The Sequent Calculus Trainer – Helping Students to Correctly Construct Proofs

Arno Ehle, Norbert Hundeshagen, and Martin Lange

School of Electrical Engineering and Computer Science
University of Kassel, Germany

Abstract

We present the Sequent Calculus Trainer, a tool that supports students in learning how to correctly construct proofs in the sequent calculus for first-order logic with equality. It is a proof assistant fostering the understanding of all the syntactic principles that need to be obeyed in constructing correct proofs. It does not provide any help in finding good proof strategies. Instead it aims at understanding the sequent calculus on a lower syntactic level that needs to be mastered before one can consider proof strategies. Its main feature is a proper feedback system embedded in a graphical user interface.

We also report on some empirical findings that indicate how the Sequent Calculus Trainer can improve the students' success in learning sequent calculus for full first-order logic.

1 Introduction

Many courses in theoretical computer science suffer from high failure rates and it is common among students to alienate themselves from such courses, c.f. [9, 11]. The reasons are manifold and vary from the way mathematics is taught in school, to lack of general problem solving competences, and not least, to a diversity in skills for adapting new knowledge.

The BSc computer science curriculum at the University of Kassel contains a 2nd-year mandatory course on logic in computer science, as it can be found in typical computer science university programs. This course has been re-organised in recent years with the aim of improving learning outcomes and therefore reducing failure rates. A central point of this re-organisation is the use of constructivistic learning theory, in particular the inverted-classroom model (c.f. [8]). This model focuses on learning, literally, as a self-organised activity; consequently, the logic course should engage students with methods and tools to assist and self-assess the use of formal logic and the calculi taught with them. One of these tools, developed for such purposes, is the *Sequent Calculus Trainer* whose design and use will be described in this paper.

We start by explaining typical problems that students encounter when being faced with a standard exercise in learning Gentzen's sequent calculus [5]: to find a proof for a given sequent, formally expressing that some formula is a logical consequence of a set of formulas. We assume the reader to be entirely familiar with first-order logic with equality [3] and proof calculi in general. Familiarity with sequent calculus in particular is not strictly necessary to follow those explanations; the principles should become clear from the examples we use. We give a brief description of the Sequent Calculus Trainer and explain its aims. At last we provide some empirical data that supports the claim that the Sequent Calculus Trainer can effectively aid the learning of the sequent calculus for first-order logic.

Université de Rennes 1
LIPICS Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)

$$\frac{\Gamma, \varphi, \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta} \qquad \frac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$$

$$\frac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi, \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi, \varphi, \Delta}{\Gamma \Longrightarrow \varphi, \Delta} \qquad \frac{\Gamma, \varphi, \varphi \Longrightarrow \Delta}{\Gamma, \varphi \Longrightarrow \Delta}$$

$$\frac{\Gamma, \psi \Longrightarrow \Delta \qquad \Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \varphi \to \psi \Longrightarrow \Delta} \qquad \frac{\Gamma, \varphi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \to \psi, \Delta} \qquad \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta} \qquad \frac{}{\Gamma \Longrightarrow s = s, \Delta}$$

$$\frac{\Gamma, \varphi[c/x] \Longrightarrow \Delta}{\Gamma, \exists x\, \varphi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi[t/x], \Delta}{\Gamma \Longrightarrow \exists x\, \varphi, \Delta} \qquad \frac{\Gamma, \varphi[t/x] \Longrightarrow \Delta}{\Gamma, \forall x\, \varphi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi[c/x], \Delta}{\Gamma \Longrightarrow \forall x\, \varphi, \Delta}$$

$$\frac{\Gamma, s = s \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta} \qquad \frac{\Gamma, \varphi[s'/x] \Longrightarrow \Delta}{\Gamma, s = s', \varphi[s/x] \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \varphi[s'/x], \Delta}{\Gamma, s = s' \Longrightarrow \varphi[s/x], \Delta}$$

Figure 1 The proof rules of the sequent calculus.

2 The Sequent Calculus

2.1 The Proof Rules

The sequent calculus is a system of proof rules that operate on *sequents* which are pairs of multi-sets of formulas, written $\Gamma \Longrightarrow \Delta$. The intended meaning of such a sequent is that the conjunction over Γ logically implies the disjunction over Δ. The rules listed in Figure 1 operate on formulas in the antecedent Γ or succedent Δ of the rule's conclusion below the line, possibly producing new premisses shown above the line. A proof for a sequent $\Gamma \Longrightarrow \Delta$ is, as usual, a finite tree of sequents formed from $\Gamma \Longrightarrow \Delta$ at its root by successively applying these rules. Each branch must end in an instance of an axiom, i.e. a rule with no premisses.

In the rules for quantified formulas, c must always be a fresh constant – called Skolem constant – that does not occur in the conclusion of this rule already; t must be a ground term over the symbols that occur in the conclusion.

2.2 A Didactic Perspective

A standard exercise in sequent calculus asks for a proof of a given sequent, e.g. $\mathcal{S}_0 :=$

$$\forall x \forall y.\ E(x,y) \to x = f(y) \implies \forall x \forall y \forall z.\ E(x,z) \wedge E(y,z) \to x = y\ .$$

Difficulties and mistakes can generally be put into two categories.

(1) The first one is about *constructing a correct proof*: many students are not able to handle formalisms well; often they can barely parse sequents and apply rules correctly. In this example, one has to introduce new names for the universally quantified variables x, y, z in this order and then decompose the Boolean operators on the right side, yielding $\mathcal{S}_1 :=$

$$\forall x \forall y.\ E(x,y) \to x = f(y), E(a,c), E(b,c) \implies a = b\ .$$

Typical mistakes at this syntactic level are concerned with wrong rule applications and include

- *confusing* rules, for instance applying the rule for conjunctions to a disjunction;
- *misplacing* rules, usually by applying a rule to a genuine subformula rather than a formula in the sequent; in other words not understanding the structure of a sequent;

- *wrong first-order instantiations*, for instance not choosing a fresh Skolem constant when needed;
- *wrong rule instantiations*, for instance by adding the symbols Γ and Δ to the sequent at hand;

and so on.

(2) There are other ways of applying rules in a syntactically correct way in this example, for instance by operating on the formulas in the premiss instead. This, however, is unwise for *finding the right proof*. For this one must proceed as described above and then have the inspiration to double one of the formulas in the premiss and instantiate both copies differently yielding

$$E(a,c) \to a = f(c), E(b,c) \to b = f(c), E(a,c), E(b,c) \implies a = b \ .$$

The rest of this proof task is relatively easy using Boolean elimination rules and very simple reasoning about equalities.

There is a clear dependency between these two challenges: the ones described in (1) need to be met before those in (2) can be met; it is impossible to find a proof unless one is able to construct correct proofs at all. The latter is clearly a very difficult task for students who already struggle with uncertainties like "am I allowed to apply this rule here?", "was the application correct?", "should I introduce a new name or instantiate with an already existing term?", etc.

This gap between syntactical and semantical understanding has been addressed in many fields like teaching programming or simply mathematics (e.g. in [10]). This phenomenon is accurately described in [4] as the ability to "write rather rigorously a simple C program", while they cannot "rigorously write down a mathematical proof of the kind needed in graph theory, formal logic, [...]" There is a hidden hint in this observation on how to tackle the problem of teaching proof calculi. Students seem to easily understand syntactic principles as long as there is a mechanism – like a compiler – which allows them to learn the formalism in a trial-and-error way.

This is where the Sequent Calculus Trainer comes into play. It is supposed to aid the constructing of correct proofs but does not help at all in finding the right proofs. From a logical point of view it is merely a proof assistant, not a theorem prover. From a didactic point of view, its use is supposed to address the students' rote memory such that they become able to reach this gap between syntax and semantics, thus enabling them to start understanding the underlying logical concepts. The Sequent Calculus Trainer therefore adheres to two design principles: it is an easy-to-use and simple assistant for building proof trees in sequent calculus. Moreover, it comes with a compiler-like feedback system, which is known for its benefits in tutorial teaching environments [1].

2.3 Related Tools

There are other high quality interactive proof systems that can be used to train the construction of correct proofs in the sequent calculus. The tool that meets the prerequisites laid out here best is LOGITEXT[1]; others worth mentioning are JAPE [2] and PANDA [4]. None of these treats first-order logic *with* equality, though. This is a major drawback since equality is – possibly together with quantification – one of the most difficult concepts for constructing

[1] `http://logitext.mit.edu`

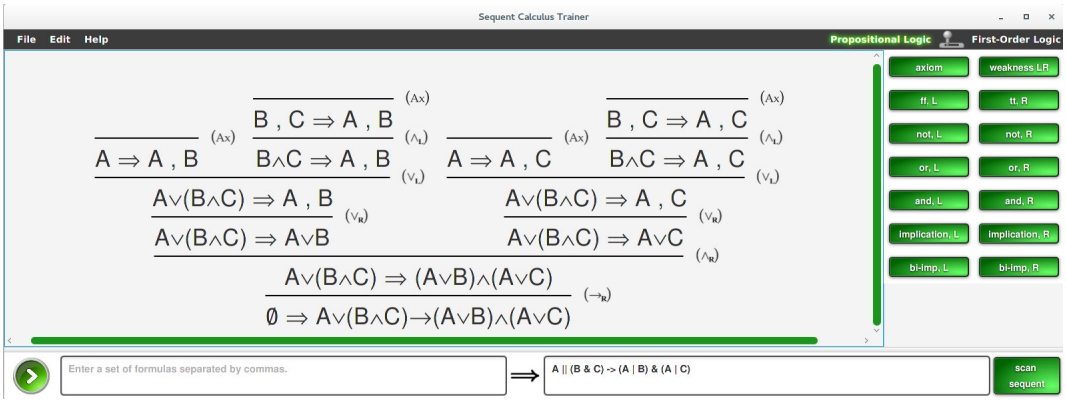

Figure 2 The user-interface.

and finding proofs. Yet equality reasoning is ubiquitous in computer science; hence, it should not be ignored in teaching contexts. Furthermore, JAPE is lacking a feedback system, and PANDA's proof calculus is natural deduction which differs from the sequent calculus.

In a setting where we measure success through the understanding of syntactic concepts, it is essential that the proof calculus used in classroom must be the same as that used by a supporting tool. None of those existing tools is general enough to serve the purposes described above – to act as a tool supporting the learning of proof construction in the sequent calculus for first-order logic with equality. One also should not underestimate the effort that would be needed in order to extend or amend an existing software tool created by others. Hence, having many tools with slightly differing features in this area should be considered advantageous.

3 The Sequent Calculus Trainer

We provide the Sequent Calculus Trainer as an open source application under the BSD-3 license and the source code as well as the binaries are publicly available[2]. Figure 2 shows the graphical user interface which is kept fairly simple. The trainer is shipped with two main views, one for propositional logic and one for first-order logic. They both differ only in the number of applicable rules shown on the right side of the windows, and in the treatment of atomic propositions, which are interpreted as 0-ary predicates in the first-order logic view. Sequents can be input either through a simple text file or in the text fields at the bottom, where the syntax specification for the input is given, too. Furthermore, it is possible to save and load proof trees in an internal format as well as export them in PNG format.

3.1 Key Features

We briefly introduce the two key features of the Sequent Calculus Trainer.

- Nearly every user action leads to a response by the program. Figure 3 exemplarily illustrates such on-screen messages. Each rule button is equipped with a short message, which occurs on mouse-over. These messages usually contain the formal definition of a

[2] `http://www.uni-kassel.de/eecs/fachgebiete/fmv/projects/sequent-calculus-trainer.html`

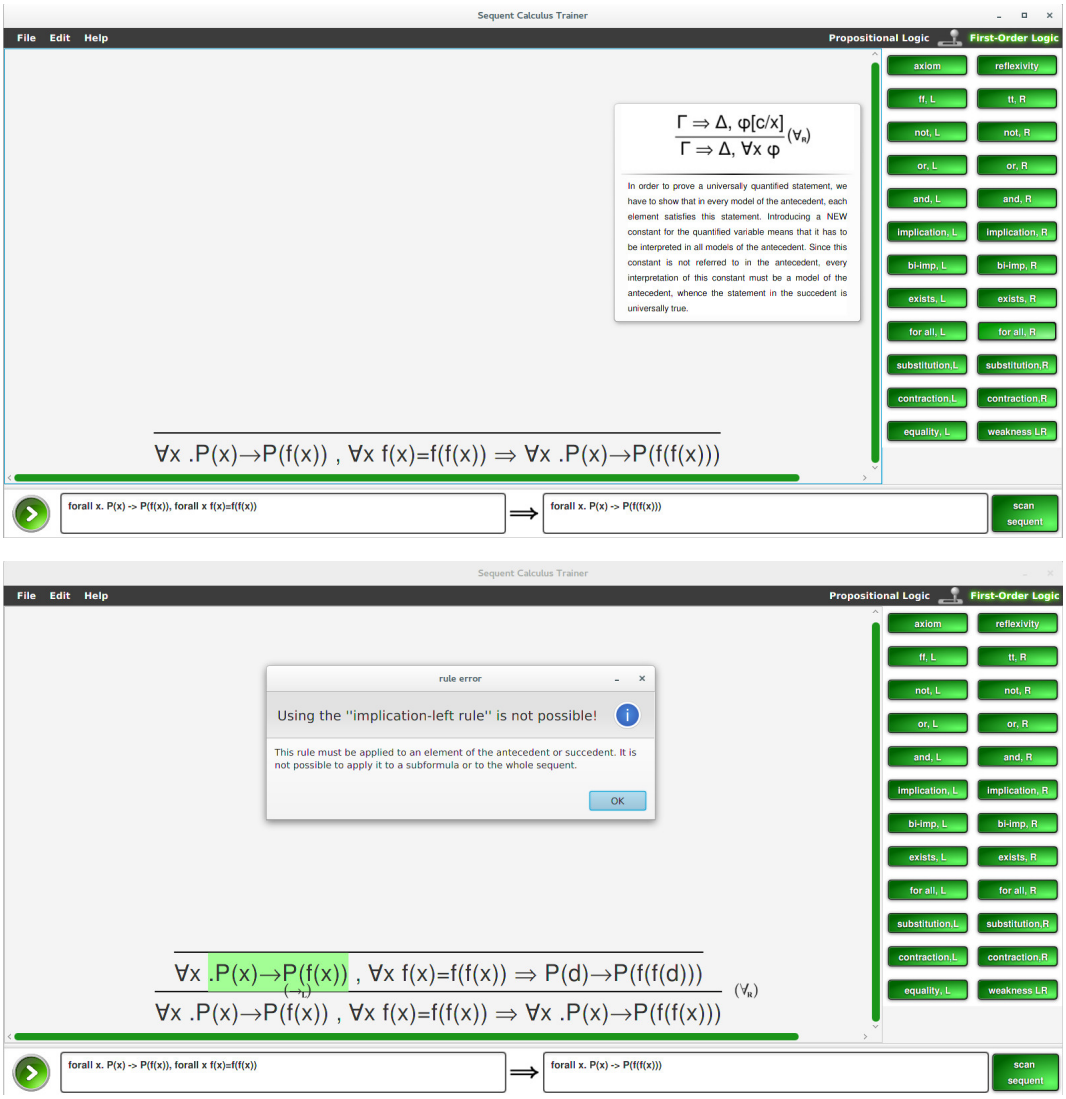

Figure 3 The feedback system.

rule as well as some appropriate high-level explanations of the rule's meaning and, if suitable, why it is a valid logical principle.

If the user has chosen a rule and tries to apply it to a formula by selecting a logical operator, the formula represented by this operator "responds" by telling the user whether or not the rule is applicable there. This happens in two ways: the part of the formula that is in scope of the selected operator or symbol is highlighted. This helps to understand precedence rules and the structure of sequents and formulas. When a wrong operator or symbol is chosen, the user is provided with an error message which includes a hint on the mistake. For instance, the reason why a current leaf in the prooftree is an axiom has to be identified via clicking on the part of the formula that causes the application of an axiom rule.

The second notable feature is the handling of sequents that include equalities. Figure 4 shows how the substitution rule works. After the rule for substitution on the left-hand or

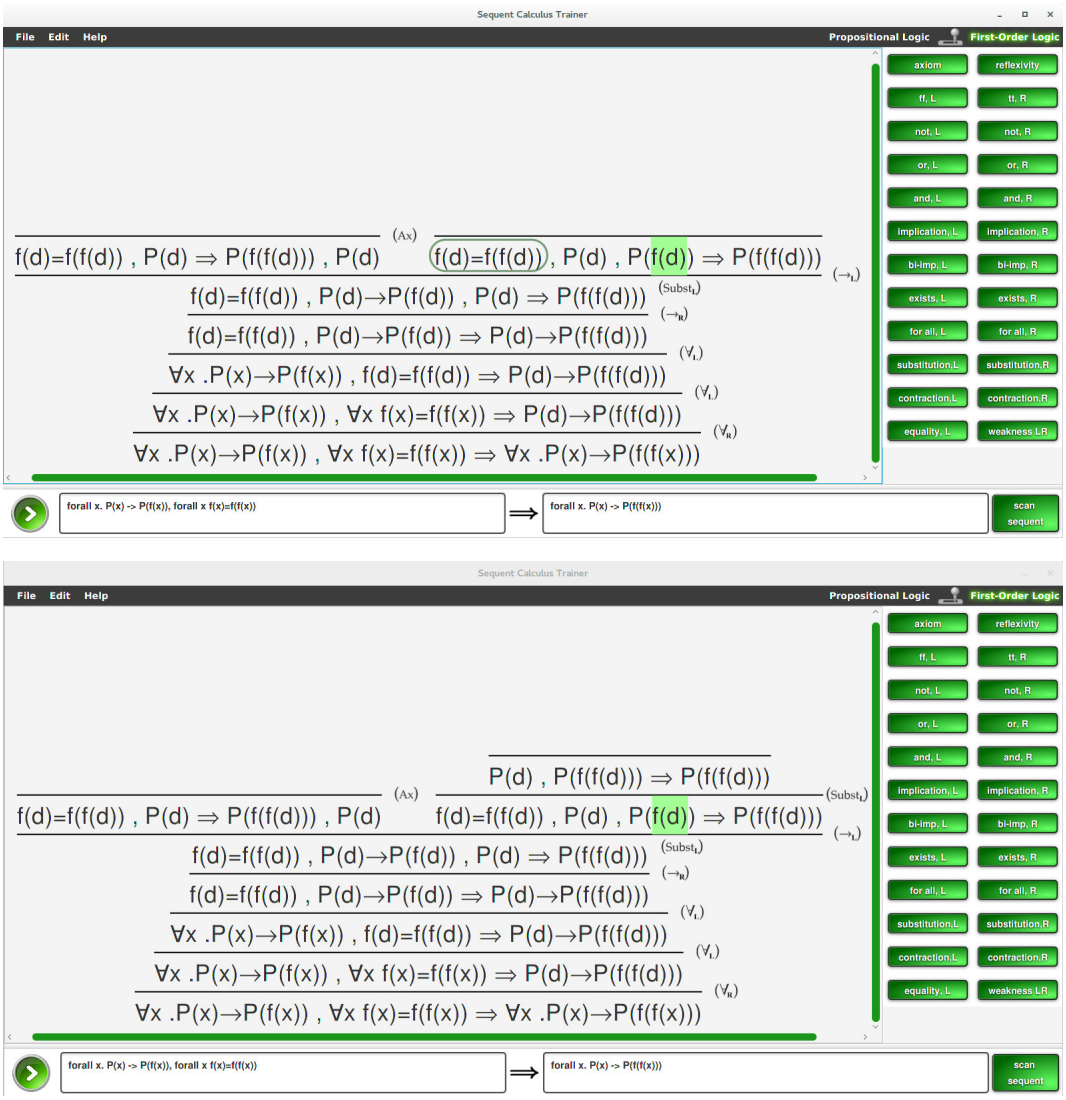

Figure 4 The substitution rule.

right-hand side of the sequent has been chosen, the program expects an atomic formula with an equality predicate to be selected. In the last step, the term that should be substituted needs to be clicked on.

A final point worth mentioning is that the user is able to undo all steps in the proof up to a certain sequent at any time by just applying a different rule to that particular sequent.

3.2 The Backend

The Sequent Calculus Trainer is meant to be used by a broad range of students; hence, it platform independency provided by an implementation in Java is a key design principle. It uses the GUI framework JavaFX, which is integrated in the Java Standard Library since the emerging of Oracle's Java 8. This mainly allows the program to have only one dependency in ControlsFX that is a small extension library for JavaFX, designed to give even more UI

controls and simple dialogs. The result is a clear and comprehensive feedback system which was another key design principle.

The Sequent Calculus Trainer is designed according to the principles provided in the model-view-controller-patterns (MVC). Thus, it consists of three parts; the data model which contains the data and algorithms as a standalone part of the program, the controllers which contain the logic for the GUI, and finally the GUI itself which is build with JavaFX while mainly using the XML based notation for GUI elements called FXML. The program is equipped with the possibility to simply add new languages in form of language sets in simple text files and new country flags for the frontend. Currently German and English are available.

4 Experiences in Teaching the Sequent Calculus

The computer science BSc curriculum at the University of Kassel contains a mandatory 2nd-year course on logic which teaches, amongst others, the sequent calculus for first-order logic with equality. In order to pass the course students need to take a written exam. We report on the results achieved in three successive years. Group 1 took the exam in winter term 2012/13. The Sequent Calculus Trainer was developed afterwards, so it was not available to them at the time of preparation for the exam. Group 2 took the exam in winter term 2013/14, and they were greatly encouraged to solve corresponding exercises using the Sequent Calculus Trainer. Group 3 took the exam in winter term 2014/15. The Sequent Calculus Trainer was made available, but in comparison to group 2 the trainer had been advertised less. The preconditions are comparable in the sense that before the exam, all groups had to pass two graded exercises on sequent calculus, they were provided with the same number of additional voluntary tasks on this topic, and they were allowed to prepare a handwritten note for the exam which contained the sequent calculus rules in many cases.

The three exams under consideration featured a question each, asking for a proof of a given sequent. The three sequents differed, with the ones used for group 2 and 3 being seemingly more difficult to prove with regards to both aspects of constructing and finding a proof.

$$\begin{array}{ll} \text{group 1:} & \forall x \exists y.\ x = v(y) \land \forall x\ F(v(x)) \implies \forall x\ F(x) \\ \text{group 2:} & \forall x \forall y.\ E(x,y) \to x = f(y) \implies \forall x \forall y \forall z.\ E(x,z) \land E(y,z) \to x = y \\ \text{group 3:} & \forall x \forall z.\ P(x,c) \land Q(z, g(x,z)) \implies \exists y \forall x\ P(x,y) \land \forall z \exists u\ Q(z,u) \end{array}$$

While the sequents for groups 2 and 3 need some insight into the properties expressed by its formulas, the sequent for group 1 can be proved using almost syntactic considerations only. Surprisingly, out of 70 students in group 1, more than 50% (46 in total) were not able to apply such a simple strategy of applying all possible rules in the correct order. Moreover, when adding the 10 students that did not even try to execute the exercise, we may conclude that 80% of group 1 did not succeed in this exercise because of problems with the correct application of rules. The mistakes most frequently made were *misplacing* of rules and *wrong first-order instantiations* according to the classification listed in Section 2.2. Some examples of typical mistakes are shown in Figure 5.

Group 2 shows a totally different picture. Out of 51 students only 12 already failed in simple rule applications, where 1 student did not execute the exercise, which leads to 25% in total. Thus, nearly 75% of the students where at least able to handle the syntactic formalism resulting in correct rule applications before a deeper understanding would be needed.

The most recent data is taken from group 3. Out of 46 students 19 failed to achieve the mentioned goal and 2 did not execute the exercise which is 46% in total.

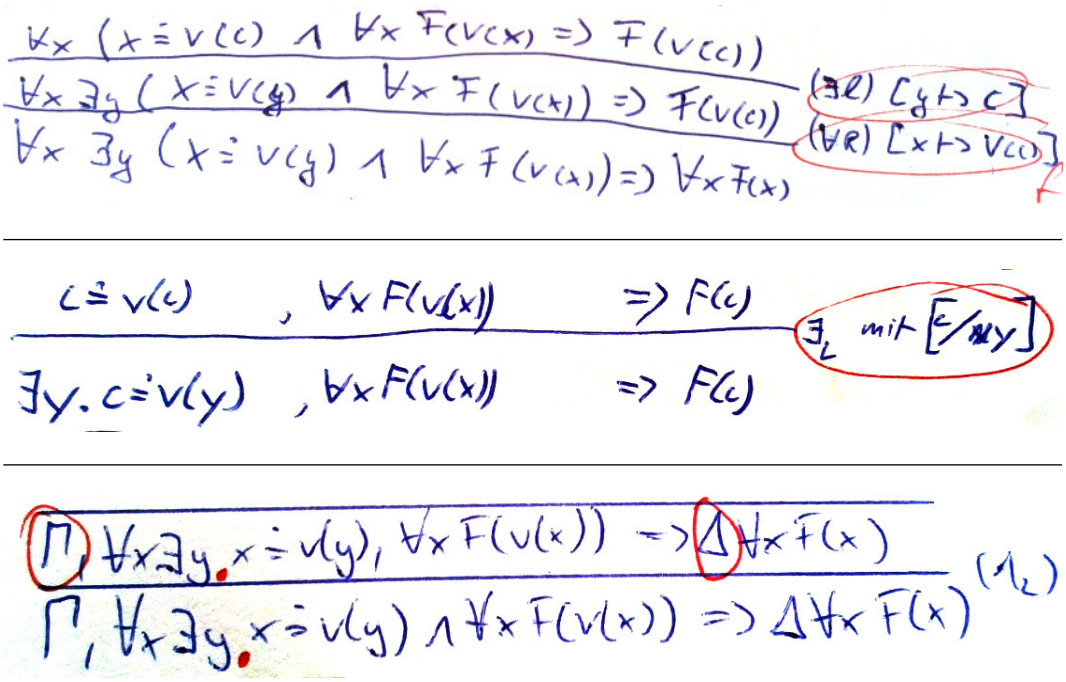

Figure 5 Examples of frequently made mistakes with wrong rule applications.

If we summarize the results of the latter two groups we get a rate of 34 students out of 97 which failed for reasons of wrong application of rules. Thus, nearly 65% of the students where at least able to handle the syntactic formalism resulting in correct rule applications. We believe that this is due to the availability of the Sequent Calculus Trainer.

This conclusion seems to be in contrast to two other – seemingly odd – observations. First of all, group 2 did not achieve significantly more points than group 1. Both exercises were graded with 4 points in total where group 1 reached an average of 1.2 points and group 2 an average of 1.5 points. As mentioned above, from a semantical perspective the sequent for group 2 is harder to proof and, therefore, needs a more involved strategy. This is the main reason for the only slight improvement in the average grade; a syntactically correct but unsuccessful proof attempt is not graded with more than 2 out of 4 points. Hence, the introduction of the Sequent Calculus Trainer into the process of learning how to correctly apply rules resulted in a 25%-gain in points between these two groups. The still low absolute average value achieved by group 2 points out the lack of "understanding" of the underlying theory which is not addressed by the use of the Sequent Calculus Trainer.

Secondly, the results of group 3 seem to worsen in comparison to group 2. Again, the exercise on sequent calculus of group 3 was graded under the same constraints with 4 points in total. The effort needed to prove this sequent is comparable to the effort for the sequent of group 2, as both need a similar strategy of replacing the quantified variables in the correct order. However, the outcome is different. In group 3 more students made mistakes in rule applications although they were also provided with the Sequent Calculus Trainer. They reached an average of 2.7 points. One explanation of this observation can be found when comparing the mistakes made in rule applications. Out of 19 students who tried to solve the exercise and failed in rule applications 7 made the same single mistake of misreading the precedence of the conjunction in the succedent, exemplarily shown in Figure 6. Such "near"

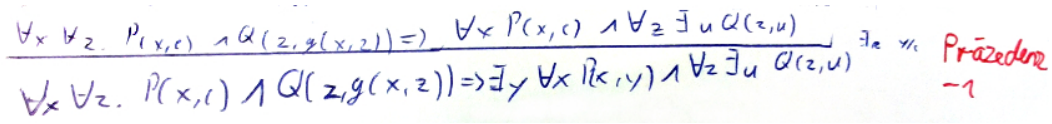

Figure 6 Example of the most frequent mistake in group 3.

solutions were graded with 3 out of 4 points.

5 Discussion and Future Work

The improvement in the exam results of the logic course at the University of Kassel in winter term 2013/14 and 2014/15 compared against those achieved in winter term 2012/13 correlate to the availability and encouragement to actively make use of the Sequent Calculus Trainer. Clearly, the improvement of results in one particular course can be caused by various reasons; it is well known that the teaching staff has the greatest impact on learning outcomes (cf. [7]). In our setting the lectures for all groups were given by the same lecturer and the main part of the exercises was given by the same teaching assistant. The role of the group's composition and the educational background of the students is of course not to be underestimated. However, the significance of the measured effect was too great to be caused solely by the aforementioned reasons. This is strongly emphasised by the fact that the differences in the total outcome of the exams are marginal. Nevertheless, it would obviously be interesting to support this further by a broader empirical evaluation, in particular by considering its effects on students at different universities.

When regarding the effect more deeply, a software for teaching sequent calculus used as an interactive compiler for the language of proof trees perfectly meets the requirements given in Section 2.2 for closing the syntactic gap that may be present in students' minds when introducing a new formal concept. From a different point of view such a piece of software offers a suitable alternative in addressing the rote memory to replace just the right amount of pen and paper exercises, needed to understand the formalism. In addition, it directly forbids mistakes that would be possible to make with pen and paper and would have to be manually marked by human teaching assistants.

Clearly, the goal in teaching a calculus for propositional or first-order logic is not just about the simple manipulation of strings. Instead students need to learn to fluently understand the properties being expressed by logical formulas, to visualise the classes of structures that are being represented by them, etc. In other words, students also need to understand the semantics of logical languages and calculi. The Sequent Calculus Trainer is not meant to address possible deficits in understanding semantics, nor does it do it automatically as the considerations at the end of Section 4 show. We do believe, though, that similar improvement in learning success could be achieved for such semantical aspects by complementing the Sequent Calculus Trainer with a tool that trains the understanding of semantics, for instance using model checking games for first-order logic [6].

Acknowledgements

We would like to thank Angelika Hoffman-Hesse for doing the statistical evaluation of which mistakes were made by how many students as reported above, and Florian Bruse for acting as a linguistic consultant in the development of the Sequent Calculus Trainer. The work has

been financially supported by the *Service Center Lehre* of the University of Kassel under an *Innovation-in-Teaching* grant.

References

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

2 R. Bornat and B. Sufrin. Animating formal proof at the surface: The jape proof calculator. *Comput. J.*, 42(3):177–192, 1999.

3 H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer, Berlin, 2nd edition, 1994.

4 O. Gasquet, F. Schwarzentruber, and M. Strecker. Panda: A proof assistant in natural deduction for all. A Gentzen style proof assistant for undergraduate students. In *Proc. 3rd Int. Congress on Tools for Teaching Logic, TICTTL 2011*, volume 6680 of *LNCS*, pages 85–92. Springer, 2011.

5 G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

6 E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

7 J. Hattie. *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement*. Taylor & Francis, 2008.

8 M. J. Lage, G. J. Platt, and M. Treglia. Inverting the classroom: A gateway to creating an inclusive learning environment. *J. of Economic Education*, 31(1):pp. 30–43, 2000.

9 Gabriel Robins. Teaching theoretical computer science at the undergraduate level: Experiences, observations, and proposals to improve the status quo. Technical report, Computer Science Department, UCLA, 1988.

10 B. Shneiderman and R. E. Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *Int. J. of Parallel Programming*, 8(3):219–238, 1979.

11 David R. Surma. Rolling the dice on theory: One department's decision to strengthen the theoretical computing aspect of their curriculum. *J. Comput. Sci. Coll.*, 28(1):74–80, 2012.