calculus while LEGEND uses natural deduction.

The rest of this paper is organized as follows: Section 2 contains a short explanation on natural deduction. Section 3 explains the interface of LEGEND and how to use it, both for generating proofs and practicing with proofs, and section 4 takes a detailed look at the algorithm behind LEGEND and explains it in depth. Finally, section 5 discusses the strengths and weaknesses of LEGEND, as well as improvements and changes that can be made.

## 2 Natural Deduction

LEGEND uses Fitch-style natural deduction to construct and represent proofs, and user-made proofs also have to be in this style. In this section I give a short introduction of Fitch-style natural deduction and an explanation of how it works.

Natural deduction as a system was developed independently by Gentzen and Jaśkowski in the 1930s as a replacement for Hilbert calculus. It is both sound and complete (Bolotov et al., 2005). The rules of natural deduction are frequently said to be more intuitive than those of Hilbert calculus and sequential calculus, and therefore easier to learn. For LEGEND, I used Fitch-style natural deduction, which is a refinement of the style of natural deduction introduced by Jaśkowski. Fitch-style natural deduction was invented by Frederic Brenton Fitch and first introduced in his book *Symbolic Logic: An Introduction* (1952).

In practice, natural deduction is both a set of syntactic inference rules and a framework in which these rules can be applied to reach a desired conclusion. The rule set contains an introduction and elimination rule for the logical connectives ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$), as well as the REITERATE rule. Introduction rules can be used to derive new propositions by introducing a connective to one proposition or in between two propositions, depending on the rule. Elimination rules, on the other hand, derive a new proposition by removing one or two connectives, again depending on the rule. Finally, REITERATE allows one to restate a proposition that is already known. Each introduction and elimination rule also has certain preconditions that must be met before they can be applied (see figure 2).

The framework consists of a column of numbered lines. Each line itself further consists of a proposition and an annotation explaining which rule and which preconditions were used to derive this proposition. Furthermore, the framework is divided into hypothesis intervals with headers containing a proposition which is assumed to be true in the respective hypothesis interval. The premises make up the header of the first hypothesis interval, while the conclusion is always at the bottom of the framework. The following is an example of a natural deduction proof for modus tollens. The premise is $(P \rightarrow Q) \wedge \neg Q$, and the conclusion is $\neg P$.

|  |  |  |
|---|---|---|
| 1 | $(P \to Q) \land \neg Q$ | Hypothesis, 1 |
| 2 | $\quad P$ | Hypothesis, 2 |
| 3 | $\quad (P \to Q) \land \neg Q$ | R, 1 |
| 4 | $\quad P \to Q$ | $\land$E, 3 |
| 5 | $\quad Q$ | $\Rightarrow$E, 2, 4 |
| 6 | $\quad \neg Q$ | $\land$E, 3 |
| 7 | $\neg P$ | $\neg$I, 2, 5, 7 |

Figure 1: Annotations are on the right. R stands for REITERATE, E for elimination, and I for introduction.

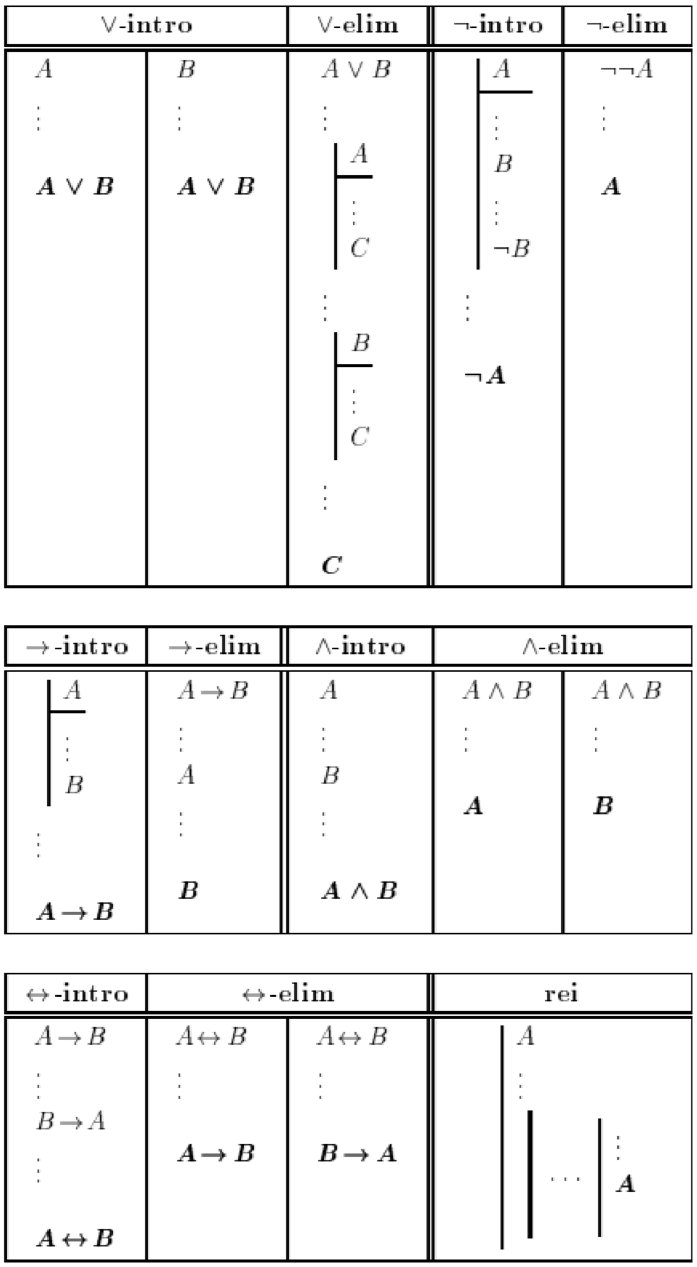


Figure 2: The introduction and elimination rules for each connective, as well as REITERATE, in Fitch-style natural deduction (Figure taken from Tonino, 2002).

## 3 LEGEND

For potential users, the interface of LEGEND is as important as the mechanisms behind it. Therefore, I have attempted to make the interface as straightforward as possible. I elaborate on how it works in this section.

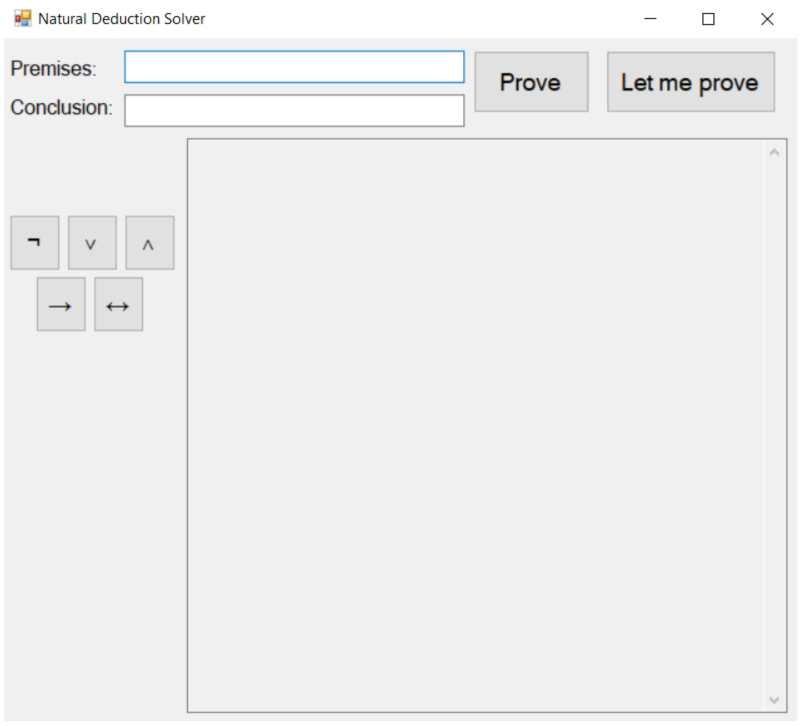
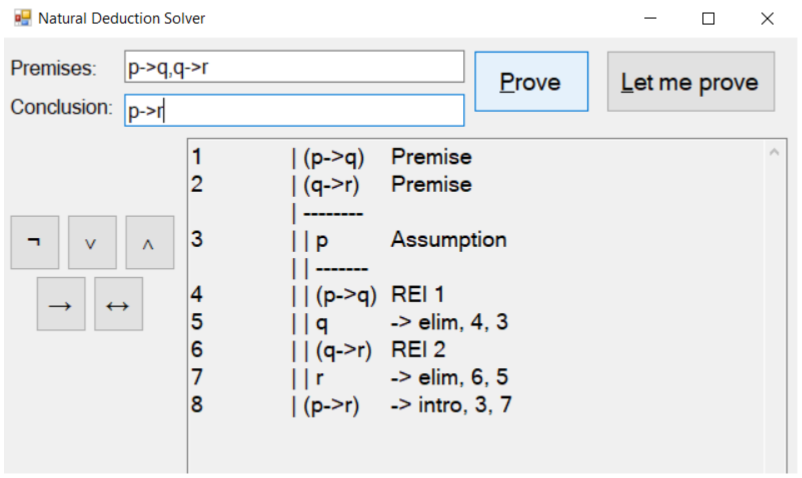


Figure 3: LEGEND immediately after running it.

Figure 3 visualizes what the interface looks like when LEGEND is ran. In the top left is the text box in which the premises, separated by commas, should be filled in. Directly below it is another text box for the conclusion. In order to aid in adding logical symbols to the formulae, the five buttons to the left of the interface can be used. Finally, in the top right, there are two buttons. The first one is "Prove", which, when pressed, prompts LEGEND to attempt to prove the conclusion from the premises and display the proof in the large box in the center of the interface (see fig. 4). The second button is "Let me prove". When this is pressed, the interface expands to allow the user to make their own proof (see fig. 5). In either case, a SAT solver is first ran to see whether the conclusion actually follows from the premises by determining if the the negation of the conclusion and the premises are unsatisfiable. This SAT solver was originally written by Julian Veltman for the bachelor AI course *Modelleren en Programmeren*, and modified by me to work with implications and bi-implications.

Note that LEGEND allows a user to construct their own proof even when it cannot find a proof itself. In such a case, however, no hints will be given.



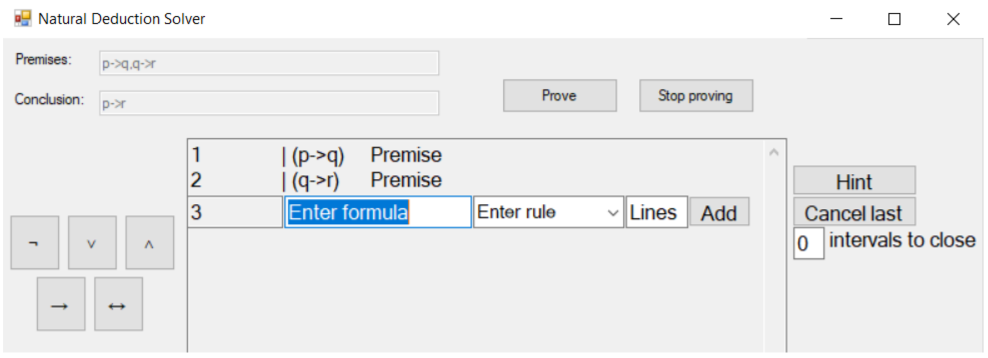
Premises: p->q,q->r

Conclusion: p->r

```
1    | (p->q)    Premise
2    | (q->r)    Premise
     | --------
3    | | p       Assumption
     | | -------
4    | | (p->q)  REI 1
5    | | q       -> elim, 4, 3
6    | | (q->r)  REI 2
7    | | r       -> elim, 6, 5
8    | (p->r)    -> intro, 3, 7
```

Figure 4: The proof is displayed in the box in the middle.

Using this expanded interface (see figure 5), the user can add formulae to the proof using the rules of natural deduction. A rule must be given each time a formula is added, and the lines which contain the formulae that justify the application of this rule must be specified as well (except when adding assumptions). When this is done, pressing the "Add" button adds the formula to the proof. If this formula is the conclusion, LEGEND notifies the user and stops asking for input. Otherwise, the user can continue adding new formulae to the proof in the same way. Furthermore, the "Hint" button on the right prompts LEGEND to generate a hint based on the proof that has been constructed thus far. However, since LEGEND is currently incapable of providing hints, this button is disabled. The "Cancel last" button below it removes the last formula added (but premises cannot be removed). Finally, the "n intervals to close" can be used to close a specified number of hypothesis intervals if so desired. Note that applying implication introduction, negation introduction, or disjunction elimination causes hypothesis intervals to be closed on their own if necessary. At any time, this process can be stopped by either pressing the “Stop proving” button, which has replaced the “Let me prove” button, or pressing the “Prove” button, which prompts LEGEND to write its own proof to the box.



Premises: p->q,q->r

Conclusion: p->r

```
1    | (p->q)   Premise
2    | (q->r)   Premise
3
```

0 intervals to close

Figure 5: The expanded interface after pressing “let me prove”.

## 4 The Algorithm

Whether LEGEND is trying to prove a desired conclusion from given premises or attempting to construct a graph depicting all the ways the same conclusion can be proved from those premises, the algorithm used is the same except for the stop condition. This algorithm is a form of bi-directional search. One direction consists of inferring formulae from the premises down with breadth-first search. The other direction comprises dividing the conclusion into subgoals, those subgoals into further subgoals, and so on, with iterative deepening search. Important to this algorithm are the terms of derivations, goals and subgoals, which I will explain first. After that, I will talk about the algorithm in depth.

### 4.1 Derivations, Goals and Subgoals

Derivations, like those in the algorithm of Bolotov et al. (2005), are propositions that are either a premise, or that have been inferred by applying an elimination rule on another derivation. Derivations also have a history detailing their parent derivations, which need to be done prior to the current one, and which rule was applied to derive it. Premises and assumptions have no parents. Furthermore, derivations have a length that is equal to the sum of the lengths of its parents plus one. Premises have a length of zero, while assumptions have a length equal to one plus the number of hypothesis intervals that are between theirs and the initial hypothesis interval.

LEGEND's algorithm also keeps a tree structure of goals. Goals are propositions that you want to know, because they will either prove the conclusion or part of it. The conclusion itself is the initial goal and serves as the root of the tree. A goal is proven when it matches a derivation.

Goals can be expanded into subgoals which are goals themselves that, when known, either fully or partly prove the parent goal. All goals have one "modus ponens" subgoal (MP subgoal) and a contradiction subgoal. Further possible subgoals depend on the main connective of the parent goal. See figure 6 for all subgoals. There are two additions to the subgoals introduced by Bolotov et al., inspired by the proof strategies of Velleman (2006). The first is that a disjunction goal $A \vee B$ has the subgoals $A, \neg B$ and $B, \neg A$ in addition to $A$ and $B$. The second is the addition of MP goals.

An MP subgoal consist of three parts, rather than just one as with all other goal types. The first is formula of the form $\mathcal{B} \rightarrow A$, where $A$ is the parent goal and $\mathcal{B}$ a meta-formula. All known implications that match this form comprise the second part. Lastly, the third part is made up of all the antecedents of the implications of the second part. These antecedent are the formulae which are actually considered during the search for a proof; the other two parts are disregarded.

The reason that $A \vee B$ has the additional subgoals of $A, \neg B$ and $B, \neg A$ is that $A$ and $B$ might not be provable by themselves, while $A \vee B$ is. For example, consider $\neg A \rightarrow B \vdash A \vee B$. Neither $A$ nor $B$ are provable, but it is true that either $A$ is true, or $\neg A$ is true and therefore $B$ is true. In either case $A \vee B$ is also true.

MP goals were added because problems of the form $A \wedge B \rightarrow P, A, B \vdash P$ and $A \leftrightarrow B \rightarrow P, A \rightarrow B, B \rightarrow A \vdash P$ could only be proven by an indirect proof otherwise, while a direct proof is just as viable a route to take.

## 4.2 Procedure

There are two objectives that LEGEND can use its algorithm for. The first is simply finding a proof from the premises to the conclusion. This is the objective when the "Prove" button is pressed (see Section 3). The second objective is finding all possible proofs without redundant steps from the premises to the conclusion and saving them as a graph, which can then be

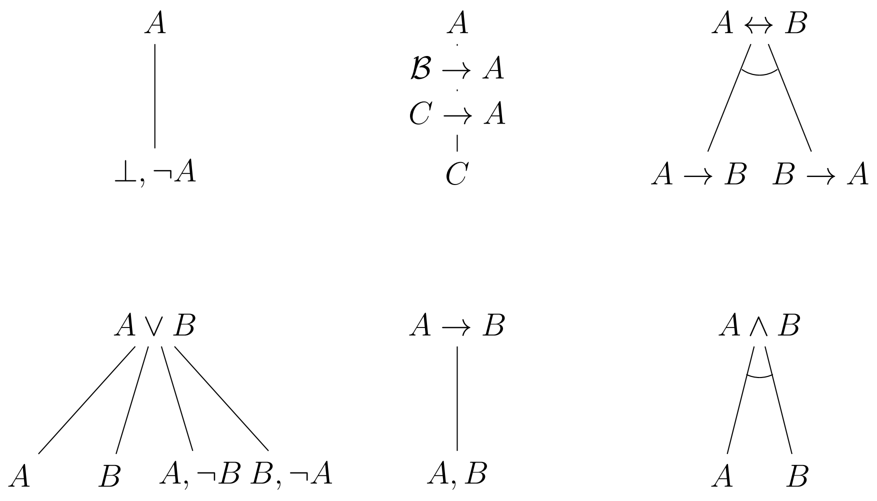


Figure 6: All subgoals. From right to left, top to bottom: 1) Contradiction goal, 2) MP goal, 3) Bi-implication subgoal, 4) Disjunction subgoal, 5) Implication subgoal, 6) Conjunction subgoal. Note that $A, B$ means $A$ under the assumption that $B$ is true.

used to provide hints when the user is making their own proof.

In either case, the algorithm starts by making all possible derivations from the premises by applying elimination rules in a breadth-first search manner. In case a derivation can be made in more than one way, the one with the lowest length is kept.

Next, the goal tree is created from the initial goal (i.e. the conclusion) by adding subgoals and then adding subgoals to those subgoals, etc. This tree is then searched with iterative deepening search. Assumptions are added whenever necessary, along with all derivations that can be made from the new assumption and the derivations that were already known. If the goal that is currently under consideration matches a derivation, it is considered proved. The derivation is then linked to the goal and the tree is checked backwards, starting from the current goal, to see which ancestors are now also proved. Neither proven goals nor their subgoals are checked in further iterations. On the other hand, if the current goal has no subgoals and cannot be proven, it and all its ancestors that require the current goal for their own proof are removed from the tree.

Contradiction goals are different from other goal types in that they cannot be directly matched to a derivation; they require that a contradiction be found instead. To do this, for all known negations $\neg A$, $A$ is set as a subgoal for the contradiction goal and is treated similarly to other subgoals with the exception that neither this subgoal nor its descendants can have contradiction goals as subgoals of their own.

Stop conditions of the algorithm depend on the objective. If the objective was to find a proof and the conclusion has been proved after some iteration, the algorithm stops. On the other hand, if the objective was to find all possible paths from the premises to the conclusion, the algorithm stops if the initial goal has only completed subgoals. Finally, regardless of the current objective, the algorithm stops if the initial goal has no subgoals left. Whether the algorithm was successful is determined by checking if the initial goal is proven or not; it was successful if and only if it is.

## 5 Discussion

I have introduced and explained the algorithm that LEGEND uses to make proofs and construct proof graphs for providing hints. LEGEND is able to constructs proofs for a large number of problems, but there are places for improvement.

The time and space complexity of LEGEND's algorithm have not been established. As it stands, however, it is likely that this complexity is needlessly increased by the serial manner in which the phases of derivation, expanding the goal tree, and searching the goal tree are carried out (see section 4, Procedures). If the steps taken during these phases were to be interleaved, I believe the complexity of the algorithm as a whole may decrease. The reason I think this is that, during the goal tree search phase, the complexities of the premises and conclusion do not matter beyond what is necessary to construct a proof. This is similar to the algorithm of Bolotov et al. For example, if some derivation $A$ is needed to prove a conclusion, the depth of the goal tree where the solution is found is the same whether $A$ is merely a proposition variable or a complex formula consisting of many variables and connectives. However, during the derivation and goal tree expansion phase, the complexities of the premises and conclusion do matter. Therefore, if these phases are carried out to completion first, unnecessary and avoidable steps are taken.

Furthermore, LEGEND's algorithm is not complete. An example of a proof that cannot be made is as follows: $(p \wedge q) \rightarrow (s \wedge t), p, q \vdash s$. This is a shortcoming of the way subgoals are made (see section 4 for reference). The goal $s$ only has a contradiction goal and an MP goal as subgoals. Both of these can be used to construct a proof[1], however, due to the way goals are handled, LEGEND does not consider the steps that must be taken to reach a proof from either of these goals. There are improvements that can be made to ameliorate these problems. For example, MP goals can be made better by checking if any of the possible derivations that can be made if the consequent were true would prove the parent goal, rather than merely trying to see if the parent goal and the consequent are the same.

A way that is certain to make LEGEND correct and complete is changing the approach it takes to proof searching from one that relies on goals and inference rules to one that relies on constructing a truth tree and translating that tree to a natural deduction frame. An example of this method is shown in *Logica*, chapter 7 (Tonino, 2002). The advantages of this approach are that constructing a truth tree and translating it to a frame are far less complex tasks, even when combined, than the approach LEGEND currently uses. The truth tree approach has also already been proven to be correct and complete. However, this approach has the drawback that it only finds one proof, and thus only one path, that could be used for tutoring. If a student were to deviate from this path, the proof would become useless and LEGEND would no longer be able to provide any hints. Since LEGEND is meant to be an ITS, I feel that this drawback outweighs the advantages of this approach.

Another possible point for improvement may be the iterative deepening search algorithm employed by LEGEND. One of the advantages of iterative deepening search is that it searches a space in the same way depth-first search does, thus saving memory usage, while guaranteeing that the first solution found is optimal. The disadvantage of iterative deepening search is that it costs more time than breadth-first search, but this overhead is only marginal if the branching factor is larger than one. At first glance this may seem to be the case because the majority of goals that LEGEND uses to search for a proof have multiple subgoals. When given a closer look, however, it becomes clear that most of these subgoals are significantly less deep than paths to actual solutions. Since unprovable goals are removed from the goal tree, the average branching factor for each level for every iteration is likely

[1]Contradiction goal: $\neg(s \wedge t)$ can be proved after assuming $\neg s$. Next, $\neg(p \wedge q)$ and $p \wedge q$ can be proved, which form a contradiction. MP goal: The only implication which can be considered for this goal type is $(p \wedge q) \rightarrow (s \wedge t)$. $s$ can be proved from $s \wedge t$, so $p \wedge q$ is a valid subgoal. This can easily be proved because both $p$ and $q$ are known.

less than two. This makes the overhead of iterative deepening search far costlier relative to the time it takes to find a solution. It might therefore be a better idea to instead use breadth-first search when the objective is to find a proof since it finds an optimal solution, and to use depth-first search when the objective is a graph of all paths from the premises to the conclusion, as this costs less memory. The branching factor and the maximum depth are guaranteed to be finite, so both algorithms are viable.

## 5.1 Future work

If LEGEND is to become an intelligent tutoring system, it needs to be able to make helpful suggestions to users who get stuck on a proof. As of right now, LEGEND and its algorithm are set up to create a graph of possible paths from the premises to the conclusion. The next step is adding functionality that allows it to provide sensible and helpful hints. In order to do this, it first has to check, for each path it found, how far along a user is in constructing a proof, and suggest taking the next step along one of those paths.

Furthermore, intelligent ways of deciding which step to suggest are necessary. It makes sense to suggest steps on the path that will lead to the conclusion the fastest. On the other hand, once a user has started progressing along a certain path, suggestions should instead aim to advance the user along that path, regardless of whether starting over from scratch might lead to a faster solution. Further research into what pedagogical theories exist is needed in order to be able to accurately determine which suggestion is the most helpful in which situation.

One issue that I can foresee arising—but I fear cannot be easily resolved—is what LEGEND should do when a student is following a path that will lead to a proof but that it does not recognize. Whereas humans can base the worth of steps that might or might not lead to a proof on a gut feeling, LEGEND only has the paths it has constructed as a reference. Thus, when students make steps that do not align with any path, LEGEND can only deem them as redundant. Whether this is desirable or not is up for debate, but I think that the way LEGEND currently handled unknown yet valid paths has more merit. Making a system that can predict the value of steps that do not align with any path would require a great amount of effort, and, like humans, it could still be wrong. On top of that, the algorithm LEGEND already has is already, in essence, such a system. The better solution is, I think, updating the existing algorithm to allow for non-included paths when they are discovered. This solves the problem just as well, and does not allow for error.

Whatever the future solution for the above will be, the algorithm still needs to be improved upon so that it can deliver proofs for problems that it currently cannot, potentially to the point where it is proven to be complete (though this will require a large amount of work still). Further work that can be done in this area is proving that the algorithm is correct.

## 6 Bibliography

Bolotov, A., Bocharov, A., Gorchakov, A., & Shangin, V. (2005). *Automated first order natural deduction.* Proceedings IICAI'05: the 2nd Indian International Conference on Artificial Intelligence, pages 1292–1311

Fitch, F. B. (1952). *Symbolic Logic: An Introduction.* Ronald Press Co.

Lodder, J., Heeren, B., & Jeuring, J. (December 2016). *Generating hints and feedback for Hilbert-style axiomatic proofs*

Ma, W., Adesope, O., Nesbit, J.C., & Liu, Q. (2014). *Intelligent tutoring systems and learning outcomes: a meta-analysis.* Journal of Educational Psychology, 106(4), pages 901–918.

Mostafavi, B., & Barnes, T. (2016). *Evolution of an intelligent deductive logic tutor using data-driven elements.* International Journal of Artificial Intelligence in Education, pages 1–32.

Perkins, D. (2007). *Strategic proof tutoring in logic.* Master's thesis, Carnegie Mellon.

Steenbergen-Hu, S., & Cooper, H. (2014). *A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning.* Journal of Educational Psychology, 106(2), pages 331-347.

Thomason, R. (Winter 2018). *Logic and Artificial Intelligence.* The Stanford Encyclopedia of Philosophy (Winter 2018 Edition), Edward N. Zalta (ed.).

Tonino, J. F. M. (July 2002). *Logica, Collegedictaat bij IN2 013 en IN2 310.*

URL: https://plato.stanford.edu/archives/win2018/entries/logic-ai/. Accessed on 12 March 2019.

VanLehn, K. (2011). *The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems.* Educational Psychologist, 46(4), pages 197–221.

Velleman, D. J. (2006). *How To Prove It, A Structured Approach, 2nd Edition.* Cambridge University Press.
Verwer, S., de Weerdt, M., & Zutt, J. (2005). *A tutoring system to practice theorem proving in Fitch.* 12th International Conference on Artificial Intelligence in Education, AIED '05, pages 33-37.