Generating hints and feedback for Hilbert-style axiomatic proofs

Josje Lodder

Bastiaan Heeren

Johan Jeuring

Technical Report UU-CS-2016-009
December 2016

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Generating hints and feedback for Hilbert-style axiomatic proofs

Josje Lodder
Open University of the Netherlands
josje.lodder@ou.nl

Bastiaan Heeren
Open University of the Netherlands
bastiaan.heeren@ou.nl

Johan Jeuring
Open University of the Netherlands and Utrecht University
J.T.Jeuring@uu.nl

ABSTRACT

This paper describes an algorithm to generate Hilbert-style axiomatic proofs. Based on this algorithm we develop LOGAX, a new interactive tutoring tool that provides hints and feedback to a student who stepwise constructs an axiomatic proof. We compare the generated proofs with expert 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. If a student diverges from the generated solution, LOGAX can still provide hints and feedback.

Keywords

propositional logic, axiomatic proofs, Hilbert axiom system, feedback, hints, intelligent tutoring, e-learning

1. 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 [3]. 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 also spend a couple of pages on Hilbert systems [14, 19, 8, 15]. 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 [25], and are treated in several textbooks on logic for computer science [4, 21, 2, 24].

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 [9, 10]. 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 [23, 22, 6]. In these ITSs, students construct proofs, and get hints and feedback. We found two e-learning tools that help students producing axiomatic proofs: Metamath Solitaire [18] and Gateway to logic [11]. 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, next steps, and complete solutions. LOGAX is part of a set 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 [17, 16].

The main contributions of this paper are:

- an algorithm for generating axiomatic proofs, and
- the generation of hints and feedback based on this algorithm.

To determine the quality of 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 to LOGAX.

This paper is organized as follows. Section 2 describes Hilbert's axiom system. Section 3 introduces the algorithm to generate proofs automatically, and Section 4 explains how we use these generated proofs for providing hints. Section 5 gives a collection of buggy rules. Section 6 shows the results of a first evaluation of our work. Section 7 concludes and presents some ideas for future work.

2. AN E-LEARNING TOOL FOR HILBERT-STYLE AXIOMATIC PROOFS

Several variants of axiomatic proof systems exist. In LOGAX we use the following set of axioms:

$\phi \to (\psi \to \phi)$	Axiom a
$(\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$	Axiom b
$(\neg\phi \to \neg\psi) \to (\psi \to \phi)$	Axiom c

SIGCSE '17, March 08 - 11, 2017, Seattle, WA, USA

DOI: 10.475/123_4

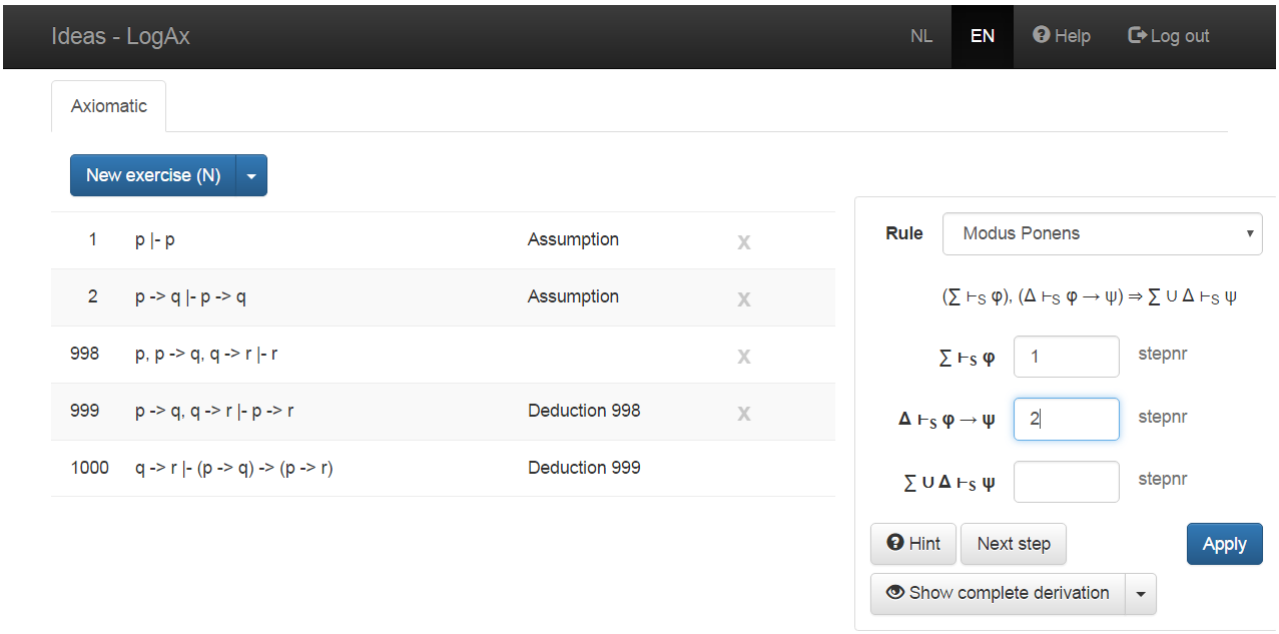

Figure 1: A partial proof of $q \to r \vdash (p \to q) \to (p \to r)$ performed in LOGAX

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 instance of an axiom, an assumption, or an application of the Modus Ponens (MP) rule:

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

LOGAX also supports a derived rule, the deduction theorem:

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

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 partial proof, constructed in our tool LOGAX. A full proof that completes this partial proof is:

1.	$p \vdash p$	Assumption
2.	$p \to q \vdash p \to q$	Assumption
3.	$p, p \to q \vdash q$	Modus Ponens, 1, 2
4.	$q \to r \vdash q \to r$	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 deduction theorem is applied in a forward step, the student also provides a formula ϕ. 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.

3. AN ALGORITHM FOR GENERATING AXIOMATIC PROOFS

An ITS for axiomatic proofs needs to provide 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 [20]. 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

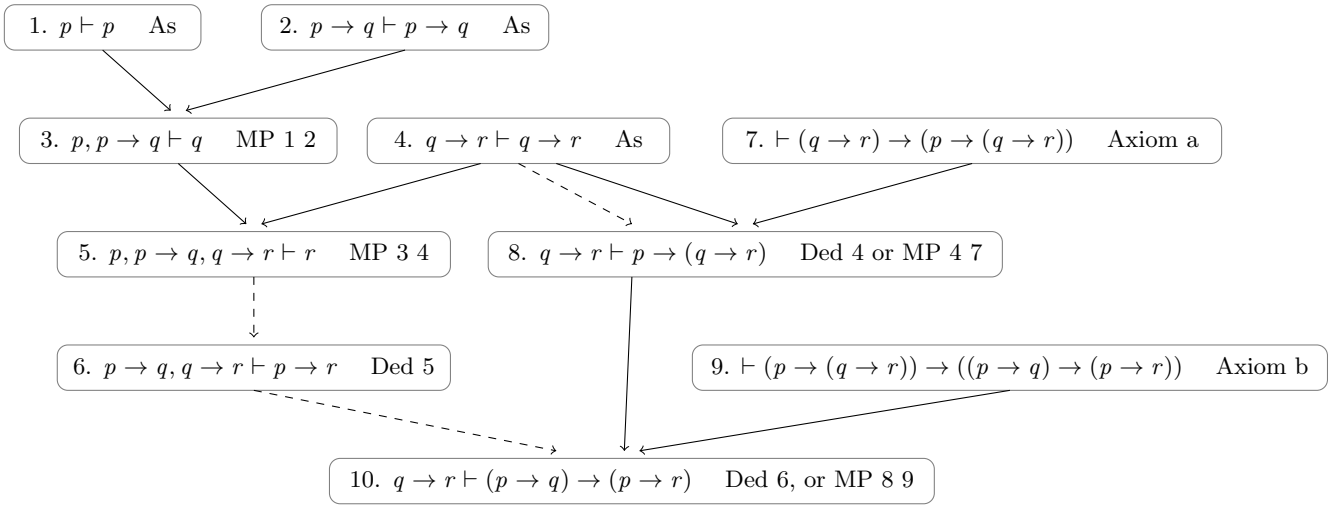

Figure 2: A DAM for the proof of $q \to r \vdash (p \to q) \to (p \to r)$

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 solve only the second problem: automatically providing hints for new exercises. Section 4 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 the kind of proofs we expect from our students. Existing algorithms, such as the Kalmar constructive completeness proof [13], or the algorithms used in automatic theorem proving [12], are unsuitable for this purpose. Natural deduction tools such as ProofLab [23] and Pandora [6] also use solution algorithms, 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 a different order, for example starting with 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 leaves with the applied rule: Assumption, Axiom, Modus Ponens or Deduction. Note that a state can be the result of different applications of rules. An example of such a DAM is shown in Figure 2. Vertices are numbered for readability. A dashed arrow means that the lower statement follows from the higher by application of the deduction theorem. A pair of ordinary arrows represents an application of Modus Ponens. This DAM contains three essentially different proofs: one that uses axioms 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 Figure 1.

The basis for our algorithm for axiomatic proofs is Bolotov's algorithm for natural deduction proofs [5]. The 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. $q \to r \vdash (p \to q) \to (p \to r)$) to our stack of goals, and the assumptions of this goal ($q \to r \vdash q \to r$) 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 $\Delta' \cup \Delta'' \subset \Delta$, 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 ($\vdash \neg\neg\phi \to (\neg\neg\neg\neg\phi \to \neg\neg\phi)$) 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 formulae.
   (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 $\Delta \setminus \phi \vdash \phi \to \psi$, we add $\Delta \setminus \phi \vdash \phi \to \psi$ to the DAM.

Motivation: use the deduction theorem.

4. (a) If the top goal is $\Delta \vdash \phi \to \psi$, we add $\phi \vdash \phi$ to the DAM and the goal $\Delta, \phi \vdash \psi$ to our stack of goals.

Motivation: prove $\Delta \vdash \phi \to \psi$ 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 $\Delta \vdash \phi \to \psi$ 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 correct and complete. Our adaptations are such that correctness and completeness are preserved. We omit a detailed proof.

The above algorithm only uses axioms in a proof of a contradiction, or in the use of double negations. However, we want our students to recognize the possibility to use axioms. We use some extra heuristic rules to add more instances of axioms to the DAM, such that we indeed can produce the example DAM in Figure 2. An example of such a rule is:

– If the top goal equals $\Delta \vdash \phi \to \psi$ and $\Sigma \vdash \neg\phi$ already belongs to the DAM, where $\Sigma \setminus \neg\psi \subset \Delta$, then add an instance $(\neg\psi \to \neg\phi) \to (\phi \to \psi)$ of Axiom c to the DAM together with a derivation of $\Sigma \setminus \neg\psi \vdash \neg\psi \to \neg\phi$ by the deduction theorem from $\Sigma \vdash \neg\phi$.

4. MODEL SOLUTIONS AND HINTS

The DAM may contain different solutions. Figure 2 shows, for example, three essentially different solutions for the proof of $q \to r \vdash (p \to q) \to (p \to r)$. We use the DAM to provide complete solutions and to trace the solution constructed by a student to give adequate feedback or hints. Therefore, we extract solutions stepwise from the DAM. In each step there is a choice between different lines that can be added to the proof. We add preferences to these choices. In general, backward applications of the deduction theorem are preferred over forward steps, and axioms or assumptions that can be used in an application of Modus Ponens together with statements that are already in the proof, are preferred above statements that cannot be used directly. For example, when we apply the extraction procedure to construct a sample solution from the example DAM (Figure 2), the first line we add to the proof is a backward step: the goal can be reached by an application of the deduction theorem. Also the next line will be an application of the deduction theorem. Now, we have to introduce an assumption, and in this case the procedure will choose line 1 or 2 from the DAM, because both these lines can be used in a next step. It will not choose line 4, since line 4 can only be used after the application of Modus Ponens on lines 1 and 2.

LOGAX uses the above extraction procedure to generate complete solutions, and to provide hints or next steps. When a student works on an exercise, LOGAX compares her steps with the steps in the DAM for the exercise. As long as the student follows a preferred or non-preferred solution from the DAM, LOGAX can give hints (e.g. perform a backward step, or apply Modus Ponens), or next steps. LOGAX compares the partial solution of the student with the DAM to find a solution path that is a continuation of the solution of the student. This solution path is used to generate hints or next steps. When a student diverges from the DAM, we build a new DAM that contains the (partial) student solution. This DAM can then be used to provide adequate hints and next steps to continue the student proof. Of course, it is possible that the new DAM does not contain a solution path that uses the partial solution of the student. In that case we can give a warning, and a hint or next step will ignore steps that are not recognized by LOGAX. Note that LOGAX does not allow students to add incorrect steps. Hence, the final proof may contain lines that are not used, but it cannot be an incorrect proof.

The DAM contains different solution paths comparable to the way behavior graphs in example-based tutors code solutions [1]. The main differences are that:

– the DAM is generated automatically;
– feedback and hints are not coded in the DAM;
– DAMs are not static, we generate a new DAM when a student follows a different solution path.

5. BUGGY RULES

Students make mistakes in axiomatic proofs. From the homework of students participating in our course we collected a set of mistakes, and classified these mistakes in three categories:

– oversights,
– conceptual errors, and
– 'creative' rule adaptations.

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

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

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. 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 [7].

LOGAX contains a dialog box to add new proof lines. The idea behind this approach is that a student can concentrate on proof construction. This also means that students can make fewer mistakes. The only possible oversights are syntactic mistakes in formulae. The example given before of a conceptual error is impossible to construct in LOGAX, since LOGAX fills in the assumptions automatically. However, it is still possible that a student tries to apply a rule incorrectly, for example an application of Modus Ponens on $\Delta \vdash \phi$ and $\Sigma \vdash \phi' \to \psi$ 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. With these rules it is possible to give informative feedback. For example, if a student wants to complete a proof

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

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.

6. EVALUATION

We evaluate the proofs generated by LOGAX in two ways. First, we compare the generated proofs with expert proofs. There are some example proofs in textbooks, and there is a large collection of proofs on the Metamath website[1]. We cannot use the latter proofs directly, since the Metamath proofs do not use the deduction theorem, and the same holds for most of the example proofs in textbooks. Therefore we perform a more indirect comparison. We use the constructive proof of the deduction theorem to transform proofs generated by LOGAX in proofs without applications of the deduction theorem, and compare these proofs with the Metamath proofs. Since Metamath proofs build on each other, we rewrite Metamath proofs such that proofs of used lemmas are inlined, and subsequently delete unused or duplicated proof lines. For our evaluation we compare 30 proofs. The most important observation is that almost all proofs are equal up to the order of the lines. In three cases the proofs constructed by LOGAX are shorter than the Metamath proofs, and the other proofs are equally long.

In a second evaluation we investigate whether or not correct student solutions can be recognized by LOGAX. We use solutions of two homework exercises, and two exam exercises. In the first exam exercise, students have to prove that $\neg\neg p \to \neg q, r \to q \vdash r \to \neg p$. 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 lines. Solutions in the first group contain an application of axioms 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 DAM. Still, the tool can provide feedback for students who introduce Axiom b, since in this case we can dynamically generate a DAM that does contain this solution. 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 $\Delta' \cup \Delta'' \subset \Delta$, then add an instance $(\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$ of Axiom b to the DAM.

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

Table 1: Recognized solutions

Exercise	preferred	non pref.	dynamic	total
Exam 1	16		3	19
Exam 2	4		2	6
Homework 1	1		16	17
Homework 2	2	13		15

In the second exam exercise, students have to prove $p \to (\neg q \to \neg r), p \to r \vdash p \to q$. Since this is a resit exam, the number of participants is much lower. Only six students handed in a correct solution to this exercise, and again the solutions are divided into two groups, one using Axiom b (two solutions), and one not using this axiom (four solutions). The solution generated by LOGAX belongs to the second group. Solutions of the first group are not recognized by the DAM that is generated at the start of the exercise, but can be recognized by a dynamically generated DAM. The first homework exercise resembles this examination 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. Here LOGAX generates this last less followed path. 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 used Axiom a and the deduction theorem, and 2 students used an extra application of the deduction theorem instead of Axiom a. In this case the preferred solution by LOGAX is the solution without use of Axiom a, but the DAM also contains a solution with Axiom a. We summarize the results in Table 1. The first column (preferred) shows the number of solutions which corresponds to the preferred solution of LOGAX the second the number which 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 some extra heuristics.

7. CONCLUSION AND FUTURE WORK

By using an existing algorithm for natural deduction, we developed a correct 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 provide hints and next steps, and to extract model solutions. The comparison of our proofs with expert solutions shows that the quality of the generated 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 solution, LOGAX can still provide hints and next steps. We derived a set of buggy rules from a set of

student solutions, and added these to LOGAX. Comparison with a test set showed that this set covers the majority of student errors.

We have not yet evaluated LOGAX in class. In the fall of 2016 we will perform a first evaluation with students. We will measure student satisfaction with LOGAX, and use logging in combination with pre-tests and post-tests to analyze their learning. Special attention will be paid to the mistakes that students make in the post-test. We hypothesize, based on repair theory, that since LOGAX assists students in constructing proofs, creative errors will occur less frequently, since students do not construct their own repairs when they meet an impasse. However, students can still make mistakes, and it might be the case that they make more mistakes in, for example, writing down the conclusion of an application of Modus Ponens. Since LOGAX adds assumptions automatically, students might make errors in adding assumptions if they do not use LOGAX. Future evaluations will show whether or not this indeed happens.

Typically, axiomatic proofs build on each other: for example, a proof of $\vdash \neg\neg p \to p$ can be used as a lemma in another proof. In the future we will extend our tool with the possibility to add lemmas and extra rules. This will also make it possible to provide a greater and more diverse collection of relatively simple exercises.

8. 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. We thank our anonymous referees for their constructive comments.

9. REFERENCES

[1] V. Aleven, B. M. Mclaren, J. Sewall, and K. R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal on Artificial Intelligence in Education*, 19(2):105–154, 2009.

[2] Arun-Kumar. Introduction to logic for computer science, 2002. Retrieved from http://www.cse.iitd.ernet.in/~sak/courses/ilcs/logic.pdf.

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

[4] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer Science & Business Media, 3rd edition, 2012.

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

[6] K. Broda, J. Ma, G. Sinnadurai, and A. Summers. Friendly e-tutor for natural deduction. In *Proceedings TFM'06: the Conference on Teaching Formal Methods: Practice and Experience*, page 2, 2006.

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

[8] H. Enderton and H. Enderton. *A Mathematical Introduction to Logic*. Elsevier Science, 2001.

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

[10] F. F. P. Galafassi, A. V. Santos, R. K. Peres, R. M. Vicari, and J. C. Gluz. Multi-plataform interface to an ITS of proposicional logic teaching. In J. Bajo et al., editor, *Proceedings PAAMS'15: Highlights of Practical Applications of Agents, Multi-Agent Systems, and Sustainability*, volume 524 of *Communications in Computer and Information Science*, pages 309–319. Springer, 2015.

[11] C. Gottschall. The gateway to logic. See https://logik.phl.univie.ac.at/~chris/gateway/formular-uk.html.

[12] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[13] L. Kalmár. Über die axiomatisierbarkeit des aussagenkalküls. *Acta scientiarum mathematicarum*, 7:222–243, 1935.

[14] J. Kelly. *The essence of logic*. The essence of computing series. Prentice Hall, 1997.

[15] C. Leary and L. Kristiansen. *A Friendly Introduction to Mathematical Logic*. SUNY Geneseo, 2015.

[16] J. Lodder, B. Heeren, and J. Jeuring. A pilot study of the use of logex, lessons learned. In *Proceedings TTL'15: the Fourth International Conference on Tools for Teaching Logic*, 2015. CoRR abs/1507.03671.

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

[18] N. D. Megill. *Metamath: A Computer Language for Pure Mathematics*. Lulu Press, Morrisville, North Carolina, 2007.

[19] E. Mendelson. *Introduction to Mathematical Logic*. Discrete Mathematics and Its Applications. CRC Press, sixth edition, 2015.

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

[21] Y. Nievergelt. *Foundations of Logic and Mathematics: Applications to Computer Science and Cryptography*. Birkhäuser Boston, 2002.

[22] D. Perkins. Strategic proof tutoring in logic. Master's thesis, Carnegie Mellon, 2007.

[23] W. Sieg. The apros project: Strategic thinking & computational logic. *Logic Journal of the IGPL*, 15(4):359–368, 2007.

[24] J. van Benthem. *Logica voor informatica*. Pearson Education Benelux B.V., 2003.

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