International Journal of Artificial Intelligence in Education
https://doi.org/10.1007/s40593-020-00222-2

ARTICLE

Generation and Use of Hints and Feedback in a Hilbert-Style Axiomatic Proof Tutor

Josje Lodder[1] · **Bastiaan Heeren**[1] · **Johan Jeuring**[1,2] · **Wendy Neijenhuis**[1]

Accepted: 27 September 2020Published online: 06 November 2020

Abstract

This paper describes LOGAX, an interactive tutoring tool that gives hints and feedback to a student who stepwise constructs a Hilbert-style axiomatic proof in propositional logic. LOGAX generates proofs to calculate hints and feedback. We compare these generated proofs with expert proofs and student solutions, and conclude that the quality of the generated proofs is comparable to that of expert proofs. LOGAX recognizes most steps that students take when constructing a proof. Even if a student diverges from the generated solution, LOGAX still provides hints, including next steps or reachable subgoals, and feedback. With a few improvements in the design of the set of buggy rules, LOGAX will cover about 80% of the mistakes made by students by buggy rules. The hints help students to complete the exercises.

Keywords Propositional logic · Axiomatic proofs · Hilbert axiom system · Feedback · Intelligent tutoring

✉ Josje Lodder
josje.lodder@ou.nl

Bastiaan Heeren
bastiaan.heeren@ou.nl

Johan Jeuring
J.T.Jeuring@uu.nl

Wendy Neijenhuis
wendy.neijenhuis@gmail.com

[1] Faculty of Science, Open University of the Netherlands, Heerlen, The Netherlands

[2] Department of Information and Computing Sciences, Universiteit Utrecht, Utrecht, The Netherlands

Introduction

The ACM 2013 computer science curriculum lists the ability to construct formal proofs as one of the learning outcomes of a basic logic course (Association for Computing Machinery (ACM) and IEEE Computer Society Joint Task Force on Computing Curricula 2013). The three main formal deductive systems are Hilbert systems, sequent calculus, and natural deduction. Natural deduction is probably the most popular system, but classical textbooks on mathematical logic usually also discuss Hilbert systems (Kelly 1997; Mendelson 2015; Enderton 2001). Hilbert systems belong to the necessary foundation to the introduction of logics (temporary, Hoare, unity, fixpoint, and description logic) used in teaching of various fields of computer science (Varga and Várterész 2006), and are treated in several textbooks on logic for computer science (Ben-Ari 2012; Nievergelt 2002; Arun 2002; van Benthem 2003). Hilbert systems are also taught in mathematics and logic programs (Leary and Kristiansen 2015; Goldrei 2005).

Students have problems with constructing formal proofs. An analysis of the high number of drop-outs in logic classes during a period of eight years shows that many students give up when formal proofs are introduced (Galafassi 2012; Galafassi et al. 2015). Our own experience also shows that students have difficulties with formal proofs. We analyzed the homework handed in by 65 students who participated in the course "Logic and Computer Science" during the academic years 2014-2015 and 2015-2016. From these students, 22 had to redo their homework exercise on axiomatic proofs. This is significantly higher than, for example, the number of students in the same group who had to redo the exercise on semantic tableaux: 5 out of 65.

A student practices axiomatic proofs by solving exercises. Since it is not always possible to have a human tutor available, an intelligent tutoring system (ITS) might be of help. There are several ITSs supporting exercises on natural deduction systems (Sieg 2007; Perkins 2007; Broda et al. 2006). In these ITSs, students construct proofs and get hints and feedback. We found two e-learning tools that can be used by a student to practice the construction of axiomatic proofs: Metamath Solitaire (Megill 2007) and Gateway to logic (Gottschall 2012). Both tools are proof-editors: a student chooses an applicable rule and the system applies this rule automatically. These systems provide no help on how to construct a proof.

In this paper we describe LOGAX, a new tool that helps students in constructing Hilbert-style axiomatic proofs. LOGAX provides feedback, hints at different levels, next steps, and complete solutions. LOGAX is part of a suite of tools assisting students in studying logic, such as a tool to practice rewriting formulae in disjunctive or conjunctive normal form, and to prove an equivalence using standard equivalences (Lodder et al. 2016; 2019).

LOGAX is an example of an ITS that gives hints and feedback to students solving tasks that can be solved in many different ways. Other domains with similar characteristics are proof systems such as natural deduction and the sequent calculus, but also proving geometry theorems (Matsuda and VanLehn 2004), and constructing a program that satisfies some given properties. Developing an ITS that gives hints for these domains is notably difficult, because not all possible solutions can be

calculated upfront or algorithmically, as for almost all kinds of tasks in, for example, algebra (Heeren and Jeuring 2014). If a student takes a step in the current solution space, LOGAX can provide feedback and hints. If a student takes a step outside the current solution space, LOGAX dynamically recalculates the solution space, taking the student step as a starting point. It then uses the new solution space as the source for hints and feedback. The dynamic approach and the algorithm to recalculate the solution space are central to our solution and make it possible to always give feedback and hints to a student. Similar techniques would be useful for ITSs for the other domains mentioned above.

The main contributions of this paper are:

- an example of a tutoring system giving feedback and hints for a domain for which feedback and hints cannot be specified algorithmically upfront
- an algorithm for generating axiomatic proofs and dynamically extending partial proofs
- an extension of this algorithm to incorporate lemmas
- generating hints and feedback based on this algorithm and studying the effect of these in small-scale experiments

To determine the quality of the proofs generated by LOGAX, we compare the proofs generated by the tool with expert proofs and student solutions. We use the set of homework exercises mentioned above to collect common mistakes, which we have added as buggy rules (rules to provide informative feedback) to LOGAX.

This paper is organized as follows. Section "Teaching Hilbert-Style Axiomatic Proofs" describes Hilbert's axiom system and the way it is introduced in textbooks and Section "An E-Learning Tool for Hilbert-Style Axiomatic Proofs" explains the interface of our e-learning tool LOGAX. Section "An Algorithm for Generating Proof Graphs" introduces the algorithm to generate proofs automatically. Section "Distilling Proofs for Students" explains how we linearize these generated proofs and Section "Lemmas" how we add the possibility to use lemmas. Section "Hints and Feedback" explains how we use the generated proofs for providing hints. This section also describes how we collect a set of buggy rules. Section "Evaluation of the Generated Proofs" and Section "Small-Scale Experiments with Students" discuss the results of several evaluations of our work. We relate our work to existing approaches of generating solutions and hints in Section "Related Work". Section "Conclusion and Future Work" concludes and presents ideas for future work.

Teaching Hilbert-Style Axiomatic Proofs

We start with a short description of Hilbert-style axiomatic proofs and the way they are introduced in different textbooks. Axiomatic proof systems come in several variants. The most common axiom systems are

$$\begin{array}{ll} \phi \to (\psi \to \phi) & \text{Axiom a} \\ (\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi)) & \text{Axiom b} \\ (\neg \phi \to \neg \psi) \to (\psi \to \phi) & \text{Axiom c} \end{array}$$

used for example in Ben-Ari (2012), Nievergelt (2002), van Benthem (2003), Goldrei (2005), and Kelly (1997), and the system consisting of Axiom a and b, but Axiom c' instead of Axiom c:

$$(\neg\phi \rightarrow \neg\psi) \rightarrow ((\neg\phi \rightarrow \psi) \rightarrow \phi) \qquad \text{Axiom c'}$$

used for example in Hirst and Hirst (2015), Arun (2002), Wasilewska (2018), and Mendelson (2015). These axioms are schemas that can be instantiated by replacing the metavariables ϕ, ψ and χ by concrete formulae. A proof consists of a list of statements of the form $\Sigma \vdash \phi$, where Σ is a set of formulae (assumptions) and ϕ is the formula that is derived from Σ. In a 'pure' axiomatic proof, each line is either an instantiation of an axiom, an assumption, or an application of the Modus Ponens (MP) rule:

$$\text{if } \Sigma \vdash \phi \text{ and } \Delta \vdash \phi \rightarrow \psi \text{ then } \Sigma \cup \Delta \vdash \psi$$

From these axioms and MP, the deduction theorem can be derived:

$$\text{if } \Sigma, \phi \vdash \psi \text{ then } \Sigma \vdash \phi \rightarrow \psi$$

The Open University of the Netherlands teaches axiomatic proofs in a bachelor course "Logic and computer science" and in a premaster program that prepares for admission to a master in computer science. The learning objective related to axiomatic proofs is:

– students are able to construct simple axiomatic proofs.

The course lectures start with recognizing instances of the axioms, and proceed with simple proofs, providing strategies such as:

– can you derive the last line of the proof by an application of the deduction theorem or Modus Ponens?
– how can you use the assumptions?

The textbooks we studied (Ben-Ari 2012; Nievergelt 2002; van Benthem 2003; Goldrei 2005; Hirst and Hirst 2015; Arun 2002; Wasilewska 2018; Mendelson 2015) do not give explicit learning goals, except for Kelly (1997), which starts each chapter with chapter aims. The aims of the chapter on axiomatic proofs are amongst others: "When you have completed your study of this chapter you should have a clear understanding of the structure of formal axiomatic systems, and be able to construct formal proofs of theorems". From the other textbooks we can deduce learning goals from the examples and exercises. The textbooks all start the chapter on axiomatic proofs with introducing the axioms, followed by some examples and exercises in which a student has to construct simple proofs or provide the motivation to given proof lines. Some of these proofs use earlier results such as lemmas or derived rules. Some books start with the first two axioms (Wasilewska 2018; Nievergelt 2002) and introduce the negation axiom (Axiom c or c') after the deduction theorem, others introduce the deduction theorem after the three axioms. After the introduction of the deduction theorem, exercises using this theorem are presented. The exercises in these textbooks suggest that constructing proofs is a learning goal. The single exception is Wasilewska (2018): here most exercises only ask to motivate steps in an already constructed proof.

Hardly any textbook provides substantial information about how to construct a proof, except from providing examples and showing the use of the deduction theorem. Wasilewska (2018) explicitly states that constructing a proof may start with searching for two statements such that the conclusion is an application of Modus Ponens on these statements, and Kelly (1997) explains how to use the deduction theorem and gives a heuristic to derive $\Sigma \vdash \psi \to \phi$ from $\Sigma \vdash \phi$. Constructing proofs requires knowledge of the syntax of propositional logic, and competencies in rewriting logical formulae. Therefore, most textbooks deal with rewriting formulas using standard equivalences (Goldrei 2005; Ben-Ari 2012; Wasilewska 2018; Arun 2002) or semantic tableaux (Kelly 1997; van Benthem 2003; Ben-Ari 2012), before the introduction of axiomatic proofs.

An E-Learning Tool for Hilbert-Style Axiomatic Proofs

The e-learning tool that we developed, LOGAX , uses the set of axioms a, b and c described in Section "Teaching Hilbert-Style Axiomatic Proofs" and Modus Ponens and the deduction theorem. A proof in this system can be constructed in two directions. To take a step in a proof, a student can ask two questions:

- How can I reach the conclusion?
- How can I use the assumptions?

An answer to the first question might be: use the deduction theorem to reach the conclusion. This answer creates a new goal to be reached, and adds a backward step to the proof. An answer to the second question might be: introduce an instance of an axiom that can be used together with an assumption in an application of Modus Ponens. This adds one or more forward steps. Figure 1 shows an example of a partial proof, constructed in our tool LOGAX. A full proof that completes this partial proof is:

1.	p	Assumption
2.	$p \to q$	Assumption
3.	$p, p \to q \vdash q$	Modus Ponens, 1, 2
4.	$q \to r$	Assumption
5.	$p, p \to q, q \to r \vdash r$	Modus Ponens, 3, 4
6.	$p \to q, q \to r \vdash p \to r$	Deduction 5
7.	$q \to r \vdash (p \to q) \to (p \to r)$	Deduction 6

Figure 1 illustrates most of the functionality of our e-learning tool LOGAX. A student starts with choosing a new exercise from the list, or formulating her own exercise. She continues working in the dialog box to add new proof lines. Here she can first choose which rule to apply: an assumption, axiom, an application of Modus Ponens or deduction theorem, or a new goal. In case of an assumption she enters a formula, and in case of an axiom, LOGAX asks for parameters to add the instantiation of the axiom to the proof. Figure 1 shows adding a Modus Ponens: a student has to fill in at least two of the three line numbers. LOGAX performs a step automatically and adds a forward or backward step to the proof. In the same way, a student provides a line number to perform a backward application of the deduction theorem. If the

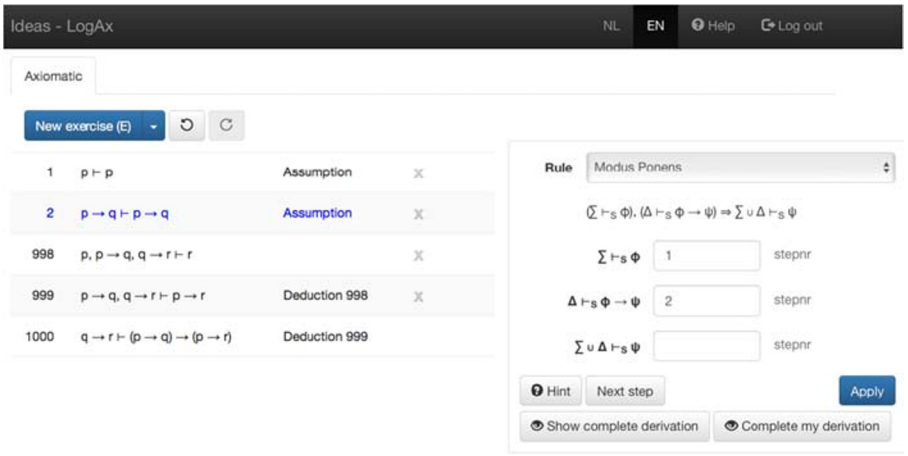


Fig. 1 A partial proof of $q \to r \vdash (p \to q) \to (p \to r)$ performed in LOGAX. On the right is the dialog box, in which a student can choose rules and fill in step numbers and help buttons below this dialog box. On the left is the proof as presented by LOGAX

deduction theorem is applied in a forward step, the student also provides a formula ϕ. The new goal option can be used to formulate a subgoal to be reached.

If a student makes a mistake, e.g. she writes a syntactical error in a formula, or tries to perform an impossible application of Modus Ponens, the tool provides immediate feedback. At any moment she can ask for a hint, next step, or a complete proof. The high number labelling the target statement (1000) is chosen deliberately, because at the start of the proof it is not yet clear how long the proof will be. After finishing the proof a student can ask the tool to renumber the complete proof.

The reason to use a dialog box in LOGAX to add new proof lines is that a student can concentrate on proof construction. The design choice to allow a student to choose a rule and let the software perform the rule has successfully been applied in several e-learning tools for logic and mathematics (Mostafavi and Barnes 2016; Beeson 1998; Robson et al. 2012). For instance, Robson et al. (2012) state that their interface "allows students to concentrate on strategies while the software carries out procedures". The use of the dialog box also implies that students can make fewer mistakes. By the time students start to develop proofs in LOGAX, they have had extensive training in writing syntactically correct formulae. Hence, LOGAX does not focus on writing correct formulae. However, a mistake with parentheses in a long formula, such as an instance of Axiom b, is easily made. By using the dialog box, students focus on proof construction and spend less time on correcting syntax errors. The only possible syntax errors students still can make occur in smaller formulae that need to be entered when adding, for example, an instance of an axiom to the proof. The evaluation in Section "Small-Scale Experiments with Students" shows that students make very few syntactical mistakes.

Our approach and design choices build upon scaffolding theories as described by Wood et al. (1976) and Belland (2017). Wood et al. (1976), referring to Bernshteĭn (1967), mention reducing the degrees of freedom as one of the scaffolding functions.

Reducing the number of steps that a student has to perform makes it possible to focus on the elements of the task that lead to learning gains (Belland 2017). The dialog box allows a student to concentrate on the steps that are closely related to the learning goal of LOGAX. 'Providing just the right amount of support' is the second scaffolding element in Belland's list. In an intelligent tutoring system scaffolding is often implemented as a sequence of hints that are increasingly supportive (Belland 2017). We have implemented this scaffolding strategy in our ITS. Since we do not employ a student model at this moment, we cannot apply fading strategies, which reduce the amount of feedback when the system thinks that the student does not need this. This is not necessarily a shortcoming: according to Belland (2017), leaving control of the support by the ITS to a student may lead to transfer of responsibility.

An Algorithm for Generating Proof Graphs

An ITS for axiomatic proofs provides hints and feedback. There are at least two ways to construct hints and feedback for a proof. First, they can be obtained from a complete proof. Such a proof can either be supplied by a teacher or an expert, or deduced from a set of student solutions. An example of an ITS for natural deduction proofs that uses student solutions has been developed by Mostafavi and Barnes (2016). A drawback of this approach is that the tool only recognizes solutions that are more or less equal to the stored proofs. The tool cannot provide hints when a student solution diverges from these stored proofs. Also, this only works for a fixed set of exercises. If a teacher wants to add a new exercise, she also has to provide solutions, and the tool cannot give hints for exercises that are defined by a student herself. The second way to provide the tool with solutions, which we use, is to create proofs automatically. At first sight this might only solve the second problem: automatically providing hints for new exercises. Section "Distilling Proofs for Students" explains how our approach makes it possible to provide hints also in case a student diverges from a model solution.

We develop an algorithm that automatically generates proofs. This algorithm should generate proofs that resemble textbook and expert proofs, which we can use to teach our students. Existing algorithms, such as the Kalmár constructive completeness proof (Kalmár 1935), or the algorithms used in automatic theorem proving (Harrison 2009), are unsuitable for this purpose, because the strategies used in these proofs differ too much from expert and textbook strategies. The Kalmár construction only provides proofs for tautologies. This is not necessarily a problem, since instead of, for example, proving $q \to r \vdash (p \to q) \to (p \to r)$, we can prove $\vdash (q \to r) \to ((p \to q) \to (p \to r))$. However, the Kalmár construction would start with eight proofs $(\neg)p, (\neg)q, (\neg)r \vdash (q \to r) \to ((p \to q) \to (p \to r))$ for each of the eight valuations of p, q and r, followed by a procedure to combine these eight proofs in a proof of $\vdash (q \to r) \to ((p \to q) \to (p \to r))$. The resulting proof will be considerably longer and more complicated than the proof we present in this article. The proofs generated by automatic theorem proving are also longer than, and different from, textbook or expert proofs. Natural deduction tools such as ProofLab (Sieg 2007) and Pandora (Broda et al. 2006) also use algorithms to

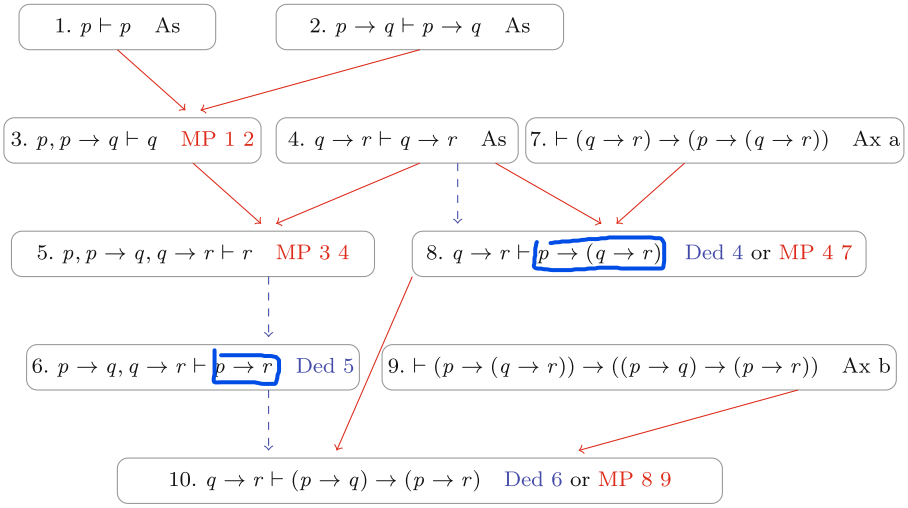


Fig. 2 A DAM for the proof of $q \to r \vdash (p \to q) \to (p \to r)$

calculate solutions, and these algorithms can provide useful hints and feedback. We adapt an existing algorithm for natural deduction to create axiomatic proofs. Before we describe the algorithm, we first explain how we represent proofs.

Figure 1 shows a partial example proof of $q \to r \vdash (p \to q) \to (p \to r)$. There are alternative ways to start this proof. A student may choose between various orders, for example swap line number 1 and line number 2. Using one or more axiom instances we may obtain entirely different proofs. Since we want to recognize different proofs, we represent proofs as labeled directed acyclic multi graphs (DAM), where the vertices are statements $\Sigma \vdash \phi$ and the edges connect dependent statements. We annotate vertices with the applied rule: Assumption, Axiom, Modus Ponens or Deduction. Note that a statement can be the result of different applications of rules. An example of such a DAM is shown in Fig. 2. Vertices are numbered for readability. A blue (dashed) arrow means that the lower statement follows from the higher by application of the deduction theorem. A pair of red (solid) arrows represents an application of Modus Ponens. This DAM contains three essentially different proofs: one that uses Axiom a and b, one that applies the deduction theorem and Axiom a, and one that uses no axioms and applies the deduction theorem twice. This last proof is a continuation of the proof provided in Fig. 1.

The basis for our algorithm for axiomatic proofs is Bolotov's algorithm for natural deduction proofs (Bolotov et al. 2005). The rules used in this system are presented in Fig. 3, restricted to the connectives $\neg$ and $\to$, since these are the only connectives

$$\neg\, elim \quad \frac{\neg\neg\phi}{\phi} \qquad \neg\, intro \quad \frac{\begin{array}{cc}\psi & \psi \\ \vdots & \vdots \\ \phi & \neg\phi\end{array}}{\neg\psi} \qquad \to elim \quad \frac{\phi \to \psi \quad \phi}{\psi} \qquad \to intro \quad \frac{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}{\phi \to \psi}$$

Fig. 3 Rules for natural deduction

used in the Hilbert axiomatic system. Here we use the same notation as presented in van Benthem (2003). A natural deduction proof is here presented as a tree-like structure. The elimination rules ($\neg_{elim}$ and $\rightarrow_{elim}$) express that you can extend a proof of $\neg\neg\phi$ with ϕ and combine subproofs of $\phi \rightarrow \psi$ and ϕ into a proof of ψ. The introduction rules discard assumptions: subproofs of ϕ and $\neg\phi$ can be combined in a proof of $\neg\psi$ by an application of rule $\neg_{intro}$ while discarding ψ. The last rule, $\rightarrow_{intro}$, states that if you have a proof of ψ, you can add $\phi \rightarrow \psi$ and discard ϕ.

The natural deduction rules for implication translate directly to rules in the Hilbert system: $\rightarrow_{elim}$ corresponds to Modus Ponens and $\rightarrow_{intro}$ to the deduction theorem. The rules for negation do not have direct counterparts in the axiomatic system. Therefore, the first adaptation that we have to make to Bolotov's algorithm is the use of axiomatic subproofs that mimic the natural deduction rules for negation. The $\neg_{elim}$ rule is translated to a single subproof, and we use seven different subproofs to translate the $\neg_{intro}$ rule, mainly to cover the possible different dependencies from ϕ and $\neg\phi$ on ψ.

The Bolotov algorithm is goal-driven, and uses a stack of goals. We build a DAM using steps that are divided into five groups. The first group contains a single step to initialize the algorithm. The steps in the second group check whether or not a goal is reached. The steps in the third group extend the DAM. The steps in group 4 handle the goals and may add new formulae to the DAM. In this group, a goal F can be added. The symbol F is not part of the language, but we use F as shorthand for "prove a contradiction". Finally, group 5 completes the algorithm, where we omit certain details for the steps that are needed to prevent the algorithm from looping.

1. We start the algorithm by adding the target statement (e.g. $q \rightarrow r \vdash (p \rightarrow q) \rightarrow (p \rightarrow r)$) to our stack of goals, and the assumptions of this goal ($q \rightarrow r \vdash q \rightarrow r$) to the DAM.

Until the stack of goals is empty, repeat:

2. (a) If the top of the stack of goals (the top goal from now on) belongs to the DAM, we remove this goal from the stack of goals.
   Motivation: the goal is reached.
   (b) If the top goal is $\Delta \vdash F$ and the DAM contains the statements $\Delta' \vdash \phi$ and $\Delta'' \vdash \neg\phi$ such that $\Delta' \cup \Delta'' \subseteq \Delta$, we add a set of axioms to the DAM that can be used to prove the goal below the top from these two statements. We remove the goal $\Delta \vdash F$ from the stack.
   Motivation: we can use the contradiction to prove the goal below the top. Apart from the instances of the axioms, this proof will use applications of Modus Ponens. Hence, the goal below the top will be removed in a later step.
3. (a) If the DAM contains a formula $\Delta \vdash \neg\neg\phi$, we add an instance of Axiom a ($\vdash \neg\neg\phi \rightarrow (\neg\neg\neg\neg\phi \rightarrow \neg\neg\phi)$) and two instances of Axiom c to the DAM. The next step uses these axioms to deduce $\Delta \vdash \phi$.
   Motivation: use the doubly negated formula.
   (b) We close the DAM under applications of Modus Ponens.
   Motivation: here we perform a broad search, and any derivable statement will be added to the DAM.

(c) If the DAM contains a formula $\Delta \vdash \psi$ and the top goal is $\Delta \setminus \phi \vdash \phi \rightarrow \psi$, we add $\Delta \setminus \phi \vdash \phi \rightarrow \psi$ to the DAM.
Motivation: use the deduction theorem.

4. (a) If the top goal is $\Delta \vdash \phi \rightarrow \psi$, we add $\phi \vdash \phi$ to the DAM and the goal $\Delta, \phi \vdash \psi$ to our stack of goals.
Motivation: prove $\Delta \vdash \phi \rightarrow \psi$ with the deduction theorem.
(b) If the goal is $\Delta \vdash \neg \phi$ we add $\phi \vdash \phi$ to the DAM and the goal $\Delta, \phi \vdash F$ to our stack of goals.
Motivation: prove $\Delta \vdash \neg \phi$ by contradiction.
(c) If the goal is $\Delta \vdash p$, where p is an atomic formula, we add $\neg p \vdash \neg p$ to the DAM and the goal $\Delta, \neg p \vdash F$ to our stack of goals.
Motivation: we cannot prove $\Delta \vdash p$ directly, and hence we prove it by contradiction.

5. (a) If the top goal is $\Delta \vdash F$ and $\Delta \vdash \phi \rightarrow \psi$ belongs to the DAM, we add $\Delta \vdash \phi$ to our stack of goals.
Motivation: we cannot prove a contradiction with the steps performed thus far. Hence, we exploit the statements we already have. Since our goal is to prove $\Delta \vdash F$, any formula is provable from Δ.
(b) If the top goal is $\Delta \vdash F$ and $\Delta \vdash \neg \phi$ belongs to the DAM we add $\Delta \vdash \phi$ to our stack of goals.
Motivation: use derived statements.

This algorithm constructs a basic DAM. Bolotov shows that his algorithm is sound and complete. Our adaptations, as for instance the replacement of a negation introduction rule by a set of instances of axioms, preserve soundness and completeness. We omit a detailed description of our adaptations and a proof of the correctness.

The above algorithm only uses axioms in a proof of a contradiction, or in the use of double negations. This means that without extra adaptations, Axiom b will never be used in a generated proof. Since we want the constructed proofs to resemble the proofs constructed by experts or students, and since LOGAX should teach our students to recognize the possibility to use axioms, we use extra heuristic rules to add more instances of axioms to the DAM. With these heuristics we can produce the example DAM in Fig. 2. The heuristics to produce the right branch with nodes 4, 7, 8, 9 and 10 of the DAM are:

- If the top goal equals $\Delta \vdash (\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi)$ and $\Delta' \vdash \psi \rightarrow \chi$ already belongs to the DAM and $\Delta' \subseteq \Delta$, then add an instance $(\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi))$ of Axiom b to the DAM.
- If the top goal equals $\Delta \vdash \phi$ and $\Delta' \vdash (\psi \rightarrow \chi) \rightarrow \phi$ and $\Delta'' \vdash \chi$ belong to the DAM with $\Delta' \subseteq \Delta$ and $\Delta'' \subseteq \Delta$, then add an instance $\chi \rightarrow (\psi \rightarrow \chi)$ of Axiom a to the DAM.

Distilling Proofs for Students

In the previous section we described the algorithm used to construct the DAM. Such a DAM may contain different solutions. For example, Fig. 2 shows three essentially

different solutions for the proof of $q \rightarrow r \vdash (p \rightarrow q) \rightarrow (p \rightarrow r)$. Since we use this DAM to generate proofs for the purpose of giving hints to students or providing sample solutions, we have to find a way to isolate single proofs. Moreover, the proofs in the DAM are structured as directed acyclic graphs, whereas an axiomatic proof is a linear structure. Hence, we need a procedure to extract linear proofs from a DAM. We will not only use this procedure to provide complete solutions, but also to generate next steps and hints, which means that the procedure should meet the following requirements:

- R1: generate a complete linear proof at once or stepwise
- R2: complete a partial proof, even if this proof diverges from the generated linear proof or contains a user-defined goal
- R3: add steps to a proof in an order that corresponds to the way students or experts add steps.

Requirement R1 is a direct consequence of our goal to use the DAM to provide sample solutions, hints, and next steps. Since a student solution may differ from the sample solution constructed from the DAM, we need requirement R2 to ensure that LOGAX can always provide a hint or a next step, using the procedure to complete a partial proof. There are two ways in which the order of the steps while constructing a proof may vary. To illustrate the first way, we look at the example proof in Section "An E-Learning Tool for Hilbert-Style Axiomatic Proofs". We could construct this proof in a forward way, from top to bottom, starting with line number 1 and finishing with line number 7. However, most textbooks advise to apply the deduction theorem backwards. Hence we prefer a solution that starts with line number 6 and 7. A second way in which the order of the steps may vary is the order of the lines in the completed proof. Take, for example, the proof in Section "An E-Learning Tool for Hilbert-Style Axiomatic Proofs" again. This proof might also start with line number 4.

To fulfil requirement R3, we studied example proofs in textbooks and student assignments. Most textbooks introduce an assumption or axiom only when it can be used directly. When an axiom or assumption is introduced in line n, it is used in an application of Modus Ponens or the deduction theorem in line $n + 1$, or in an application of Modus Ponens in line $n + 2$ combined with another component in line $n + 1$. We found one case in which an assumption in line n was followed by a proof of a second component of Modus Ponens, and an application of Modus Ponens. All but two of the textbooks described in Section "Teaching Hilbert-Style Axiomatic Proofs" use this proof order. Mendelson (2015) and Wasilewska (2018) form the only exceptions: they start a proof with stating all the necessary assumptions and axioms. In the homework assignments we also notice that students tend to introduce an assumption or axiom only when it is needed. This was confirmed by the pretest, described in Section "Small-Scale Experiments with Students", where all students who completed at least half of the axiomatic proof in propositional logic (16 out of 18 students) followed this strategy. In exams we also find sometimes student solutions starting with stating all the assumptions. If a student asks for a hint, we want LOGAX to provide the step that would be advised by an expert or a fellow student, hence R3 requires an order of the steps corresponding to the way students or experts add

steps. In the rest of this section we will first explain how we extract linear proofs and motivate why this way of extracting proofs matches the requirements later in this section.

The correctness of the algorithm defined in Section "An Algorithm for Generating Proof Graphs" ensures that the DAM contains a complete proof. Extracting a single proof can be seen as searching for a subtree. Linearization of this subtree requires topological sorting. Since generating a stepwise solution is one of the requirements, we perform these two tasks, extracting and linearization, simultaneously.

The procedure for proof extraction consists of four different kinds of steps, which are repeated until the linearized proof is complete. In each step a new line is added to the proof under construction, or an unmotivated line is motivated. In the following list, the different steps are ordered according to the preference in which a certain step is chosen:

- a close step: add a motivation to an unmotivated proof line
- a backward step: add a backward application of the deduction theorem to a proof
- a forward step: add a forward application of the deduction theorem or Modus Ponens to a proof
- an introduction step: add an assumption or axiom to a proof.

While performing these steps, the procedure keeps track of the partial linearized proof, which consists of the grounded part (the already proven proof lines), and the ungrounded part. The latter part consists of lines that are already in the linearized proof, but are either unmotivated, or their motivation depends on unmotivated proof lines. The unmotivated lines are part of a list of goals, which may also contain a set of other subgoals to be reached.

If possible, the procedure performs a close step, since in general this step completes the proof. The next preferred step is a backward step since a backward application of the deduction theorem replaces the goal to be reached by a simpler goal. When applications of the deduction theorem are impossible, the procedure tries to use already proven lines in an application of a forward step, and only if all other three kinds of steps are impossible, the procedure introduces an assumption or an axiom. Here we have to take care of the logical structure of the proof. We illustrate this by means of the example in Fig. 2. Suppose that the partial proof consists of nodes 6 and 10, which means that deduction was applied to node 10. Continuing with node number 7 would add a superfluous line to the proof. To prevent this, the procedure trims the DAM into a subDAM using the first goal of the list of subgoals as a root. In our example, node number 6 becomes the new root, and the leaves in this subDAM are the nodes 1, 2 and 4. Suppose the procedure continues with node number 1. That leaves two possibilities for the next step, namely node number 2 or 4. Because of the last requirement R3, node number 2 is preferred, since in general students or experts choose an assumption that can be used directly over an assumption that can only be used later. The procedure realizes this preference by adding subgoals to the list of subgoals after performing an introduction step. These subgoals consist of the nodes between the introduced leaf and the node that corresponds to a subgoal already in the list of subgoals. In our example, the line numbers 3 and 5 are added as subgoals, which forces the procedure to look for a next leaf in the subtree rooted by

line number 3 in the next step. As a consequence, line number 2 is indeed added in this step.

We claim that this procedure meets the three requirements given above. Requirement R1, generating complete proofs, is guaranteed by the construction of the DAM. To show that requirement R2 is met, we distinguish two situations. As long as the steps in the student solution correspond to the steps generated by the procedure described in this section, this proof can be completed directly. If the student solution diverges from the generated solution, LOGAX will use the student solution as a starting point to build a new DAM. Motivated lines will be marked as grounded lines and unmotivated lines will be part of the list of goals. This ensures that the procedure indeed extends the partial proof into a complete proof. Note that this complete proof may contain superfluous lines, for example when the student introduced an assumption that cannot be discarded by an application of the deduction theorem. This will result in a path in the DAM that is not connected to the goal. In such a case, LOGAX will not remove the student lines, but complete the proof using the student lines that lead to the goal.

From student solutions to exercises and the log data collected from LOGAX we know that students can perform the steps of a proof in many different orders. However, there are some heuristics in the construction of a proof, such as trying to use assumptions, or simplifying the goal by applying the deduction theorem. The preference on the order of the steps in our procedure ensures that the procedure follows these heuristics (requirement R3). This implies that steps can be added in two directions, forward and backward, and that a user can switch direction at any moment. Moreover, the order of the steps should be such that we can always motivate the next line: why do we perform a certain step at a certain moment. To achieve this, we use a dynamic programming approach, where subproblems are defined by the list of subgoals. The restriction to subDAMs as described above, ensures that we complete the subproblem defined by the first node of this list before we start a new subproblem. All steps can thus be motivated by a subgoal.

Lemmas

Reusing proven results is common practice in mathematics and logic. For example, the proof of the fundamental theorem of arithmetic (every number larger than 1 can be written in a unique way as a product of primes) uses the lemma that a prime divisor of a product $a \cdot b$ is also a divisor of a or b. In logic the use of proven results is widespread too. Here, proven results are sometimes presented as derived rules, such as for instance the rule Modes Tollens ($\neg \phi$ can be derived from $\phi \rightarrow \psi$ and $\neg \psi$) in Huth and Ryan's textbook (2004). Axiomatic proofs often build on each other: for example, a proof of $\vdash \neg\neg p \rightarrow p$ can be used as a lemma in another proof.

Lemmas appear in several ITSs that deal with constructing proofs. They serve various purposes, such as a starting set to generate geometry problems (Alvin et al. 2014), or just as a predefined set that can be used by the student to solve a problem (Matsuda and VanLehn 2005). Perhaps more interesting is the possibility to allow the addition of lemmas by the user. In both the Jape natural deduction proof assistant (Bornat

2017) and the proof assistant described by Aguilera et al. (2000), a student can save proven results and use these results as lemmas in a new proof. The proof assistant Gateway to Logic for axiomatic proofs offers a user the possibility to state and use lemmas too (Gottschall 2012).

Lemmas in axiomatic proofs have various shapes. For example, we distinguish tautologies ($\vdash \phi$) and valid sequents ($\phi_1, ...\phi_n \vdash \psi$), but also schemas (for example $\neg\neg\, \phi \vdash \phi$) and instantiations of schemas ($\neg\neg\, p \vdash p$). We include the use of lemmas in LOGAX. The main purpose of including lemmas is to support adding relatively easy exercises. Without lemmas, many axiomatic proofs are too lengthy and complicated to be used in education. With the possibility to use lemmas, a new class of relatively easy exercises becomes available. The second goal is to give users the possibility to use their own lemmas. LOGAX can provide a student with an exercise together with a lemma that may be used in the proof, and in a user-defined exercise the user can use her own lemmas. We impose two restrictions: predefined exercises use only instantiations of lemmas, and the interface only accepts user-defined lemmas that are instantiations of a tautology. The latter is not a real restriction since a valid sequent $\phi_1, ...\phi_n \vdash \psi$ can always be rewritten as a tautology $\vdash (\phi_1 \rightarrow (... \rightarrow (\phi_n \rightarrow \psi)))$.

We adapt the algorithm to create a DAM to support the use of lemmas. In predefined exercises, lemmas are added to the DAM at the start of the algorithm, comparable to the addition of assumptions. The construction of the DAM and the extraction of a linear proof work in the same way as described in Section "An Algorithm for Generating Proof Graphs" and "Distilling Proofs for Students". Students who solve these predefined exercises receive the lemma as a first line of the proof. To facilitate user-defined lemmas, a lemma rule is added to the set of rules. In a user-defined exercise, a student can introduce a lemma at any stage during the proof. The algorithm constructs a DAM based on the partial proof including the lemma and uses this DAM to provide hints and feedback.

Figure 4 shows an example of an exercise with a lemma. Since a motivation of line 999 as an application of Modus Ponens to the lemma in line 1 and line 4 completes the proof, the hint tells the student to add a motivation.

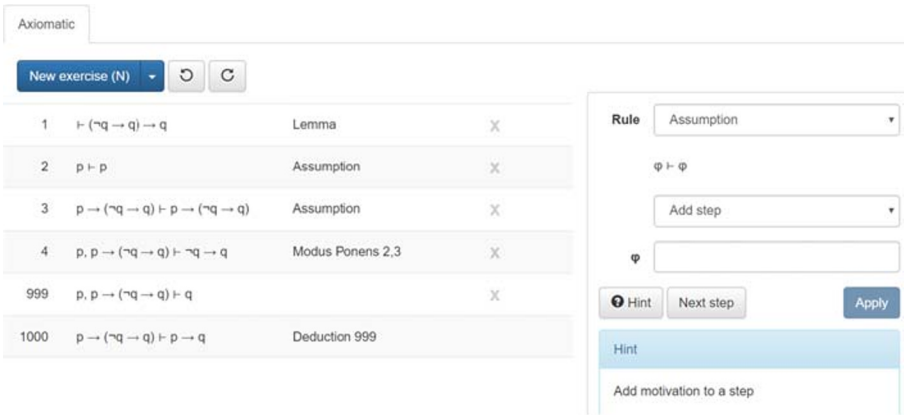


Fig. 4 A partial proof with a lemma, performed in LOGAX

Hints and Feedback

Hints

One of the reasons for the effectiveness of human tutors is that they provide feedback at the level of solution steps, and help a student to overcome impasses using hints (Merrill et al. 1992). Hence, for ITSs to be as effective as human tutors, they should give stepwise feedback and some form of help. In a first version of LOGAX, we implemented a hint sequence consisting of three hint types: the direction of a next step (forward or backward), the axiom or rule to apply, and a bottom-out hint that shows how to perform a next step. Although these hints can help students to complete a proof, they might not always help a student to understand why a certain step is useful. A study with the Geometry Tutor (McKendree 1990) shows that students who receive informative feedback combined with information about a subgoal are more effective in correcting mistakes than students who only receive informative feedback. Several logic tutors offer hints containing subgoals. An early attempt is the P-logic tutor (Lukins et al. 2002), in which students learn to construct proofs using standard equivalences and inference rules. Since this tutor cannot construct proofs, it uses heuristics to construct possible useful subgoals, such as an atomic formula from which the truth can be deduced. A drawback of this approach is that the tutor might suggest an unnecessary subgoal. The Deep Thought Logic tutor (Eagle et al. 2012; Barnes and Stamper 2008) uses datamining to construct proofs and subgoal hints from student solutions. In a comparison of the performance of students receiving next step hints with students receiving hints about a subgoal to be reached in two or three steps, the latter group outperformed the former one in the more difficult exercises both with respect to the time needed to take a step as well as accuracy (Cody et al. 2018).

In LOGAX we keep track of a list of subgoals while constructing a proof. We use this list to provide hints about a subgoal. We do not give a subgoal as hint if a student can still apply deduction backwards, when a subgoal coincides with an unmotivated line in the proof, or when a subgoal coincides with the next step. In the other cases, we give a hint concerning a subgoal instead of a hint about the direction of the proof. For instance, the hint for the unfinished proof in Fig. 5 will be: try to prove $p, p \rightarrow q \vdash q$. An example where our algorithm deliberately not gives a subgoal as hint can be found in the proof in Fig. 1. Here, he first hint will be: perform a forward step, since in this case the subgoal $p, q \rightarrow q \vdash q$, Modus Ponens, is equal to the next step.

Feedback

In this subsection we first analyze student errors, and then describe how we use this analysis to create feedback. Students make mistakes in axiomatic proofs. From the homework of 40 students participating in our course we collected a set of mistakes, and classified these mistakes in three categories:

– oversights (19),

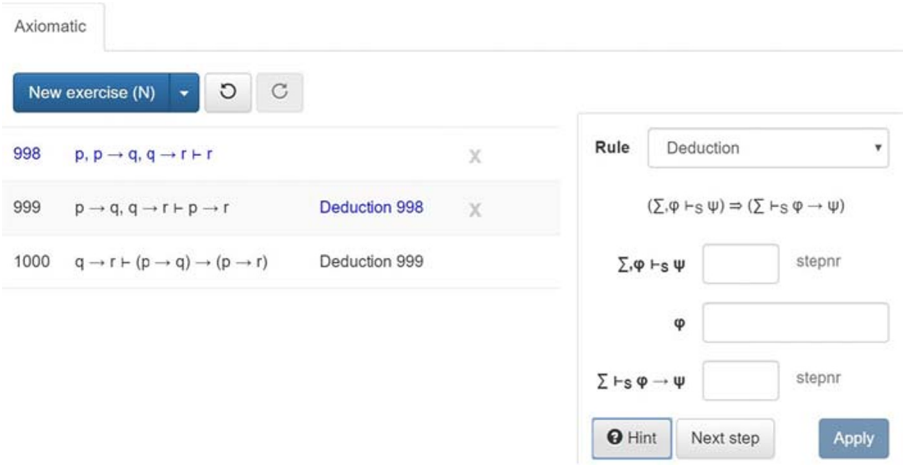


Fig. 5 The start of a proof for $q \to r \vdash (p \to q) \to (p \to r)$

– conceptual errors (11), and
– 'creative' rule adaptations (9).

Mistakes such as missing parentheses belong to the first category. This category mainly consists of missing parentheses in Axiom b. A typical example of a mistake in the second category is the following application of Modus Ponens:

1.	$\neg p, \neg q \vdash \neg q$	Assumption
2.	$\neg q \vdash \neg p \to \neg q$	Deduction 2
3.	$\vdash (\neg p \to \neg q) \to (q \to p)$	Axiom c
4.	$\neg p \to \neg q \vdash q \to p$	Modus Ponens 2, 3

Here, the student has the (wrong) idea that after an application of Modus Ponens on $\Delta \vdash \phi$ and $\Sigma \vdash \phi \to \psi$, the formula ϕ becomes the assumption of the conclusion. Creative rule adaptations may take various forms. An example of such a rule adaptation is:

1.	$\neg p \to (q \to \neg r) \vdash \neg p \to (q \to \neg r)$	Assumption
2.	$q \vdash q$	Assumption
3.	$\neg p \to (q \to \neg r), q \vdash \neg p \to \neg r$	Modus Ponens 1, 2

In this example the ultimate goal is to prove that $\neg p \to (q \to \neg r), q \vdash r \to p$. The student tries to reach this via the subgoal $\neg p \to (q \to \neg r), q \vdash \neg p \to \neg r$, but she misses the possibility to reach this subgoal with an instantiation of Axiom b and Axiom a. Instead, she creates her own variant of Modus Ponens in line 3 of the proof.

Further analysis of the homework exercises suggests that students typically make these mistakes when they do not know how to proceed. This is in line with the repair theory, which describes the actions of students when they reach an impasse (Brown and VanLehn 1980).

The example of a conceptual error given above is impossible to construct in LOGAX, since LOGAX fills in the assumptions automatically. In the evaluation in Section "Small-Scale Experiments with Students" we analyze whether students recognize these kinds of conceptual mistakes after practicing with LOGAX. However, it is still possible that a student tries to apply a rule incorrectly. For example, a student might apply Modus Ponens on $\Delta \vdash \phi$ and $\Sigma \vdash \phi' \rightarrow \psi$ where ϕ and ϕ' are equivalent but not equal. We used homework solutions to define a set of buggy rules for mistakes that can be made in LOGAX. Most of these rules relate to Modus Ponens (8 buggy forward applications, 3 buggy backward applications and 2 closure rules), the other to deduction (2 buggy backward applications and 4 buggy closure rules). Using these rules, LOGAX can give informative feedback. Shute's guidelines (2008) state that feedback should be elaborate, specific, clear, and as simple as possible. Our feedback not only points out a mistake, but if possible also mentions exactly which formula, subformula or set of formulae do not match with the rule chosen. For example, if a student wants to complete the proof

1.	$\neg q, \neg p \vdash \neg q$	Assumption
2.	$\neg q \vdash \neg p \rightarrow \neg q$	Deduction 1
3.	$\vdash (\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$	Axiom c
4.	$\neg p \rightarrow \neg q \vdash q \rightarrow p$	

by applying Modus Ponens to lines 2, 3 and 4, she gets a message that line 4 cannot be the result of an application of Modus Ponens on lines 2 and 3, since the assumption of line 2 does not belong to the set of assumptions in line 4.

Evaluation of the Generated Proofs

We first evaluate the proofs generated by LOGAX by comparing them with expert proofs. After that, we evaluate the recognition of student solutions by LOGAX. Section "Small-Scale Experiments with Students" describes the final part of the evaluation, which is a small-scale experiment with students using LOGAX.

Comparison of the Generated Proofs with Expert Proofs

We evaluate the proofs generated by LOGAX in two ways. First, we compare the generated proofs with expert proofs. Since example proofs and worked solutions in textbooks often use earlier proofs as a lemma, the number of proofs that we can compare with the LOGAX proofs without lemmas is small. We found 10 examples we could use for a comparison in the textbooks of Ben-Ari, Kelly, and Goldrei, and in the lecture notes of a course on logic (LenI) (Lodder and et al. 2018). The list of exercises can be found in Table 1.

Eight of the ten expert proofs are equal to the LOGAX proofs. In exercise 6, LOGAX uses the deduction theorem instead of Axiom a. The LOGAX proof of $\vdash \neg p \rightarrow (\neg\neg p \rightarrow p)$ (exercise 5) is one step longer than the expert proof, since LOGAX proves $\vdash \neg\neg p \rightarrow p$ directly without making use of an assumption $\neg p$. The LOGAX proof uses a standard construction. Although this standard construction is

Table 1 Exercises without lemmas in textbooks and lecture notes. The last column compares thet LOGAX proof with the expert proof

	Exercise	Textbook	Equal
1.	$q, p \to (q \to r) \vdash p \to r$	Kelly, LenI	yes
2.	$p \to q, q \to r \vdash p \to r$	Kelly, LenI	yes
3.	$p \to q \vdash p \to (r \to q)$	Kelly	yes
4.	$p \to q \vdash (r \to p) \to (r \to q)$	LenI	yes
5.	$\vdash \neg p \to (\neg\neg p \to p)$	LenI	no
6.	$\vdash ((p \to q) \to (p \to r)) \to (p \to (q \to r))$	LenI	no
7.	$\vdash (p \to q) \to ((q \to r) \to (p \to r))$	Ben-Ari	yes
8.	$\vdash (p \to (q \to r)) \to (q \to (p \to r))$	Ben-Ari	yes
9.	$\vdash \neg p \to (p \to q)$	Ben-Ari, Goldrei	yes
10.	$\vdash \neg\neg p \to p$	Ben-Ari	yes

useful in quite a lot of proofs, in this case, missing the shorter solution is a shortcoming of LOGAX since a goal of the exercise is that the student recognizes the possibility to use this assumption. However, the completion of LOGAX of a student proof that starts using this assumption in an application of Axiom c, equals the proof in the lecture notes. In the future, we might fine-tune LOGAX such that the generated solution equals the lecture notes solution also in other cases. We conclude that for most of the examples and exercises in textbooks, LOGAX generates a proof that is equal to the expert proof.

To evaluate the version of LOGAX with lemmas, we have nine expert proofs available. The course on logic contains five examples of exercises using lemmas, together with example proofs of these exercises, see Table 2. In the first three exercises only instantiated lemmas are given. The proofs generated by LOGAX for these exercises are equal to the solutions in the course notes. The fourth and fifth exercise ask for a proof in predicate logic, but we can compare the propositional part of these proofs. Both exercises present lemmas as schemas, and instantiations of these schemas are used in the solution. We add these instantiations as lemmas in LOGAX. The solution to the fifth exercise is equal to the proof generated by LOGAX, except for the order of the proof lines. In the fourth exercise, LOGAX originally only used one of the two lemmas, and the proof by LOGAX was longer than the solution in the logic course notes. After some minor changes in the implementation of the heuristics, LOGAX uses no lemmas, but generates a shorter proof. Both proofs are given in Appendix A. Since exercises in textbooks also use derived rules, we have only 4 extra exercises with lemmas in these books. All the proofs of these exercises are equal to the LOGAX proofs.

To evaluate more proofs we use the large collection of proofs on the Metamath website.[1] This website collects formal proofs, not only for logic statements, but also for mathematical statements. The part on propositional logic contains proofs of

[1] http://us.metamath.org/mpegif/mmtheorems.html

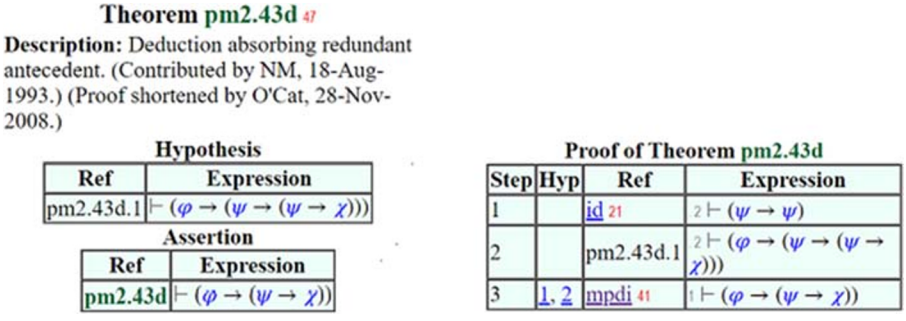
Table 2 Exercises in a logic course

Textbook	Exercise	Lemma	Equal
LenI	$\vdash (p \to q) \to (\neg q \to \neg p)$	$\vdash (p \to q) \to (\neg\neg p \to \neg\neg q)$	yes
LenI	$\neg\neg p \vdash \neg p \to \neg\neg\neg p$	$\neg\neg p \vdash \neg\neg\neg\neg p \to \neg\neg p$	yes
LenI	$\vdash \neg\neg p \to p$	$\neg\neg p \vdash \neg p \to \neg\neg\neg p$	yes
LenI	$\vdash \neg (p \to q) \to (\neg p \to \neg q)$	$\neg\neg q \vdash q, p \to q \vdash \neg\neg (p \to q)$	no
LenI	$p \to (q \to \neg p) \vdash p \to \neg q$	$\neg\neg q \vdash q$	yes
Ben-Ari	$\vdash (p \to q) \to (\neg q \to \neg p)$	$\neg\neg p \to p, \vdash \neg q \to (q \to \neg\neg q)$	yes
Ben-Ari	$\vdash (\neg p \to q) \to ((\neg p \to \neg q) \to p) \vdash (\neg p \to p) \to p$		yes
Kelly	$\vdash \neg (p \to q) \to \neg q$	$\vdash (q \to (p \to q)) \to (\neg (p \to q) \to \neg q)$	yes
Kelly	$\vdash (p \to q) \to ((\neg p \to q) \to q)$	$\vdash (\neg q \to q) \to q, \vdash (p \to q) \to (\neg q \to \neg p)$	yes

well-known theorems, for example from the Principia Mathematica by Russell and Whitehead. In general, a Metamath theorem is presented as follows:

$$\vdash \phi_1 \text{ and } \vdash \phi_2 \text{ and ... and } \vdash \phi_n \Rightarrow \vdash \psi$$

So if $\phi_1, ..., \phi_n$ are provable, then ψ is provable. Since LOGAX cannot deal with general tautologies, we translate a Metamath theorem in a theorem with assumptions. Instead of $\vdash \phi_1$ and $\vdash \phi_2$ and ... and $\vdash \phi_n \Rightarrow \vdash \psi$, we prove $\phi_1, \phi_2, ..., \phi_n \vdash \psi$. Since proofs in Metamath build on each other, these proofs seem to be natural candidates to use in a comparison with LOGAX with lemmas. However, the best way to compare proofs is not immediately clear. To demonstrate this, we present an example of a Metamath proof in Fig. 6. This example shows a proof of $\phi \to (\psi \to (\psi \to \chi)) \vdash \phi \to (\psi \to \chi)$. The proof uses two previous theorems: id ($\vdash \psi \to \psi$) and mpdi (from $\vdash \phi \to \chi$ and $\vdash \phi \to (\psi \to (\chi \to \theta))$ follows $\vdash \phi \to (\psi \to \theta)$). The most direct way to translate this in a LOGAX proof with lemmas would be by rewriting mpdi in a theorem with assumptions $(\phi \to \chi, \phi \to (\psi \to (\chi \to \theta)) \vdash \phi \to (\psi \to \theta))$, and using instantiated versions of these theorems as lemmas in the proof. However, to complete this proof, a forward application of deduction followed by an application of Modus Ponens by LOGAX suffices, which makes the comparison not very informative. A more interesting way to



Theorem pm2.43d 47

Description: Deduction absorbing redundant antecedent. (Contributed by NM, 18-Aug-1993.) (Proof shortened by O'Cat, 28-Nov-2008.)

Hypothesis

Ref	Expression
pm2.43d.1	$\vdash (\varphi \to (\psi \to (\psi \to \chi)))$

Assertion

Ref	Expression
pm2.43d	$\vdash (\varphi \to (\psi \to \chi))$

Proof of Theorem pm2.43d

Step	Hyp	Ref	Expression
1		id 21	2 $\vdash (\psi \to \psi)$
2		pm2.43d.1	2 $\vdash (\varphi \to (\psi \to (\psi \to \chi)))$
3	1, 2	mpdi 41	1 $\vdash (\varphi \to (\psi \to \chi))$

Fig. 6 An example of a Metamath proof

compare proofs would be by letting LOGAX find useful instantiations of lemmas, but so far, we have not implemented this functionality. In the comparison, we therefore add just a single instantiated lemma to LOGAX. We inline the other lemmas in the Metamath proofs. Since Metamath proofs do not make use of the deduction theorem, a last adaptation we have to make is to remove applications of the deduction theorem in the LOGAX proofs. We use the constructive proof of the deduction theorem to remove occurrences of deduction in LOGAX proofs, and compare these proofs with the Metamath proofs. The results are shown in Appendix B. The comparison consists of 24 theorems, 12 with and 12 without negation. Nearly two thirds (15 out of 24) proofs are equal except for the order of the lines. The LOGAX proof is shorter than the Metamath proof in eight cases. In seven of these cases LOGAX does not use the lemmas. It is not surprising that in these cases, inlined proofs in Metamath are longer: the Metamath proofs are constructed by choosing suitable lemmas. Metamath does not inline proofs and uses lemmas that are known theorems, or variants, for example originating from the Principia Mathematica. Hence a Metamath proof is short when it uses a small set of lemmas, without caring about the total length of the inlined proof.

In Lodder et al. (2017) we compared 30 Metamath proofs with proofs generated by LOGAX without lemmas. After inlining the used lemmas in Metamath and removing applications of the deduction theorem in the LOGAX proofs, we found that 27 LOGAX proofs were equally long as the Metamath proofs. In three cases the LOGAX proof was shorter than the Metamath proof. Although the comparison of LOGAX with Metamath can only be done indirectly, the results (from the proofs with lemmas nearly two thirds of the LOGAX proofs are equal to Metamath proofs, and most of the LOGAX proofs have equal length as these proofs) indicate that LOGAX indeed generates proofs that are comparable to expert proofs.

Recognizing Student Solutions

In a second evaluation we investigate whether or not correct student solutions can be recognized by LOGAX. Axiomatic proofs are part of a course on logic where we also deal with other topics, such as semantics of predicate logic, axiomatic proofs in predicate logic, structural induction and Hoare calculus. Students may hand in homework to earn a bonus point. The homework exercises contain one exercise about propositional logic axiomatic proofs and one on predicate logic axiomatic proofs. Furthermore, the exams usually have an exercise on axiomatic proofs. We use solutions of two homework exercises, and one exam exercise, to determine whether or not LOGAX recognizes student proofs.

In the exam exercise, students have to prove that $\neg\neg p \rightarrow \neg q, r \rightarrow q \vdash r \rightarrow \neg p$. The correct student solutions to this exercise can be divided into two groups, where each group contains solutions that are equal up to the order of the proof lines. Solutions in the first group contain an application of Axiom a, b and c, and no application of the deduction theorem. Solutions in the second group contain an application of the deduction theorem, and of Axiom c. From the 19 correct solutions, the majority (16) belongs to the second group, and the remaining three solutions to the first group. The example solution provided by LOGAX also belongs to the second group. The solutions of the first group do not (yet) appear in our initial DAM, but they do appear

Table 3 Recognized solutions

Exercise	preferred	non-pref.	dynamic	total
Exam	16		3	19
Homework 1	1		16	17
Homework 2	2	13		15

in the DAM we dynamically obtain when a student introduces Axiom b, and we use this DAM to provide feedback. In the future we might add an extra heuristic for the use of Axiom b:

– If the top goal equals $\Delta \vdash \phi \to \chi$, and $\Delta' \vdash \phi \to \psi$ and $\Delta'' \vdash \psi \to \chi$ both appear in the DAM, where $\Delta' \cup \Delta'' \subseteq \Delta$, then add an instance $(\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$ of Axiom b to the DAM.

In the first homework exercise students have to prove that $q, \neg p \to (q \to \neg r) \vdash r \to p$. Here almost all student solutions (16) use Axiom a, b and c. Only one student uses the deduction theorem instead of Axiom b. LOGAX generates this last proof. Solutions using the three axioms are not part of the DAM that is generated at the start of the exercise, but can be recognized by a dynamically generated DAM. The second homework exercise is an exercise in predicate logic, but it contains a propositional part that amounts to a proof of $(p \to q) \to \neg p \vdash q \to \neg p$. Again, there were two groups of solutions: 13 students use Axiom a and the deduction theorem, and 2 students use an extra application of the deduction theorem instead of Axiom a. In this case the solution generated by LOGAX is the solution that does not use Axiom a, but the DAM also contains a solution with Axiom a.

We summarize the results in Table 3. The first column (preferred) shows the number of solutions that corresponds to the preferred solution of LOGAX, the second the number that corresponds to a non-preferred solution, and solutions in the third column can be recognized by a dynamically generated DAM. The conclusion of this evaluation is that with the use of dynamically generated DAMs, we can recognize all student solutions, and also give hints. Still, we might optimize LOGAX by adding more heuristics, e.g. such that the solution generated by LOGAX for homework exercise 2 equals the student solutions. In the current implementation, heuristics for the use of Axiom b and the deduction theorem interfere: extra heuristics for Axiom b can broaden the DAM, but the algorithm for the distillation of a linear proof prefers applications of the deduction theorem, a local decision. We would have to extend this algorithm with global heuristics to ensure that the extracted proof contains instances of axioms when applicable.

Small-Scale Experiments with Students

We have performed several small-scale experiments with LOGAX. The main results in this section are obtained from an experiment with LOGAX without lemmas,

performed in May 2018. The 18 participants in this experiment were preparing for admission to a master program Computer Science at the Open University of the Netherlands. A course on logic is part of the premaster program. We required participants to submit a solution to the first homework exercise, see Section "Comparison of the Generated Proofs with Expert Proofs", before participating in the experiment. Thus, we guaranteed that participants had studied the subject before the experiment. We used their solutions to the exercise as an indication of their prior knowledge. The experiment consisted of a 20-minute (online) instruction, after which students practiced with the tool for 75 minutes. The experiment concluded with a 20-minute posttest. All interactions of the students with LOGAX were logged. The 10 exercises in the tool and the questions in the posttest can be found in Appendix C.

We use the results of this experiment

(1) to evaluate the hints and feedback given by LOGAX,
(2) to analyze the way students use LOGAX, and
(3) to evaluate the effect of using LOGAX on students' performance.

Evaluation of Hints and Feedback

We start with evaluating the generated hints and feedback by answering the following questions:

– does LOGAX recognize common mistakes?
– is the feedback sufficient to repair mistakes?
– do hints on subgoals help students to reach these subgoals?

To answer these questions, we analyze the log data of the experiments. Table 4 summarizes the results of this analysis, and in the following paragraphs we will discuss these results in more detail.

Apart from 5 syntax errors, the log data contain 179 incorrect steps of a total of 1480 steps performed by the students. The syntax errors were performed by 5 different students who could repair this error in the next step without asking for extra help. It seems that the dialog box indeed prevents students from spending time repairing syntax errors. In 24% (43/179) of the incorrect steps, a student tries to apply Modus Ponens, but interchanges the first and second line in the dialog box, as

Table 4 Number of different errors in logged student steps

	recognized common error	86	
	interchanged lines MP	43	
	not yet recognized common error	15	
	other incorrect steps	35	
	total number of incorrect steps		179
syntax errors			5
total number of correct steps			1296
total number of steps			1480

shown in Fig. 7. This typically occurs when the implication ($\phi \rightarrow \psi$) has a smaller line number than the antecedent of this implication (ϕ). During the experiment we did not have a buggy rule implemented for this situation, and hence students received the feedback 'ϕ is not an implication', or 'Modus Ponens is not applicable'. In 34 out of the 43 occurrences of this kind of mistake, this feedback was sufficient for the student to fill in the dialog box correctly. In the other cases students asked for a hint or next step, or continued with another rule. One student did not realize that the implication may precede the antecedent, and consequently constructed the proofs in such a way that implications always have a higher line number than the antecedents.

A second category of mistakes also seems to have its origin in an incorrect use of the dialog box. Examples are backward applications of Modus Ponens where the last line is already motivated, or the other line is unmotivated (10 occurrences). A student who puts the number 2 in the bottom field in Fig. 7 and leaves the middle field open makes this type of mistake. The feedback message in such a case was not very helpful: "Cannot apply Modus Ponens". Some buggy applications of Modus Ponens were only recognized when students completed the dialog box 'correctly' (entering the implication in the second field). For instance, an erroneous application of Modus Ponens on formulae $\Sigma \vdash \phi \rightarrow \psi$ and $\Delta \vdash \psi$ is not recognized if the student enters the line number of the first line in the uppermost (antecedent) field, and the second in the middle (implication) field. Some misreading of parentheses is recognized by LOGAX, but, for example, the misreading of parentheses in an application of Modus Ponens on $\Sigma \vdash p \rightarrow (q \rightarrow r)$ and $\Delta \vdash (p \rightarrow q) \rightarrow (r \rightarrow (p \rightarrow q))$ is not recognized as a common mistake. Also mistakes in the introduction of an axiom or assumption in the dialog box, such as interchanging p and q with a faulty Modus Ponens application on $\Sigma \vdash \neg p \rightarrow \neg q$ and $\vdash (\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q)$ as a result, is not recognized. The remaining errors that are not recognized cannot be classified as buggy rules, since we cannot find a pattern in, or a misconception as a cause of, these errors.

We further analyzed the log data to see whether in the cases that a common error is detected (86 errors), the error message is sufficient to help a student in making progress. In 60% (52/86) of these cases, a student can proceed without help of the system, in over 8% (7/86) a student gives up on the exercise directly after this mistake, and in 3% (3/86) after one or more erroneous steps, and in the other cases a student can proceed with a hint or next step. If a student needs more help, this does

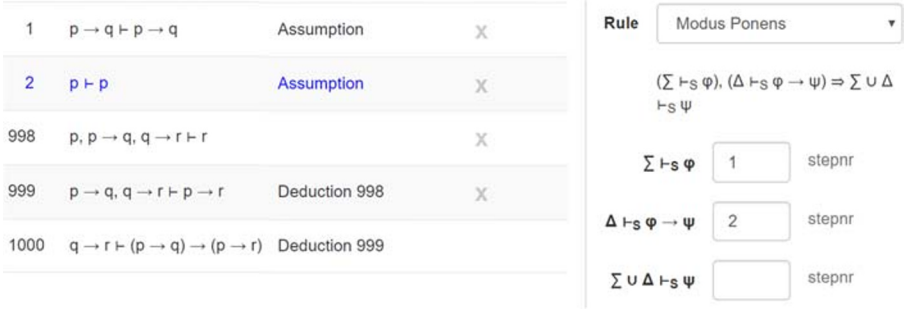


Fig. 7 A common error: interchanging line 1 and 2 in the dialog box for Modus Ponens

not necessarily mean that the error message is not clear: the log data suggest that often a student recognizes the mistake (in 60% of the cases a student does not make the same error again during the session), but does not know how to proceed.

We conclude that we can improve error messages by recognizing the cases where a student interchanges the lines in the dialog box. In the 43 cases where interchanging the lines was the only mistake, students will receive a message about how to fill in the dialog box correctly. In 15 other cases, where students now get a default message or a message that probably does not refer to the actual mistake, we will also improve the feedback. An example of this situation is the application of Modus Ponens on $\Delta \vdash p \rightarrow q$ and $\Delta' \vdash q$ (entered in this order). At this moment, students receive the error message that q is not an implication, but after recognizing this mistake as a combination of a common error and swapping lines in the dialog box, the error message will say that the formula in the second line should be equal to the left-hand side of the implication instead of the right-hand side. With these improvements we would have provided specific feedback in 80% ((86 + 43 + 15) / 179) of the mistakes made by students, instead of in 48% (86/179) of the mistakes in the version used in the experiment.

Since the possibility to give a hint about a subgoal was new in the LOGAX version that we used in this experiment, we evaluate the effect of this type of hints. A student receives a hint that indicates a subgoal to be reached if it takes more than one step to reach this subgoal, and if the subgoal is not already present in the proof as an unmotivated line. LOGAX gives a hint about a subgoal in 75 of its 192 hints, and 40 of these subgoals were reached by the students without further assistance of LOGAX. In the other cases, the students used next step hints which told them the rule to proceed or a next step. This number might seem somewhat disappointing, but a more detailed analysis shows that in general only students who ask a lot of help do not reach the subgoal without extra help. Figure 8 presents a scatter plot with the total number of different subgoal hints given to the student on the x-axis, and the number of reached subgoals after the hint, without further help on the y-axis. The figure shows that students who use fewer than eight different subgoal hints (hints in different exercises or in different stages of their proof), in general reach the subgoal themselves.

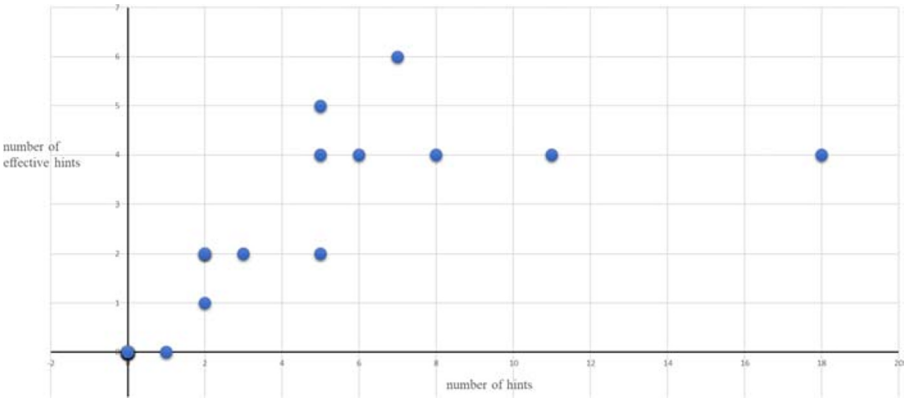


Fig. 8 The effectivity of subgoal hints

Use of LOGAX

The second part of this evaluation is the question: how do students use LOGAX. Do students misuse the system by asking too much help, or by randomly filling in the dialog boxes, or do they struggle without using the help offered by LOGAX? The results of our experiment show that the use of LOGAX is related to performance in the homework exercise, which we use as a pretest. From the 12 students who score at least 0.8 (out of 1) in the pretest, eight students can complete the exercises without using much help or making many mistakes. One student reported that he completed the exercises with pen and paper and used LOGAX mainly to check his answers. Two students use quite a lot of hints, also to complete the exercises, and one student seems to misuse the system by performing lots of actions (338 interactions versus an average of 173). From the students who score lower than 0.8 on the pretest, only one student completes all the exercises without much help. In this group help seeking strategies differ considerably: two students hardly ask any help, one student performs 65 help-seeking actions. We conclude that in this experiment, in general, good students use LOGAX as intended, and can complete exercises without a lot of help. Since only six weaker students participated, we cannot draw hard conclusions, but the log data seem to indicate that these students either tend to overuse or underuse help. Another observation is that most of the weaker students can complete the first four or five easier exercises, but the later exercises seem to be too difficult for this group. This indicates a shortcoming in our experiment: it seems the difference in difficulty between the first five and the other exercises is too big. We could regulate hint use by letting LOGAX provide unsolicited hints when students make too many mistakes or do not make progress in their exercise, and on the other hand maximize the number of hints that a student can ask for.

Evaluation of Learning Effects

The last question we want to answer is whether LOGAX supports students with learning axiomatic proofs. Most of the participants in the experiment were good students, who performed already well on the pretest: the average score on the pretest was 0.84, where the maximum possible score was 1. This might have influenced learning effects negatively. The posttest consisted of two parts. In the first part, students had to point out possible errors in five small proofs. Four of the five proofs were incorrect. The incorrect proofs contained common errors collected from the homework exercises, see Section "Hints". We deliberately added errors that are possible to make in LOGAX and errors that are prevented by the interface, since we wanted to know whether the last type would occur more often in the posttest. In the second part, students had to provide a proof.

The scores of the posttest can be found in Table 5. The low scores on exercises 1b and 1d are remarkable. The score on exercise 1b was more or less expected, because the exercise contains an error in the set of assumptions after applying Modus Ponens. Since LOGAX automatically determines this set, students do not practice in correctly determining this set. Exercise 1a also contained an error that is not possible in LOGAX (mixing an application of Modus Ponens with Axiom b), but this error

Table 5 Results of the posttest

Exercise	1a	1b	1c	1d	1e	2
Average score	0.78	0.39	0.72	0.34	0.83	0.61

was recognized by students. Exercise 1d applies deduction in 'the wrong direction', a common mistake made by students, which is apparently not sufficiently corrected while practicing with LOGAX (the log data contain only 3 occurrences of this error). The misreading of parentheses in exercise 1e (a possible error in LOGAX) is recognized by most students. Students scored lower on exercise 2, an exercise in which a student has to construct a proof, than in the pretest. We hypothesize that this is caused by fatigue (since most of our students combine study with a job, the experiment took place in an evening, and the posttest started at 21:15), and the fact that we asked students not to spend more than 20 minutes on the posttest, while they could spend as much time as they needed for the pretest, since this was part of the homework assignment.

We also looked at the results on the exam. Students who participated in the experiment receive on average the same score for the exercise on axiomatic proofs as they got for the pretest. Since most students participated in the experiment, we cannot compare their results with students who did not participate. The results of an earlier experiment (January 2018) with 9 students were more or less comparable. The average score on the pretest of this group is a little lower (0.8) and also the score on the posttest proof, exercise 2, is lower (0.52). The results on the exam are a bit higher (0.89), but the exercise was slightly different, and the time between the experiment and the exam was considerably shorter: 13 days instead of 40 days for the last experiment. We conclude that at this moment we do not have enough data to evaluate learning effects. However, experiments with other tools (Lodder et al. 2019) show that this kind of tutoring system can be effective.

Limitations

In the previous subsections, we described our pilot experiments and mentioned some limitations. First, the number of participants in the experiments is too low to draw statistical conclusions or measure learning gains. Second, most Open University students combine their study with a job, and hence the online lessons are organized in the evening. This may have influenced the results in the posttest. Third, the difference in difficulty between the first five and the remaining exercises is a problem. Weaker students could have benefited from a more gradual increase in difficulty. There are other factors that could have influenced learning effects, for example gaming behaviour of students while working with LOGAX. Since the participants in our experiments where motivated adults, preparing for admissions to a master program, we did not expect much gaming behaviour. Still, analysis of the log data shows that one student overused the possibility to let the system perform a next step. This might have been caused by frustration, one of possible causes of gaming behaviour, mentioned by Baker et al. (2008).

Related Work

As mentioned in the introduction, there are two e-learning tools that can be used to practice the construction of Hilbert-style axiomatic proofs in propositional logic: Metamath Solitaire (Megill 2007) and Gateway to logic (Gottschall 2012). Both tools are proof-editors: a student chooses an applicable rule and the system applies this rule automatically. These systems provide no help on how to construct a proof. There are quite a lot of systems that help students with other kinds of exercises in logic, and many more in other subjects.

In the AProS project, Sieg and collegues have developed Proof Tutor, a tutor that teaches students natural deduction (Sieg 2007; Perkins 2007). They have developed an automated proof search method, which differs from the Bolotov method in the use of normal proofs. Their algorithm uses a set of tactics that are explicitly used as hints for students. Perkins (2007) describes how they provide help such as hints or next steps in the case that the partial solution of a student diverges from a generated model solution. First, they check if the subgoals of the partial solution are indeed derivable from the assumptions (we do not need this check since a situation in which a subgoal is not derivable is not possible in LOGAX). Second, they check whether the partial solution can be completed by the Proof Tutor; if this is not the case, they let the student erase the lines of the part that does not belong to a generated proof. In LOGAX we do not let students erase lines. The consequence is that a final proof may contain unnecessary lines, but also that we will not erase a useful part that is not recognized as useful by LOGAX.

We use a strategy language to generate both the solutions and the feedback (Heeren et al. 2010). The use of such a language is related to the use of production rules as, for example, in Anderson et al. (1995) and Corbett et al. (1997), and the way in which we recognize student solutions is akin to their model-tracing approach. The Geometry Tutor makes use of contextualized rules, which means that a rule will only fire in a specific context. The different axioms that we add in step 2(b) of our version of the algorithm depend on assumptions in the statements $\Delta' \vdash \phi$ and $\Delta'' \vdash \neg\phi$, and these rules could also be perceived as contextualized rules. When more than one rule can be applied in the same situation, tools based on production systems may add preferences to specific rules (Anderson et al. 1995; Jaques et al. 2013). We use the strategy language to specify a preference in the application of the rules.

Ahmed et al. (2013) use a different approach to generate and solve natural deduction exercises. Their main idea is to use truth tables, representing an equivalence class of logical formulae. The representations of these formulae are used in a proof graph of predefined size. Proof generation consists of finding a truth-table-representative of the assumptions and conclusion, searching for a proof using these representatives, and adding rewrite steps (such as replacing subformulae of the form $\neg\neg\phi$ by ϕ, or vice versa) to this proof. In this way, they can generate exercises and solutions typically used in education. However, in their approach it is essential that rewrite steps on subformulae are allowed, which is not the case in Hilbert axiomatic systems, and also often not in natural deduction systems.

Answer set programming, for example used by O'Rourke et al. (2019), is related to the production systems used by Anderson et al. (1995). Their program finds all different solutions of an algebra exercise, using deductive rules and integrity constraints that forbid certain solutions. Also explanations and a subset of misconceptions can be generated automatically. To restrict the search space to a finite space (which is necessary in their program), the number of terms in an equation and the solution length is maximized. In LOGAX we do not need to maximize the length of formulae; formula length is not a good measure for the expected proof length. For technical reasons, we maximize calculation time, but thus far LOGAX has been able to solve all problems within this limit.

For some domains it is possible to use existing tools to generate solutions or correct student solutions. An example of this approach is (Sadigh et al. 2012): to solve state machine problems such as finding a trace for a given model that violates or satisfies a certain property, they use existing model checking tools. This approach can be very useful to produce solutions or grade exercises, but is in general less suitable for giving hints, since steps performed by a tool not always correspond to human steps.

Conclusion and Future Work

By using an existing algorithm for natural deduction, we developed a sound and complete algorithm to generate Hilbert-style axiomatic proofs, and introduced a representation of these proofs as a directed acyclic multi graph (DAM). We use these DAMs in a new interactive tutoring tool LOGAX to give hints and next steps to students, and to extract model solutions. Comparing the generated proofs with expert solutions shows that the quality of the proofs is comparable to that of expert proofs. The tool recognizes most of the steps in a set of student solutions, and in case a step diverges from the generated proof, LOGAX can still provide hints and next steps. This holds both for the original version of LOGAX as well as for the extension with lemmas. We derived buggy rules from a set of student solutions, and added these to LOGAX. Evaluation with a test set showed that this set covers the majority of student errors. In an experiment with students we discovered that we overlooked a source of errors originating in the user interface (i.e. errors made while filling in the dialog box for Modus Ponens). We expect that after adding buggy rules for this kind of error, LOGAX will recognize about 80% of the errors.

We performed several pilot evaluations with LOGAX. Since the number of participating students was low, and students performed already well on the pretest, we cannot derive conclusions about the learning effect of LOGAX. However, we conclude that well-prepared students use the system as intended, and can complete most of the exercises without using much help or making a lot of mistakes. Students who do not ask more hints than on average, reach the subgoal given in a hint without extra help. Future evaluations can investigate whether students indeed learn by using LOGAX. It might be necessary to add more easy exercises for weaker students, since the results of the evaluation indicate that the difficulty level of the exercises rises too quickly for

this category of students. Also mechanisms to diminish excessive hint use, or stimulate hint use in case of minimal use, might help the weaker students to benefit from the use of LOGAX.

We developed algorithms for generating DAMs and distilling proofs from this DAM for the domain of Hilbert-style axiomatic proofs. The strategy language used to formulate these algorithms is much broader applicable, and we expect that also some of the ideas used in our algorithms can be applied to other problem domains. For example, the dynamic extension of partial student proofs, or the use of the strategy language to order the introduction of the proof steps in a way that corresponds to an expert's pen-and-paper proof, could be based on techniques similar to LOGAX. For instance, it would be interesting to compare the approach in the AProS project with our approach. Contrary to the LOGAX approach, AProS erases lines in a partial student proof that are not used in the generated completion. These different approaches could have different effects on learning and motivation.

Also geometry tutors could benefit from our approach. The geometry prover used by Matsuda and VanLehn (2004) restricts the number of points that can be used as a starting point in a construction to prevent combinatorial explosion. With dynamically generated strategies it could be possible to add points introduced by students without a high increase in CPU time. QED-Tutrix is a geometry tutor that uses a different approach by first extending a problem figure into a super-figure that contains possible useful points and segments (Font et al. 2020). A student who uses this system can only use points and segments in the figure and the super-figure. The authors state that a limitation of their system is the necessity to provide all the possible elements of the proof in advance. With the dynamic approach, it could be possible to find also solutions using extra points of segments introduced by students.

Acknowledgments We thank Marianne Berkhof and Daan de Wit for their work on the student interface, and our students from the course 'Logic and Computer Science' for their permission to use their solutions in our research, and participation in the experiment. The reviewers made many useful comments on previous versions of this paper.

Appendix A: Exercise 11.1.5

In Tables 6 and 7 we compare the solution of the logic course exercise 11.1.5 by the solution generated by LOGAX. Note that the second solution does not use the lemmas and hence is in fact shorter than the solution in the course.

Table 6 Solution of exercise 11.1.5 in the logic course:

1.	$\vdash q \rightarrow (p \rightarrow q)$	Axiom a
2.	$\neg\neg q \vdash q$	Lemma
3.	$\neg\neg q \vdash p \rightarrow q$	Modus Ponens 1 2
4.	$p \rightarrow q \vdash \neg\neg(p \rightarrow q)$	Lemma
5.	$\vdash (p \rightarrow q) \rightarrow \neg\neg(p \rightarrow q)$	Deduction 4
6.	$\neg\neg q \vdash \neg\neg(p \rightarrow q)$	Modus Ponens 3 5
7.	$\vdash \neg\neg q \rightarrow \neg\neg(p \rightarrow q)$	Deduction 6
8.	$\vdash (\neg\neg q \rightarrow \neg\neg(p \rightarrow q)) \rightarrow (\neg(p \rightarrow q) \rightarrow \neg q)$	Axiom c
9.	$\vdash \neg(p \rightarrow q) \rightarrow \neg q$	Modus Ponens 7 8
10.	$\neg(p \rightarrow q) \vdash \neg(p \rightarrow q)$	Assumption
11.	$\neg(p \rightarrow q) \vdash \neg q$	Modus Ponens 9 10
12.	$\vdash \neg q \rightarrow (\neg p \rightarrow \neg q)$	Axiom a
13.	$\neg(p \rightarrow q) \vdash \neg p \rightarrow \neg q$	Modus Ponens 11 12
14.	$\vdash \neg(p \rightarrow q) \rightarrow (\neg p \rightarrow \neg q)$	Deduction 13

Table 7 Solution of exercise 11.1.5 generated by LogAx:

1.	$\neg\neg q \vdash q$	Lemma
2.	$p \rightarrow q \vdash \neg\neg(p \rightarrow q)$	Lemma
3.	$\neg p \vdash \neg p$	Assumption
4.	$\vdash \neg p \rightarrow (\neg q \rightarrow \neg p)$	Axiom a
5.	$\neg p \vdash \neg q \rightarrow \neg p$	Modus Ponens 3 4
6.	$\vdash (\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q)$	Axiom c
7.	$\neg p \vdash p \rightarrow q$	Modus Ponens 5 6
8.	$\neg(p \rightarrow q) \vdash \neg(p \rightarrow q)$	Assumption
9.	$\neg(p \rightarrow q) \vdash \neg\neg q \rightarrow \neg(p \rightarrow q)$	Deduction 8
10.	$\vdash (\neg\neg q \rightarrow \neg(p \rightarrow q)) \rightarrow ((p \rightarrow q) \rightarrow \neg q)$	Axiom c
11.	$\neg(p \rightarrow q) \vdash (p \rightarrow q) \rightarrow \neg q$	Modus Ponens 9 10
12.	$\neg(p \rightarrow q), \neg p \vdash \neg q$	Modus Ponens 7 11
13.	$\neg(p \rightarrow q) \vdash \neg p \rightarrow \neg q$	Deduction 12
14.	$\vdash \neg(p \rightarrow q) \rightarrow (\neg p \rightarrow \neg q)$	Deduction 13

Appendix B: Metamath Theorems Compared with LOGAX with Lemmas

Table 8 compares a set of proofs in Metamath with generated proofs by LOGAX using lemmas. The first column gives the name of the theorem in Metamath, the second column the theorem and the third the lemma that is used. The last two columns show the number of steps. Proofs with the same number of steps are equal except for any differences in the order of the steps.

Table 8 Comparison of Metamath proofs with LOGAX with lemmas

name	theorem	used lemma	nr. steps Metamath	nr. steps LOGAX
idd	$\vdash p \to (q \to q)$	$\vdash q \to q$	3	3
pm2.27	$\vdash p \to ((p \to q) \to q)$	$\vdash (p \to q) \to (p \to q)$	9	9
pm2.43d	$p \to (q \to (q \to r)) \vdash p \to (q \to r)$	$\vdash q \to q$	13	13
pm2.43	$\vdash (p \to (p \to q)) \to (p \to q)$	$\vdash p \to ((p \to q) \to q)$	3	12
syldd	$p \to (q \to (r \to s)), p \to (q \to (s \to t)) \vdash p \to (q \to (r \to t))$	$\vdash (s \to t) \to ((r \to s) \to (r \to t))$	21	21
imim1d	$p \to (q \to r) \vdash p \to ((r \to s) \to (q \to s))$	$\vdash p \to (s \to s)$	39	22
imim1	$\vdash (p \to q) \to ((q \to r) \to (p \to r))$	$\vdash (p \to q) \to (p \to q)$	33	16
pm2.83	$\vdash (s \to (p \to q)) \to ((s \to (q \to r)) \to (s \to (p \to r)))$	$\vdash (p \to q) \to ((q \to r) \to (p \to r))$	11	11
com23	$p \to (q \to (r \to s)) \vdash p \to (r \to (q \to s))$	$\vdash r \to ((r \to s) \to s)$	29	26
pm2.04	$\vdash (p \to (q \to r)) \to (q \to (p \to r))$	$\vdash (p \to (q \to r)) \to (p \to (q \to r))$	23	20
com34	$s \to (t \to (p \to (q \to r))) \vdash s \to (t \to (q \to (p \to r)))$	$\vdash (p \to (q \to r)) \to (q \to (p \to r)))$	11	11
loowoz	$\vdash ((p \to q) \to (p \to r)) \to ((q \to p) \to (q \to r))$	$\vdash ((p \to q) \to (p \to r)) \to (q \to (p \to r))$	7	7
pm2.21	$\vdash \neg p \to (p \to q)$	$\vdash \neg p \to \neg p$	7	8
pm2.24	$\vdash p \to (\neg p \to q)$	$\vdash \neg p \to (p \to q)$	9	9
pm2.28	$\vdash (\neg p \to p) \to p$	$\vdash \neg p \to (p \to \neg (\neg p \to p))$	17	17
pm2.18d	$p \to (\neg q \to q) \vdash p \to q$	$\vdash (\neg q \to q) \to q$	7	7
notnot2	$\vdash \neg\neg p \to p$	$\vdash \neg\neg p \to (\neg p \to p)$	29	22
notnotrd	$p \to \neg\neg q \vdash p \to q$	$\vdash \neg\neg q \to q$	7	7
notnotri	$\neg\neg p \vdash p$	$\vdash \neg\neg p \to p$	3	3
con2d	$p \to (q \to \neg r) \vdash p \to (r \to \neg q)$	$\vdash \neg\neg q \to q$	29	25
pm3.2im	$\vdash p \to (q \to \neg (p \to \neg q))$	$\vdash p \to ((p \to \neg q) \to \neg q))$	49	36
jc	$p \to q, p \to r \vdash p \to \neg (q \to \neg r)$	$\vdash q \to (r \to \neg (q \to \neg r))$	37	11
expi	$\neg (p \to \neg q) \to r \vdash p \to (q \to r)$	$\vdash p \to (q \to \neg (p \to \neg q))$	11	11
pm2.251	$\vdash \neg (p \to q) \to (q \to p)$	$\vdash \neg (p \to q) \to p)$	7	7

Appendix C: Exercises Used in the Experiment and the Posttest

The exercises presented by LOGAX in the experiment are listed in Table 9, while the questions of the posttest are presented below:

Exercise 1

Check whether these proofs are correct. In case of an incorrect proof, indicate the incorrect step, and explain why this step is not correct.

a

1.	$p \to (p \to \neg q) \vdash (p \to p) \to (p \to \neg q)$	Axiom b
2.	$p \to p \vdash p \to p$	Assumption
3.	$p \to (p \to \neg q) \vdash p \to \neg q$	Modus Ponens 1, 2

b

1.	$p \vdash p$	Assumption
2.	$p \to (q \to r) \vdash p \to (q \to r)$	Assumption
3.	$p, p \to (q \to r) \vdash q \to r$	Modus Ponens 1, 2
4.	$q \vdash q$	Assumption
5.	$q \to r, q \vdash r$	Modus Ponens 3, 4

c

1.	$p \vdash p$	Assumption
2.	$\vdash p \to p$	Deduction 1
3.	$\vdash (p \to p) \to (p \to (p \to p))$	Axiom a
4.	$\vdash p \to (p \to p)$	Modus Ponens 2, 3

d

1.	$p \to q \vdash p \to q$	Assumption
2.	$p \to q, p \vdash q$	Deduction 1

e

1.	$p \to (q \to r), p, p \to q \vdash p \to (q \to r)$	Assumption
2.	$p \to (q \to r), p, p \to q \vdash p \to q$	Assumption
3.	$p \to (q \to r), p, p \to q \vdash r$	Modus Ponens 1, 2

Exercise 2

Prove axiomatic: $(p \to q) \to r \vdash q \to r$

Table 9 Exercises in LOGAX

1.	$\neg q \to \neg p \vdash p \to q$
2.	$p, p \to q, q \to r \vdash r$
3.	$p \to (q \to r) \vdash (p \to q) \to (p \to r)$
4.	$\vdash p \to ((p \to q) \to q)$
5.	$p, p \to q \vdash r \to q$
6.	$p \to q \vdash (r \to p) \to (r \to q)$
7.	$p, \neg p \vdash q$
8.	$q, (p \to q) \to r \vdash r$
9.	$p \to \neg q \vdash p \to (q \to r)$
10.	$q, \neg (p \to q) \vdash r$

References

Aguilera, G., de Guzmán, I.P., Ojeda, M., Valverde, A. (2000). Master theses for providing feedback to the logic classroom. In Manzano, M. (Ed.) *Proceedings of the first international congress on tools for teaching logic* (pp. 169–173).

Ahmed, U., Gulwani, S., Karkare, A. (2013). Automatically generating problems and solutions for natural deduction. IJCAI International Joint Conference on Artificial Intelligence 1968–1975.

Alvin, C., Gulwani, S., Majumdar, R., Mukhopadhyay, S. (2014). Synthesis of geometry proof problems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (pp. 245–252). Association for the Advancement of Artificial Intelligence (AAAI).

Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R. (1995). Cognitive tutors: lessons learned. *The Journal of the Learning Sciences*, *4*(2), 167–207.

Arun, K. (2002). Introduction to logic for computer science. Retrieved from http://www.cse.iitd.ernet.in/sak/courses/ilcs/logic.pdf.

Association for Computing Machinery (ACM), & IEEE Computer Society Joint Task Force on Computing Curricula (2013). Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Retrieved from http://www.acm.org/education/CS2013-final-report.pdf.

Baker, R., Walonoski, J., Heffernan, N., Roll, I., Corbett, A., Koedinger, K. (2008). Why students engage in "gaming the system" behavior in interactive learning environments. *Journal of Interactive Learning Research*, *19*(2), 185–224.

Barnes, T., & Stamper, J. (2008). Toward automatic hint generation for logic proof tutoring using historical student data. In Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (Eds.) *Intelligent tutoring systems* (pp. 373–382). Berlin: Springer. ISBN 978-3-540-69132-7.

Beeson, M.J. (1998). Design principles of MathPert: Software to support education in algebra and calculus. In Kajler, N. (Ed.) *Computer-human interaction in symbolic computation* (pp. 89–115): Springer.

Belland, B.R. (2017). Instructional scaffolding: Foundations and evolving definition. In *Instructional Scaffolding in STEM Education: Strategies and efficacy evidence* (pp. 17–53). Cham: Springer International Publishing. ISBN 978-3-319-02565-0.

Ben-Ari, M. (2012). *Mathematical logic for computer science*, 3rd edn. Berlin: Springer Science & Business Media.

Bernshteĭn, N.A. (1967). *The co-ordination and regulation of movements*. Oxford: Pergamon Press.

Bolotov, A., Bocharov, A., Gorchakov, A., Shangin, V. (2005). Automated first order natural deduction. In *Proceedings IICAI'05: the 2nd Indian international conference on artificial intelligence* (pp. 1292–1311).

Bornat, R. (2017). Jape. Retrieved from https://www.cs.ox.ac.uk/people/bernard.sufrin/personal/jape.org/MANUALS/natural_deduction_manual.pdf.

Broda, K., Ma, J., Sinnadurai, G., Summers, A. (2006). Friendly e-tutor for natural deduction. In *Proceedings TFM'06: the Conference on Teaching Formal Methods: Practice and Experience*.

Brown, J.S., & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, *4*(4), 379–426.

Cody, C., Mostafavi, B., Barnes, T. (2018). Investigation of the influence of hint type on problem solving behavior in a logic proof tutor. In *Artificial intelligence in education 19th international conference, AIED 2018* (pp. 58–62): Springer. ISBN 978-3-319-93845-5. https://doi.org/10.1007/978-3-319-93846-2_11.

Corbett, A.T., Koedinger, K.R., Anderson, J.R. (1997). Intelligent tutoring systems. In Helander, M., Landauer, T.K., Prahu, P. (Eds.) *Handbook of human-computer interaction*. 1st edn. (pp. 849–874): Elsevier Science.

Eagle, M., Johnson, M.W., Barnes, T. (2012). Interaction networks: Generating high level hints based on network community clusterings. In *Proceedings of the international conference on Educational Data Mining (EDM)* (pp. 164–167).

Enderton, H. (2001). *A mathematical introduction to logic*. Amsterdam: Elsevier Science. ISBN 9780080496467.

Font, L., Cyr, S., Richard, P., Gagnon, M. (2020). Automating the generation of high school geometry proofs using prolog in an educational context. *Electronic Proceedings in Theoretical Computer Science*, *313*, 1–16. https://doi.org/10.4204/EPTCS.313.1.

Galafassi, F.F.P. (2012). Agente pedagógico para mediação do processo de ensino-aprendizagem da dedução natural na lógica, PhD thesis, Universidade do Vale do Rio dos Sinos.

Galafassi, F.F.P., Santos, A.V., Peres, R.K., Vicari, R.M., Gluz, J.C. (2015). Multi-platform interface to an ITS of propositional logic teaching. In Bajo, J., et al. (Eds.) *Proceedings PAAMS'15: Highlights of practical applications of agents, multi-agent systems, and sustainability, volume 524 of Communications in Computer and Information Science* (pp. 309–319): Springer. ISBN 978-3-319-19033-4. https://doi.org/10.1007/978-3-319-19033-4_26.

Goldrei, D. (2005). *Propositional and predicate calculus, a model of argument*. Berlin: Springer. ISBN 0387573895. https://doi.org/10.1007/1-84628-229-2.

Gottschall, C. (2012). The gateway to logic. Retrieved from https://logik.phl.univie.ac.at/chris/gateway/formular-uk.html.

Harrison, J. (2009). *Handbook of practical logic and automated reasoning*, 1st edn. New York: Cambridge University Press. ISBN 0521899575, 9780521899574.

Heeren, B., & Jeuring, J. (2014). Feedback services for stepwise exercises Science of Computer Programming. *Special Issue on Software Development Concerns in the e-Learning Domain*, *88*, 110–129.

Heeren, B., Jeuring, J., Gerdes, A. (2010). Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, *3*(3), 349–370.

Hirst, H.P., & Hirst, J.L. (2015). A primer for logic and proof (2015 edition). Retrieved from http://www.appstate.edu/hirstjl/primer/hirst.pdf.

Huth, M., & Ryan, M. (2004). *Logic in computer science: Modelling and reasoning about systems*. Cambridge: Cambridge University Press. ISBN 9781139453059.

Jaques, P., Seffrin, H., Rubi, G., de Morais, F., Ghilardi, C., Bittencourt, I., Isotani, S. (2013). Rule-based expert systems to support step-by-step guidance in algebraic problem solving: The case of the tutor pat2math. *Expert Systems with Applications*, *40*, 5456–5465. https://doi.org/10.1016/j.eswa.2013.04.004.

Kalmár, L. (1935). ÜBer die axiomatisierbarkeit des aussagenkalküls. *Acta scientiarum mathematicarum*, *7*, 222–243.

Kelly, J. (1997). *The essence of logic. The essence of computing series*. Upper Saddle River: Prentice Hall. ISBN 9780133963755.

Leary, C., & Kristiansen, L. (2015). A friendly introduction to mathematical logic. SUNY Geneseo. ISBN 9781942341079.

Lodder, J., Heeren, B., Jeuring, J. (2016). A domain reasoner for propositional logic. *Journal of Universal Computer Science*, *22*(8), 1097–1122.

Lodder, J., Heeren, B., Jeuring, J. (2017). Generating hints and feedback for hilbert-style axiomatic proofs. In Caspersen, M.E., Edwards, S.H., Barnes, T., Garcia, D.D. (Eds.) *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education, Seattle, WA, USA, March 8–11, 2017* (pp. 387–392): ACM. https://doi.org/10.1145/3017680.3017736.

Lodder, J., Heeren, B., Jeuring, J. (2019). A comparison of elaborated and restricted feedback in LogEx, a tool for teaching rewriting logical formulae. *Journal of Computer Assisted Learning*, *35*(5), 620–632. https://doi.org/10.1111/jcal.12365.

Lodder, J., et al. (2018). Logica en informatica; Lecture notes (in Dutch). Open Universiteit Nederland. ISBN 9789492739094.

Lukins, S., Levicki, A., Burg, J. (2002). A tutorial program for propositional logic with human/computer interactive learning. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education 2002* (pp. 381–385), https://doi.org/10.1145/563487.563490.

Matsuda, N., & VanLehn, K. (2004). Gramy: A geometry theorem prover capable of construction. *Journal of Automated Reasoning*, *32*(1), 3–33. ISSN 0168-7433. https://doi.org/10.1023/B:JARS.0000021960.39761.b7.

Matsuda, N., & VanLehn, K. (2005). Advanced geometry tutor: An intelligent tutor that teaches proof-writing with construction. In *Proceedings of the 2005 conference on artificial intelligence in education: Supporting learning through intelligent and socially informed technology* (pp. 443–450). Amsterdam: IOS Press. ISBN 1-58603-530-4.

McKendree, J. (1990). Effective feedback content for tutoring complex skills. *Human-computer Interaction*, *5*, 381–413. https://doi.org/10.1207/s15327051hci0504_2.

Megill, N.D. (2007). *Metamath: A computer language for pure mathematics*. Morrisville: Lulu Press.

Mendelson, E. (2015). *Introduction to mathematical logic. Discrete mathematics and its applications*, 6th edn. Boca Raton: CRC Press. ISBN 9781482237788.

Merrill, D.C., Reiser, B.J., Ranney, M., Trafton, J.G. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences*, *2*(3), 277–305. https://doi.org/10.1207/s15327809jls0203_2.

Mostafavi, B., & Barnes, T. (2016). Evolution of an intelligent deductive logic tutor using data-driven elements. International Journal of Artificial Intelligence in Education, pp 1–32. ISSN 1560-4306. https://doi.org/10.1007/s40593-016-0112-1.

Nievergelt, Y. (2002). *Foundations of logic and mathematics: Applications to computer science and cryptography*. Boston: Birkhäuser. ISBN 9780817642495.

O'Rourke, E., Butler, E., Tolentino, A.D., Popović, Z. (2019). Automatic generation of problems and explanations for an intelligent algebra tutor. In *20th international conference on artificial intelligence in education, AIED 2019 - Chicago, United States*.

Perkins, D. (2007). Strategic proof tutoring in logic. Master's thesis, Carnegie Mellon. Retrieved from http://archive.org/details/thesis_201502.

Robson, D., Abell, W., Boustead, T. (2012). Encouraging students to think strategically when learning to solve linear equations. International journal for mathematics teaching and learning. Retrieved from http://www.cimt.org.uk/journal/robson.pdf.

Sadigh, D., Seshia, S.A., Gupta, M. (2012). Automating exercise generation: A step towards meeting the MOOC challenge for embedded systems. In: Proceedings of the workshop on embedded and cyber-physical systems education, WESE '12, New York, NY, USA. Association for Computing Machinery. ISBN 9781450317658. https://doi.org/10.1145/2530544.2530546.

Shute, V.J. (2008). Focus on formative feedback. *Review of Educational Research*, *78*(1), 153–189.

Sieg, W. (2007). The AProS project: Strategic thinking & computational logic. *Logic Journal of the IGPL*, *15*(4), 359–368. https://doi.org/10.1093/jigpal/jzm026.

van Benthem, J. (2003). Logica voor informatica. Pearson Education Benelux B.V. ISBN 9789043007221.

Varga, K.P., & Várterész, M. (2006). Computer science, logic, informatics education. *Journal of Universal Computer Science*, *12*(9), 1405–1410.

Wasilewska, A. (2018). *Logics for computer science*. Berlin: Springer. ISBN 978-3-319-92591-2.

Wood, D., Bruner, J.S., Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, *17*, 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x.