LMU
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHR- UND FORSCHUNGSEINHEIT FÜR PROGRAMMIER- UND MODELLIERUNGSSPRACHEN
Institut für Informatik

Conception, Implementation and Evaluation of Proof Editors for Learning

Korbinian Staudacher

Bachelorarbeit

Beginn der Arbeit:	20.07.2018
Abgabe der Arbeit:	18.09.2018
Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Prof. Dr. François Bry
	Sebastian Mader

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

München, den 18.09.2018 Korbinian Staudacher

Abstract

At the beginning of computer science studies, students often have problems understanding the abstract way of reasoning of formal methods. To make it easier for students to learn formal methods, editors for the logical proof methods Resolution and Natural Deduction have been developed. Both editors allow users to develop step by step a proof tree for logical reasoning, where partial aspects of the proof, such as the application of formal rules, are already executed by the program. The editors are designed to help students understand the methods of Resolution and Natural Deduction as well as more generally the forms of reasoning of formal methods. In order to make the learning process as easy as possible, the editors have been designed following the material provided in the lecture 'Logic and Discrete Mathematics' which introduces students to logical proofs at the beginning of their studies. In addition, users in the editors receive support through the didactic methods scaffolding and feedback. The editors are presented and the usefulness of these are evaluated based on students' performance. Finally, an outlook on possible enhancements and improvements is given.

Zusammenfassung

Zu Beginn eines Informatikstudiums haben Studenten oftmals Probleme die abstrakte Denkweise formaler Methoden zu verstehen. Um Studenten das Erlernen formaler Methoden zu vereinfachen wurden Editoren für die logischen Beweistechniken Resolution und Natürliches Schließen entwickelt. Beide Editoren ermöglichen es Benutzern Schritt für Schritt einen Beweisbaum für logische Schlüsse zu entwickeln, wobei Teilaspekte des Beweises, wie zum Beispiel die Anwendung formaler Regeln, bereits durch das Programm ausgeführt werden. Studenten soll dadurch sowohl das Verständnis der Beweismethoden Resolution und Natürliches Schließen erleichtert werden, als auch das allgemeine Verständnis formaler Methoden. Um den Lernprozess so einfach wie möglich zu gestalten, wurden die Editoren in Anlehnung an das in der Vorlesung "Logik und diskrete Strukturen" bereitgestellte Material konzipiert, da diese Vorlesung Studenten zu Beginn ihres Studiums in logische Beweise einführt. Zusätzlich erhalten Benutzer in den Editoren Unterstützung durch die Lehrmethoden Scaffolding und Feedback. Die Editoren werden präsentiert und die Nützlichkeit dieser wird anhand studentischer Leistungen evaluiert. Abschließend wird ein Ausblick auf mögliche Erweiterungen und Verbesserungen gegeben.

Acknowledgement

The author would like to thank Sebastian Mader for supervising this thesis, his continuous help and for embedding the editors into Backstage. He would also like to thank Professor François Bry for his useful suggestions and motivating discussions.

Contents

Chapter 1

Introduction

At the beginning of a computer science degree course, students often face major challenges adapting themselves to education at university level. While computer science and mathematics are mostly task-based in school education, the focus in university education lies often on how to solve general and abstract subjects, such as in computer science *formal methods*. Formal methods are "mathematically based languages, techniques, and tools for specifying and verifying [...] systems" [CW96, p. 626]. They are applied for example in the field of specifications when describing a system and its properties, such as a programming language, and in the field of software verification where the correctness of a program can be established using theorem provers [CW96]. As formal methods usually require that general statements are made about all possible manifestations of a defined subject, the ability of abstract mathematical thinking is absolutely necessary in order to understand and apply formal methods. Due to the mostly task-based education received in school, it is difficult for students in their first terms to adapt this type of abstract thinking. In this work, proof editors for logic have been implemented in order to assist students in understanding formal methods and developing abstract thinking. As students have shown to have significant problems with the proof methods Resolution and Natural Deduction, the focus of this work lies on the conception, implementation and evaluation of proof editors for these two methods. With focus on facilitating the understanding of formal methods, there have been implemented two separate editors for both Resolution and Natural Deduction allowing students to prove logical consequences in zeroth and first order logic by building up a proof tree.

Editors for Resolution and Natural Deduction are not a completely new field of science as several editors can already be found on the world wide web, but none of them is designed for supporting beginners. While most editors for Resolution so far available solve a given statement automatically, which

does not serve the learning purpose, most editors for Natural Deduction so far available require much knowledge and use a linear syntax rather than a tree syntax, which usually is easier to understand for beginners. The editors in this work were designed with the purpose to ease learning of Resolution and Natural Deduction as much as possible. This is accomplished by using a simple tree syntax and by making use of the didactic methods feedback and scaffolding. Students should be able to enhance their knowledge of formal methods independently from help by tutors, professors or other persons by solving exercises with the proof editors. Yet the aim of these editors is not only the facilitation of learning the principles of Resolution and Natural Deduction, but also the enhancement of understanding formal methods in general. For evaluation purpose, the editors were implemented in the classroom communication system "Backstage", were the editors were used as part of the exam preparation for the course "Logic and discrete Mathematics". This course usually takes place in the second term of a computer science student and introduces the methods of Resolution and Natural Deduction to students. The following chapters first give an overview of the didactic methods feedback and scaffolding and the research which has already been made in these areas, then the methods of Resolution and Natural Deduction are formally introduced and the designed editors are presented. Different aspects of how feedback and scaffolding that have been taken under consideration are explained and finally the usefulness of the editors is evaluated using the results of 200 students, who participated using the editors for their exam preparation in "Backstage".

Chapter 2

Related Work

As the whole design of the proof editors is built upon the didactic methods feedback and scaffolding, the research already made in these areas is presented in this chapter.

2.1 Feedback

Feedback, especially in the context of education has been the subject of much research regarding its effectiveness in various situations. Feedback can be defined as "actions taken by (an) external agent(s) to provide information regarding some aspect(s) of one's task performance" [KD96, p. 255]. Thus, feedback can be seen as a "consequence of performance" [HT07, p. 81]. In relation to instructions, feedback can either be clearly separated from a previous instruction or even fulfill the form of a new instruction [Kul77]. Therefore, feedback and instruction can be considered a continuum, although there must be an initial instruction for feedback to have an effect [HT07]. If the student is completely unfamiliar with the provided material, feedback has no effect, but otherwise it can be a powerful help to students seeking out errors and enhance their learning behavior [HT07]. In a study analyzing more than 100 factors influencing students' achievements, feedback was ranked among the top 10 factors improving learning [HT07]. However, the effects of feedback have led to much discussion, since it is difficult to measure the exact impact of feedback on student's learning behavior. While Kulhavy concludes that a teacher should provide feedback "as often as possible during the course of a lesson" [Kul77, p. 229], in a later work he argues that 'more' feedback does not always equal more learning [KWT+85] and more recent studies like Kluger and DeNisi's meta-analysis of given feedback instructions showed that it can also have no or even negative effects on students' performances [KD96]. Therefore, it is important

to specify how and when feedback should best be given and what types of feedback can improve student performance.

In general, negative feedback has a greater impact than positive feedback [KD96] and if "goals are specific and challenging but task complexity is low" [HT07, p. 85f.], that is to say goals that seem difficult to achieve like learning a programming language are best achieved by approaching the goal with small tasks like writing snippets of code rather than a whole program. Hattie provides a model which discriminates between four levels of feedback [HT07, p. 90]:

- Feedback at task level concentrates on giving information about the specific task to be solved.
- Feedback at process level focuses on how to solve all tasks similar to the given task.
- Feedback at the level of self-regulation is aimed to improve one's abilities regarding self-monitoring, self-evaluation, and directing actions.
- Feedback at self level includes personal evaluation about the student.

Basically, feedback aimed at the task itself or how to solve a problem results being more useful than feedback aimed at the student as person. Such feedback, for instance praise or reward, might even have a negative effect on student performance [HT07]. The reason may be that this type of feedback draws the attention of students away from the task itself towards self-esteem [BW98] which is of small use solving an exercise. Feedback at task level may be powerful when combined with either feedback at process level or at the level of self-regulation, but mostly this type provides answers such as correctness to a specific task and therefore has only effect on the current question and not to other questions related to the same topic [HT07]. Feedback at process level is very powerful as it enhances deep learning and understanding of the whole topic as well as construction of relationships and transference to other tasks [HT07].

Feedback at level of self-regulation is most likely the most powerful level. It can be seen as "an active constructive process whereby learners set goals for their learning and monitor, regulate, and control their cognition, motivation, and behaviour, guided and constrained by their goals and the contextual features of the environment." [PZ02, p. 64]. A study of Orsmond has shown that the capability of self-regulated learning for students in higher education is almost equivalent to being a "high-achieving" student [OM13]. Orsmond also concludes, that this capability is also essential to process feedback effectively in the first place [OM13]. The study revealed that high-achieving students are able to extract essence and significance out of feedback, while

low-achieving students often had a rather atomistic instead of an holistic approach regarding feedback and tried to memorize the feedback rather than understanding its objective [OM13]. This capability is also considered to be necessary for a learner to become less reliant on tutors' regulation and to become a lifelong learner [NMD06].

Regarding the importance of feedback improving self-regulation Nicol and Macfarlane-Dick developed several principles for good feedback practices [NMD06, p. 205]. According to them feedback needs to:

1. help clarify what good performance is such as expected standards and goals;
2. facilitate the development of self-assessment in learning;
3. deliver high quality information to students about their learning;
4. encourage dialogues with teachers and peers;
5. encourage motivation and self esteem;
6. provide opportunities to close the gap between current and desired performance;
7. provide information to teachers that can be used to help shape teaching.

They identify many ways to meet these requirements, which are the most prominent:

- Exemplary solutions can "define a valid standard against which students can compare their work" [NMD06, p. 207] thus are excellent for clarifying good performance.
- Peer Dialogue such as providing feedback on each other's work helps to develop the skills necessary for objective judging against standards. This can be transferred to judge own work.
- Many "easy-to-solve" tasks are better to motivate students than one difficult task, as well as providing students with feedback comments rather than with grades.
- Providing feedback to the teacher as well for example in form of "One-Minute-Papers" with the intention that the teacher is able to identify current problems such as lack of understanding and to adjust the lessons accordingly.

While Hattie considers both feedback at process and self-regulation level to be important, Orsmond concludes, that the form of self-regulated learning might be the only effective way to provide feedback. A tutor-centered feedback model can be of limited use for high-achieving students, because they are able to produce feedback for themselves due to their capability of self-regulation, and low-achieving students might be encouraged to further depend on external regulation as the process of self-assessment does not occur [OM13].

2.2 Scaffolding

Scaffolding is a learning concept based upon the theory of *Zone Proximal Development (ZPD)* of Lev Vygotsky [VdPVB10]. Originally used in the context of child education it is defined as "the distance between the actual development [...] and the level of potential development [...] under adult guidance or in collaboration with more capable peers." [Vyg80, p. 86]. More recent research has transferred the theory of ZPD to general education leading to scaffolding as a means to guide and help learners during their studies [Gib02]. This is achieved through the temporary assistance of another persons helping the student perform tasks, which are in his Zone of Proximal Development, which means he cannot solve the problem alone but with guidance, and gradually remove the support until the student is able to perform the tasks without help [Gib02, VMKK03]. This process allows a student to develop task competence in "[...] a pace that would far outstrip his unassisted efforts" [WBR76, p. 90]. Although there is no exact definition of scaffolding, according to van de Pol there are some general characteristics which classify scaffolding [VdPVB10]. As shown in Figure 2.1 scaffolding consists of:

1. Contingency: The teacher has to determine the current knowledge level of the student via diagnosing strategies based upon student responses and provide support adjusted at the current level.

2. Fading: The gradual withdrawal of scaffolding as the student gathers knowledge.

3. Transfer of Responsibility: In opposition of the gradual withdrawal of scaffolding, responsibility is transferred to the student.

Furthermore, scaffolding strategies can be divided into *means*, which are strategies corresponding to how the scaffolding is done and *intentions*, aimed at what is scaffolded. Typical means are for example giving hints, modelling (demonstrate something), or feedback, while intentions are for example frus-

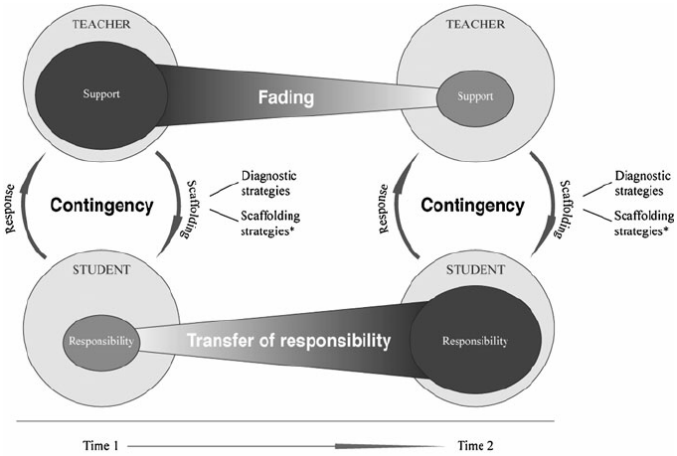

Figure 2.1: Conceptual model of scaffolding [VdPVB10, p. 274]

tration control, aimed at the support of students affect, or cognitive structuring [VdPVB10].

While this model of van de Pol classifies scaffolding itself, Kirschner et al. have proposed a design model for complex learning based on scaffolding [VMKK03]. According to Kirschner, learning new demanding tasks can often be overwhelming to students due to their complexity. As the human mind is limited in its processing capacity, it needs to have as much "free space" as possible to focus on the problem. Therefore he proposes reducing cognitive load during the learning process by using scaffolding [VMKK03]. Regarding scaffolding from the point of reducing cognitive load, given support can either decrease *intrinsic* cognitive load, which is students move gradually from the simplest task towards more complex tasks, or decrease *extraneous* cognitive load, such as starting with worked-out examples, filling out gaps and moving towards conventional tasks [VMKK03]. In both cases, tasks can be divided into *task classes* according to their difficulty. The student then starts with solving tasks in the simplest task class and gradually moves to more complex task classes [VMKK03]. It is considered to be more useful to provide whole tasks than only parts of a task, so that students can gain an overview of the structure and relations between the parts of a task [VMKK03]. Useful types of learning tasks according to Kirschner are for instance worked-out examples as they enable a student to extract general solutions or schemas as well as completion tasks [VMKK03]. In terms of the right timing of information Kirschner divides between *supportive information*, such as cognitive strategies or mental models, which is best presented before the task is processed, and *procedural information*, such as constant task components or repetitive aspects, "that are performed as routines by experts", which is best presented during the task procession [VMKK03].

Considering these aspects while designing tasks for learners should yield a good framework for scaffolding.

Regarding its effectiveness, scaffolding is considered to be very useful in learning processes, although it is difficult to measure [VdPVB10]. Clearly scaffolding can best be performed with a teacher who individually guides the student, but the "scaffolder" does not need to be human and the process of scaffolding can be provided by technology as well [AH05]. Some studies based upon computer scaffolds have shown, that these were also effective for "moving students towards more sophisticated models" as well as increasing the capability of self-regulation [AH05, p. 371].

Chapter 3

Proof Editors

3.1 Resolution Editor

3.1.1 Resolution Principle

Resolution is a proof method using the proof of contradiction to prove the satisfiability of clauses in first order predicate logic [Rob65]. Predicates in first order logic are formed with [Bry15a, Bry15c]:

- Predicate symbols such as $(p, q, r, ...)$ with arity ≥ 0. A predicate symbol with arity 0 is called "propositional variable";
- Function symbols such as $(a, b, c, ...)$ with arity ≥ 0. A function symbol with arity 0 is called "constant";
- Quantifiers such as $\forall$ and $\exists$ with variables (x, y, z) bound to them;
- a set of logical symbols, namely),(, $\perp$,$\top$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Longleftrightarrow$.

A *term* in first order logic is recursively defined as being either a variable or a n-ary function symbol followed by n terms in parentheses and separated by commas, e.g. $t\,(t_1, t_2, ..., t_n)$ [Rob65]. Building on this a *literal* is defined as being a (negated) predicate symbol followed by n terms, such as $p\,(t_1, t_2, ..., t_n)$ and a *clause* as being a disjunction of literals [Rob65]. A literal L is called negative, if the predicate symbol is negated, else it is called positive [Bry15d]. Moreover the complement of a literal $\overline{L}$ is defined as the corresponding negative literal if the literal is positive and vice versa [Bry15d]. For instance the complement of A is $\neg A$ and the complement of $\neg A$ is A. Zeroth order logic can be considered a special case of this definition as the predicate symbols always have arity 0, and therefore there occur neither function symbols nor variables [Bry15c]. Each formula in first, and

especially zeroth order logic is logically equivalent to a formula in *conjunctive normal form*, where the formulas are conjunctions of clauses [Bry15d]. In order to transform a formula to conjunctive normal form the following steps have to be performed:

1. **Rectification**
   A formula is rectificated if the variables bound to a quantifier are pairwise disjoint and if each of them is different from every free variable in the formula [Bry15d].

2. **Transformation to prenex normal form**
   A rectificated formula is transformed into prenex normal form by moving all quantifiers of a formula F to the formula's beginning. A quantifier Q placed at the beginning remains the same if Q appears in F with positive polarity, which is if Q appears in an even amount of negations or left sides of the implication junctor. Otherwise Q has negative polarity, which is if Q appears in F in an odd amount of left sides of the implication junctor or negations, and the quantifier Q will be transformed into its opposite [Bry15d]. For example the formula $(p(a) \Rightarrow (\forall x\, q(x) \Rightarrow \exists y\, r(y)))$ is transformed into $\exists x \exists y\, (p(a) \Rightarrow (q(x) \Rightarrow r(y)))$, because $\forall x\, q(x)$ has negative polarity as it appears within an odd number of left sides of the implication junctor and $\exists y\, r(y)$ has positive polarity because it it appears neither within negations, nor within left side of an implication junctor.

3. **Skolemization**
   The formula is stripped of all $\exists$-quantifications. The following algorithm returns an equivalent satisfiable formula S in skolem form for any given formula F in prenex normal form [Bry15d]:

   - Initialize L' as L being the language containing the used predicate, function and logical symbols and initialize S with F.
   - Repeat the following if S contains an $\exists$-quantifier:
     If S is of the form $\exists x\, G$ then perform a substitution $[a/x]$ (see definition below), where $a \notin L'$ and add a to L'. Else if S is of the form $\forall y_1 \ldots y_n \exists x G$, then perform a substitution $[f(y_1, \ldots, y_n)/x]$, where f is a n-dimensional function symbol and $f \notin L'$ and add f to L'.
   - Return S

4. **Transformation into conjunctive normal form**
   A formula in skolem form is brought into conjunctive normal form by applying transformation rules, such as for instance the law of de Morgan [Bry15d].

Because of its uniform structure, clauses in conjunctive normal form are usually written by separating literals with a "," rather than a $\vee$ [Bry15e]. A typical clause in first order logic is for instance $\{p(a(y), x), \neg q(b(x, z), y)\}$ with p being a predicate symbol, a, b being function symbols and x, y, z being variables, where a clause in zeroth order logic would be $\{p, \neg q\}$.

The Resolution method for zeroth order logic can be defined as followed: Let C_1, C_2 be two clauses of a finite set of zeroth order clauses in conjunctive normal form, such that a literal ℓ exists in C_1 and the complement of this literal $\overline{\ell}$ exists in C_2. The resolvent of C_1 and C_2 is defined as the clause $C_1 \backslash \{\ell\} \cup C_2 \backslash \{\overline{\ell}\}$[Rob65]. In zeroth order logic this is directly applicable by eliminating propositional variables, for instance:

$$\frac{\{a, \neg b, c\} \qquad \{a, b, \neg d, \neg e, f\}}{\{a, c, \neg d, \neg e, f\}}$$

Figure 3.1: Example for a Resolution step in zeroth order logic [Bry15e, p. 17]

However,the Resolution method for first order logic is more sophisticated. To equalize the variables in two literals where one predicate symbol is the complement of the other predicate symbol the principle of substitution is needed. An elementary substitution is any expression of the form t/v, where v is any variable and t is any term different from v [Rob65]. A substitution is a finite set of elementary substitutions $\sigma = \{t_1/v_1, ...t_n/v_n\}$. The application of the substitution σ to a clause C is obtained by simultaneously replacing every occurrence of v_i in C by t_i $(1 \leq i \leq n)$ [Rob65]. If the application of the substitution of a set of clauses A results being a singleton then the substitution is called an "unifier" of A. The Resolution method for first order logic thus can be generalized as:

$$\frac{\{L_1\} \cup K_1 \qquad \{\overline{L_2}\} \cup K_2}{K_1\sigma \cup K_2\sigma}\,\sigma$$

Figure 3.2: Resolution rule for first order logic [Bry15e, p. 34]

where σ is a most general unifier (the minimal set of substitutions) of L_1 and $\overline{L_2}$, and K_1 and K_2 are clauses in conjunctive normal form. [Rob65]. It is assumed, that $\{L_1\} \cup K_1$ and $\{\overline{L_2}\} \cup K_2$ do not share variables. If there are shared variables, those variable have to be renamed (standardization apart) to prevent misleading deductions [Bry15e].

In addition to the Resolution rule in first order logic the factorization rule is defined as:

$$\frac{\{L_1, L_2\} \cup K}{\{L_1\sigma\} \cup K\sigma}\,\sigma$$

Figure 3.3: Factorization rule for first order logic [Bry15e, p. 34]

where σ is a most general unifier of L_1 and L_2, and K is a clause in conjunctive normal form. [Bry15e].
Robinson proved, that "If S is any finite set of clauses, then S is unsatisfiable if and only if $\mathcal{R}^n(S)$ contains $\square$, for some $n \geq 0$" [Rob65, p. 30], where $\mathcal{R}(S)$ is the set obtained from S by extending S with the clause resulting from applying either the Resolution method to two clauses of S or the factoring method to one clause of S and $\mathcal{R}^n(S)$ is defined recursively as $\mathcal{R}\left(\mathcal{R}^{n-1}(S)\right)$. Since $F \vDash G$ holds if and only if $(F \wedge \neg G)$ is unsatisfiable, $F \vDash G$ holds if and only if the empty clause can be finitely derived form a clausal form of $(F \wedge \neg G)$ [Bry15e].

3.1.2 Implementation

The proof editor for Resolution was designed to assist students performing a proof in either zeroth or first order logic. Exercises can be designed as building up a proof tree from the beginning or from an already worked out part of the proof where users have to continue the proof. Figure 3.4 shows an example of the proof editor at the beginning of an exercise with no worked out part. On the upper screen, the task description is displayed. The interactive part of the editor below the task description basically splits up in two parts, namely the set of clauses, or clause container, at the left side where all clauses are displayed, and a Resolution tree at the right side. The Resolution rule is applied as shown on Figure 3.5: First, the user can click on a clause in the clause container, which subsequently is inserted into the next free space of the Resolution tree (Step1). Now further clauses can be inserted into the Resolution tree by clicking on them in the clause container (Step 2), and the literals in the clauses can be selected. The Resolution rule can be applied on selected literals via the button "Resolve" at the top left corner (Step 3). If the application of the rule is correct, a conclusion line is drawn beneath the selected clauses and the result of the Resolution is displayed below. Any literal of a clause can be selected as long as there has been no rule applied to the clause or more informal, if the clause is positioned on the lowest line of the tree. The proof tree begins at the top and grows to the bottom with each new conclusion. A complete proof in zeroth order logic therefore is built up repeating the following steps:

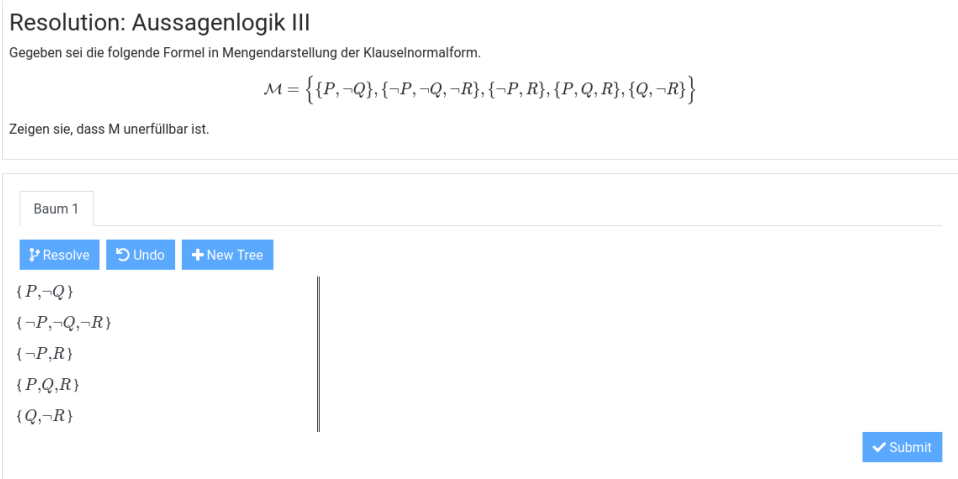

Figure 3.4: Resolution editor displayed at the beginning of an exemplary exercise

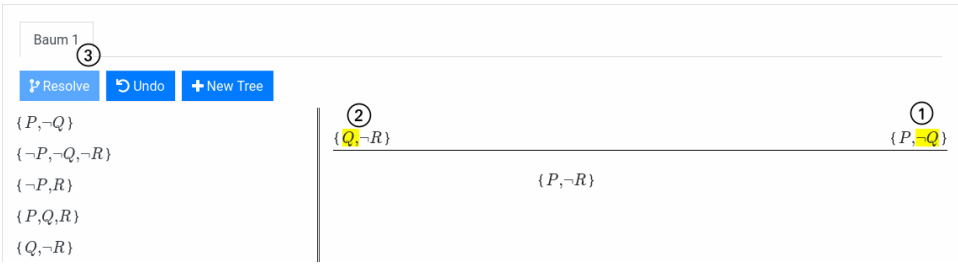

Figure 3.5: Exemplary use of the Resolution rule in zeroth order logic

1. Clicking on one (at the beginning two) clauses in the clause set.
2. Selecting in each of the newly added clause and the root of the so far constructed proof tree two literals that are complements of each other.
3. Clicking on the button “Resolve”.

Each of these steps can be reverted by either clicking on “Undo” or deselecting literals.
If users make a mistake such as selecting more than two literals or applying the Resolution rule to literals which are no complements of each other, an error with a description of the failure is displayed at the top of the Resolution tree (see Figure 3.6). Moreover, it is only possible to apply the Resolution rule if there are two clauses on the last level of the tree, otherwise the button “Resolve” is grayed out (see Figure 3.4). Furthermore, users have the possibility to continue the proof in a new subtree by clicking on “New Tree”. A new empty tab will be opened with the last clause of the current Resolution tree added to the formula set. Now the proof can be continued

Figure 3.6: Exemplary error message

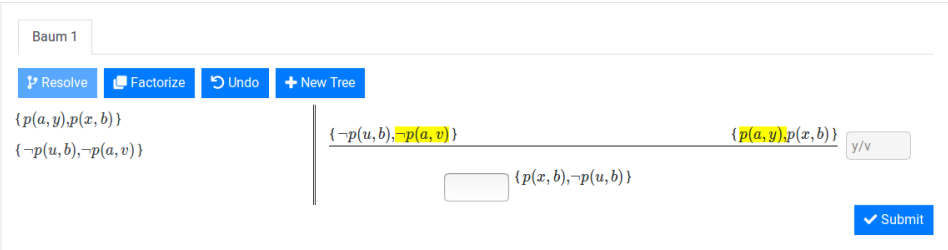

Figure 3.7: Exemplary Resolution step in first order logic with substitution

with the expanded formula set. This gives users the possibility to "save" an already derived result and to reuse it at another location for further Resolution steps (see also Figure 3.9). If the proof is correct, that is, if there is a $\square$ below the last conclusion line, a message will appear at the top of the Resolution tree, stating that the proof is correct.
The editor of first order logic extends the functionality described above. First of all it is possible to perform one or more substitutions by writing them in a text box placed at the side of each conclusion (see Figure 3.7). The substitutions follow the specifications of Robinson, that is, a substitution can have as many substitution elements of the form t/v as needed, with t being any term and v any variable different from t. Substitution elements have to be separated by a comma and it a variable can not replace a function symbol [Rob65]. If these conditions are not met, an error message is displayed. When applying the Resolution or factorization rule, the substitutions specified in the text box are applied automatically. The program also ensures that before applying the Resolution rule, the two clauses do not contain equal variables. This is achieved by checking each variable of a clause before inserting it into the Resolution tree. If a variable is already in use, then one or more dashes are assigned to the new variable as shown in Figure 3.8 at the second line. Furthermore, it is possible to apply the factorization rule by clicking on "Factorize". Analogously to the Resolution rule, there have to be selected exactly two literals, but this time only in one

clause. Figure 3.8 and Figure 3.9 show a complete proof in first order logic using substitution as well as factorization.

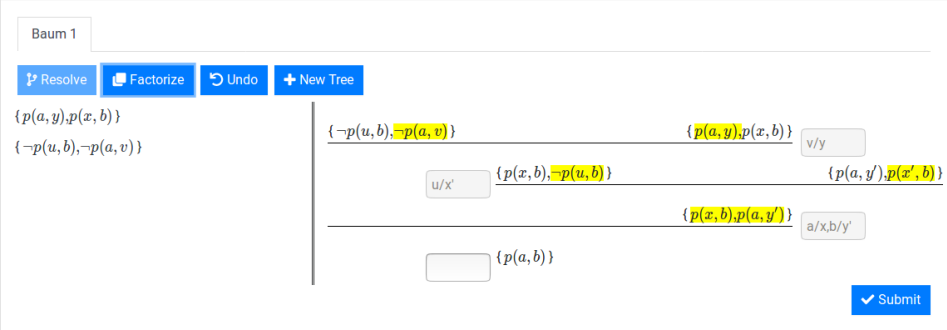

Figure 3.8: First part of a complete proof in first order logic

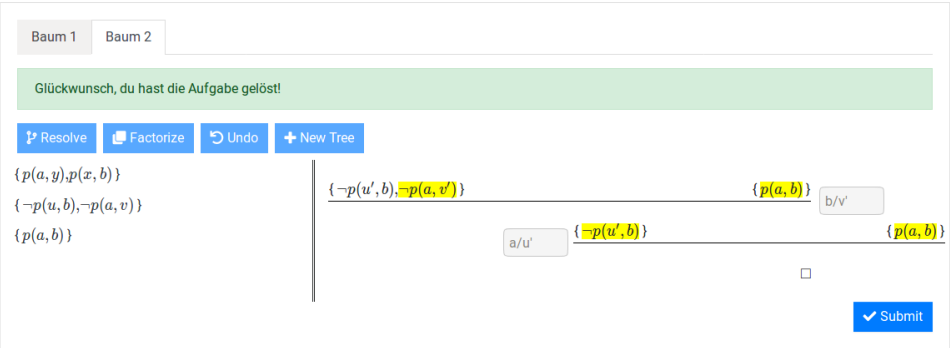

Figure 3.9: Second part of a complete proof in first order logic

Usually a Resolution proof tree is of the form

$$\dfrac{\{\neg A\} \qquad \dfrac{\{C\} \qquad \dfrac{\{\neg B\} \qquad \{A, B, \neg C\}}{\{A, \neg C\}}}{\{A\}}}{\square}$$

causing the whole tree to move to the left as it grows, because every new inserted clause, is displayed left of the actual tree. As one can see for instance in Figure 3.8, in this proof editor the third conclusion line is exactly under the first, as well as the fourth is exactly under the second. Such layout has been chosen due to the limited space for a proof tree in a web browser. Considering a large proof, either the clauses have to shrink to an nearly unreadable size, or the tree grows out of the boundaries of the webpage. Mathematically however it would be better if each newly added clause was inserted outside the previous proof tree, because it can be misleading if the newly added clause is positioned below a previous conclusion line. Apart from this, the attempt was made to design the whole Resolution tree as similar as possible to the examples given in the lecture, so that the students can work in a familiar environment.

3.2 Natural Deduction Editor

3.2.1 Natural Deduction Principle

Natural Deduction was invented by the german mathematician Gerhard Gentzen in 1933. Based upon former systems of logical deduction, the intention of Gentzen was to build a system of logical inference similar to intuitive or informal reasoning [Pra06]. Therefore, Gentzen developed the calculus of Natural Deduction as a system for deducting formulas in zeroth and first logic order. This work considers only Natural Deduction for zeroth order logic. As being closely related to Sequent Calculus, also invented by Gentzen, which is a fundamental basis of automated theorem proving besides the Resolution method, and "because of the close correspondence to practices common in intuitive reasoning, systems of Natural Deduction have often been used [...] for didactic purposes" [Pra06, p. 103]. Characteristic for a system of Natural Deduction are "conclusion figures"[Gen35] or deduction rules. An deduction rule is of the form

$$\frac{A_1 \dots A_n}{B} \; (n \geq 1)$$

[Gen35, p. 181] where A and B are formulas of the zeroth order logic. $A_1 ... A_n$ are called *premises* and B is called *conclusion* [Pra06]. A deduction is formed by a several deduction rules under the limitation that each formula is the conclusion of at most one deduction rule, each formula is premise of

at most one conclusion (except the last conclusion), and the tree formed by the repeated application of deduction rules is a non-cyclic graph [Gen35]. Restricted by these rules, a proof tree of Natural Deduction usually has a shape similar to a Resolution proof tree, witch the formula to be derived as root [Gen35]. A basic example of Natural Deduction is shown in Figure 3.10. There are different schemes for a deduction rule, which split up into *insertion* rules and *elimination* rules. Basically there are insertion and elimination rules for each logical junctor such as $\neg, \vee, \wedge, \Rightarrow$ and $\neg\neg$ as well as insertion figures for $\top$ and $\bot$ and an elimination figure for $\bot$ [Bry15b]. An insertion figure always "inserts" a formula or junctor to the premises given, for instance,

$$\frac{A \qquad B}{A \wedge B} \wedge_E$$

while an elimination figure "eliminates" parts of the premises, such as:

$$\frac{A \wedge B}{A} \wedge_{Br}$$

Note, that the abbreviations for elimination and insertion rule used by Gentzen are German, so any rule containing an E is an insertion rule, and any rule containing a B is an elimination rule. In the given example of the $\wedge$ insertion rule, the logical junctor $\wedge$ is inserted connecting A and B because if both A and B are valid, it can be followed, that $A \wedge B$ is valid as well. In the example of the elimination rule both $\wedge$ and B are eliminated, because if $A \wedge B$ is valid, it can be followed, that A is valid as well.
Many of the conclusion rules depend on assumptions. For instance the insertion figure of $\Rightarrow$ is defined as:

$$\frac{\begin{array}{c} A^{(n)} \\ \vdots \\ B \end{array}}{(A \Rightarrow B)}\,{}^{(n)} \Rightarrow_E$$

Informal, the reasoning of this rule can be described as *if B is proven using the assumption A, it can be concluded that A implies B*. Each formula between A and the $\Rightarrow$ rule is said to be dependent on A, but further conclusions are no longer dependent on A [Gen35]. It is said, that the application of the $\Rightarrow$ insertion rule "discharges" the assumption A [Bry15b, Gen35]. In a derivation, the assignment between assumption and removing rule has to be clearly marked, usually by assigning numbers in brackets to both parts [Gen35, Bry15b]. An already removed assumption can not be removed again, but more than one assumption can be removed by a rule, if they are syntactically identical [Bry15b]. A proof of a derivation $A \vDash B$ is valid if B is not dependent on any assumptions but A [Gen35], that is, if all other assumptions made during the creation of the Natural Deduction tree, have been removed by corresponding rules. Furthermore B is a tautology, if it depends on nothing [Bry15b], such as in Figure 3.10.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\neg A \Rightarrow \neg B)^{(1)} \quad \neg A^{(2)}}{\neg B}\,\Rightarrow_{\mathsf{B}} \qquad B^{(3)}}{\bot}\,(2)\ \neg_{\mathsf{E}}}{\neg\neg A}\,\neg\neg_{\mathsf{B}}}{A}}{(B \Rightarrow A)}\,(3)\ \Rightarrow_{\mathsf{E}}}{((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))}\,(1)\ \Rightarrow_{\mathsf{E}}
$$

Figure 3.10: Natural Deduction proof of $((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))$

3.2.2 Implementation

Like for the Resolution editor it is both possible to design the exercise as building up a proof tree from the beginning or from a worked out part of the tree. The interface of the deduction editor is split up in four parts, as shown on Figure 3.11:

- The task description displayed at the top of the screen.
- Buttons for each insertion and elimination rule.
- The assumptions used in the proof.
- The Natural Deduction proof tree.

At the beginning of an exercise the formula to be derived is displayed as root of an otherwise empty tree. Initial assumptions are already displayed in the assumption container. A formula in the proof tree can be selected by clicking on it and subsequently either an insertion or elimination rule can be applied by clicking on a button in the container at the left. Only the formulas to which no rule has been applied yet can be selected. The proof tree will grow from the bottom, that is, from its conclusions to the premises. Therefore each rule is inverted, for instance given the conclusion $A \wedge B$, clicking on the $\wedge$ insertion rule will produce:

$$
\dfrac{A \qquad B}{A \wedge B}\,\wedge_E
$$

If there are any further assumptions made such as by applying the $\Rightarrow$ insertion rule, the rule will be numbered and the new assumption will be displayed with the same number in the assumption container (see Figure 3.14). For each further rule application, the program checks whether the formula just created corresponds to an assumption, and if this is the case, the formula is provided with the same number as the assumption and the

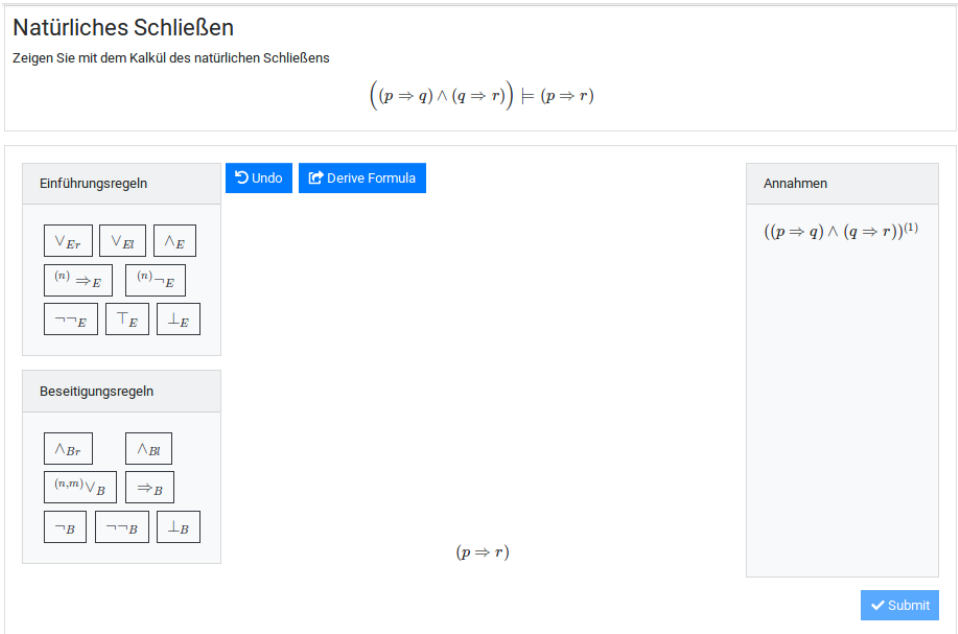

Figure 3.11: Natural Deduction editor displayed at the beginning of a proof

program registers that the assumption was used. If all assumptions made are used at least once and there are no leaves in the tree that do not match any assumption, the evidence is considered correct. This is equal to the definition of correctness by Gentzen (see above). In some cases, further interaction with the student is necessary, for instance when applying the $\Rightarrow$ elimination rule:

$$\frac{(A \Rightarrow B) \qquad A}{B}\,(n)\ \Rightarrow_B$$

As the tree grows from its conclusions, it is impossible for the program to know which formula A is meant. Therefore, a dialogue window is opened, allowing users to type in the formula they want to introduce (see Figure 3.12). The program then parses the entered string and creates a new formula for A. Similar to the Resolution editor, each step can be undone. A selected formula can be deselected and an application of a rule can be reverted by clicking on "Undo". Furthermore, if users want to apply a rule which is not possible on the selected rule, an error message is displayed at the top of the proof tree. The error message describes in detail which conditions of the rule were not met. Furthermore, the attempt was made to adjust the design of this editor as close as possible to the presentation of Natural Deduction proofs in learning material. Possible improvement still can be made regarding the conclusion lines. As for the current state, the conclusion lines will not adapt their width to the width of the formula below the conclusion

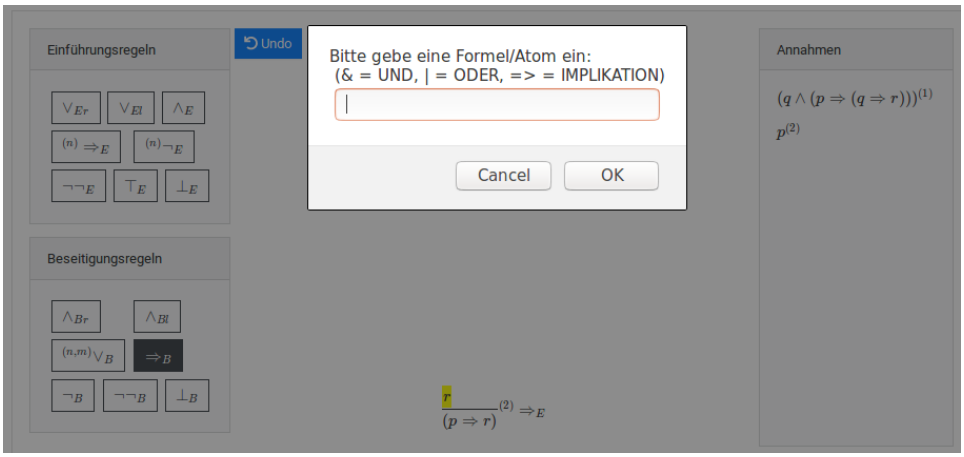

Figure 3.12: This Figure shows the window prompt, where a student can type in a formula.

line, but to the width of the whole tree above the conclusion line. As it can be seen in Figure 3.14 this causes extremely large conclusion lines. For instance, the lowest conclusion line in this example only needs to be as far as $(p \Rightarrow r)$. Due to technical limitations regarding the programming language in which the program was written, it was not possible to easily solve this problem, because the only solution seems to be manual calculation of the length of each conclusion line depending on screen size and the width of the formula below the conclusion line.
However, despite their name, derivations in Natural Deduction can be hard to understand especially for beginners who had no previous courses of logical reasoning. When beginning a proof of the form $A \vDash B$ most of the time it is easier to start from the formula to derive B and not from the assumption A. Yet, at some point or another, it will become easier to solve the rest of the proof from its assumptions, that is from the top to the bottom. Therefore the decision was made to allow derivations in both directions. The user can decide to solve parts of the proof (or the whole proof) from the top to the bottom at any time by selecting a formula and click on "Derive". Below the current proof a new proof tree will be created containing the insertion and elimination rule buttons, an empty proof tree window and all assumptions valid for the formula to be derived. In this new tree users can derive the formula using assumptions. Contrary to the main proof window, the assumptions in the assumption container are clickable and new assumptions can be made using the "new assumption" button where the same window prompt as in Figure 3.12 is opened, allowing the student to type in a new assumption. Regarding the above described further interaction with the student, when building the proof tree from the top to the bottom, it sometimes is necessary to determine the exact assumption step that the student wants

to remove. For instance when applying the insertion rule for $\Rightarrow$:

$$\dfrac{\begin{array}{c} A^{(n)} \\ \vdots \\ B \end{array}}{(A \Rightarrow B)}\,{}^{(n)}\,\Rightarrow_E$$

Here the program needs to know which assumption the user wants to have removed by the rule, for there can be more assumptions between A and B. This problem is solved by letting users type the number of the assumption step in a window prompt. Apart from that, this editor works just like the main editor with each rule inverted. If the user considers the formula to be derived, they can click on "Insert Derivation", which causes the program to check the derivation and if correct, it will be inserted in the main proof tree. Clicking "Undo" will remove both inserted assumptions and applied rules. If the student clicks on "Undo" in the main proof window, the whole sub proof will be erased. An example of this editor can be seen in figure 3.13 and 3.14. Starting with the formula $(p \Rightarrow q)$ as formula to derivate, it is very clear for a student, that there is only one rule that leads to further progress, namely the $\Rightarrow$ insertion rule. In general it is easier to work from the bottom to the top until there is only a single literal left, in this case r. At this point it is very difficult for a student to continue in the same direction. The next step, which would be applying the $\Rightarrow$ elimination rule is very unintuitive and only a person familiar to Natural Deduction proofs could easily find this step. Therefore at this point, it is better to continue the derivation r from the top to the bottom. Using the assumption $((p \Rightarrow q) \wedge (q \Rightarrow r))$, it is easy to figure out, that, given the second assumption p, it makes sense to eliminate the right part of the conjunction and apply the $\Rightarrow$ elimination rule on the result $(p \Rightarrow q)$ and p. Given the result q again it is easy to guess the next steps and r is derived much easier as it would have been from the bottom to the top.

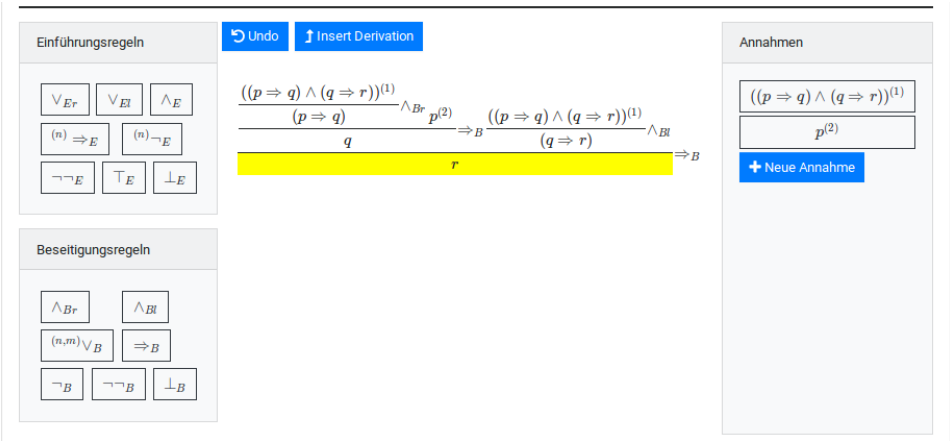

Figure 3.13: Exemplary derivation of r in the derive sub editor using the assumptions at the right

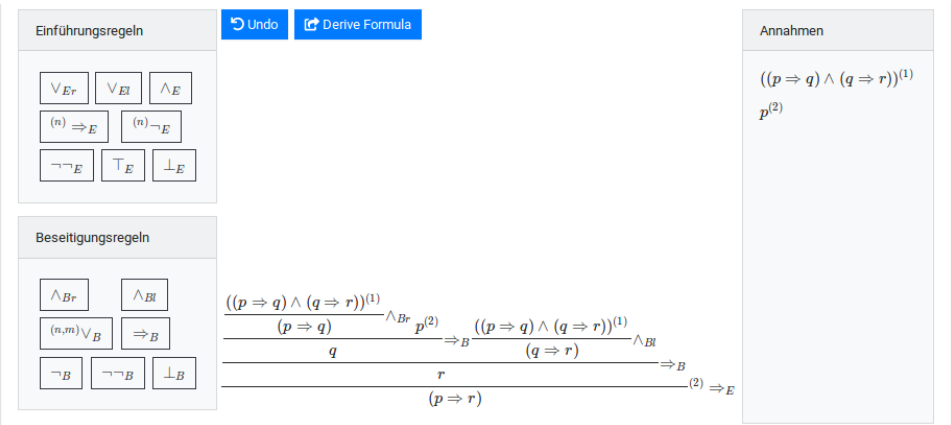

Figure 3.14: This Figure shows the derivation of Figure 3.13 after clicking on “Insert Derivation” in the main proof window.

Chapter 4

Pedagogical support

Compared to solving tasks using Resolution or Natural Deduction by hand on a piece of paper there is a lot of pedagogical help provided by the editors, mainly concerning feedback and scaffolding. However, as feedback and scaffolding are closely intertwined concepts (according to van de Pol [VdPVB10], feedback is a form of scaffolding), it is difficult to fully distinguish the two concepts. Thus, this chapter first will analyze the help provided in the program from the point of scaffolding and later from the point of feedback, although there are some intersections for help may both have scaffolding and feedback function.

4.1 Scaffolding

Following the classification of Kirschner, the program provides procedural information rather than supportive information. The scaffolds are mainly designed to reduce the cognitive load during task solving and aim to simplify *recurrent* task aspects rather than *non recurrent* aspects, such as building up mental models, which is already done by the lecture. The main scaffolds providing procedural information are:

- **Error messages**
  Error messages provide information regarding the correct usage of either Resolution or Natural Deduction rules. It can be both very time-consuming and thought-intensive to search for an error if the program simply would do nothing when a mistake has been made. With an adequate error message users can immediately see, where the error is and correct it. Thus, error messages clearly reduce cognitive load during the task process.

- **Visual scaffolding**
  The visual displaying of the formulas and the proof tree is an important scaffold. Users do not have to write formulas by themselves which

may lead to spelling errors and they do not have to take care about correct parenthesis or the drawing of conclusion lines. Furthermore, the possibility to revert each step is much more simpler than erasing the tree on a piece of paper or crossing something out and restart building a proof tree. Therefore users can concentrate more on building the proof tree, rather than designing it. As being designed closely to the visualization introduced in the lecture, the editors may also improve cognitive structuring as defined by van de Pol [VdPVB10], because the whole process of proving is reduced to its relevant parts.

- **Application of rules**
  Another important scaffold is the application of both Resolution and Natural Deduction rules as the correct application of the rules can be seen as the main component in a proof. It is ensured by the program that, under the condition that the application is possible, a selected rule it is applied correctly. Once more this decreases cognitive load significantly as users do not even need to be familiar with the application of a rule in order to try it out. When experimenting with the rules users might even learn the function of a rule just by receiving a direct outcome.

- **Didactic Reduction**
  Regarding the automatic management of assumptions, the Natural Deduction editor provides Didactic Reduction as defined by Grüner [Grü67]. As it can be seen in Figure 4.1, Didactic Reduction simplifies certain parts of a task or model so that learners do not have to learn the omitted parts at the beginning of their learning process [Grü67]. In the Natural Deduction editor each assumption is numbered automatically and when derivating a formula from the top to the bottom, all assumptions valid for the formula are automatically displayed in the new proof editor window. Thus, users never have to take care about assumptions at all, which means they do not have to learn this task in order to successfully prove something. This feature extends the original meaning of scaffold as it does not help users to perform an action, but performs the action by itself. A similar concept can be found in the Resolution editor as well, as dashes are assigned automatically to variables in order to avoid duplicates.

The editors are designed to allow both decreasing *intrinsic* and *extraneous* cognitive load, as they support the arrangement of tasks in simple to complex order, as well as the usage of already worked out parts of the proof tree. For decreasing intrinsic cognitive load, the process of fading can be achieved by grouping the exercises into task classes with increasing complexity. As for Resolution, the main task classes could be zeroth order logic as first task class, and first order logic as second, where these classes could be divided

once again into criteria such as the amount of conclusions or the amount of new trees necessary for the minimal proof tree. For Natural Deduction possible criteria for task classes could be the height of the minimal proof tree, or the amount of different deduction rules, because proofs based upon the repeated usage of a single rule, e.g. the $\Rightarrow$ rules, tend to be simpler than proofs with many different rules. Nevertheless, a proof tree based upon few rules and having a small minimal tree height still can be unintuitive and difficult to understand. Thus, as complexity is hard to measure by objective criteria, the grouping by task classes is best be done individually by the task creator. For decreasing extraneous load, the process of fading can be achieved by having exercises of more or less the same complexity and starting with a large part of the proof already worked out. In further exercises this part can be reduced, until there is no worked out part left and users can solve the task from the beginning of the proof.

4.2 Feedback

Feedback in both editors is given mostly at task or process level. First of all, users receive a positive feedback if they have solved an exercise. This feedback is directed at the task level, since it only states the correctness of a proof. Regarding the principle of Hattie, that feedback works best if "goals are specific and challenging but task complexity is low" [HT07, p. 85f.], it would be the best if users would receive feedback about correctness after every application of a rule to reduce task complexity, but since there are many ways to construct a proof, this is not possible. Nevertheless, especially in the Natural Deduction editor, this feedback is important as the correctness of a proof might not be directly visible for a student, and it also has a positive affect on motivation. Another important feedback is the visual feedback provided when applying rules. Users immediately receive feedback directed at task level about whether the visual result matches their expectations of the result and feedback at process level, as this feedback also may lead to further knowledge about how to apply the selected rule in general. The error messages provided when users have made a mistake can also be seen as feedback directed both at task and at process level. Error messages provide feedback about what mistake has been made especially in this task, but also feedback about the general application of rules. As mentioned in Chapter 2, this combination of task and process level is a very powerful type of feedback as this knowledge can be applied on all subsequent tasks and improves the understanding of the whole topic.

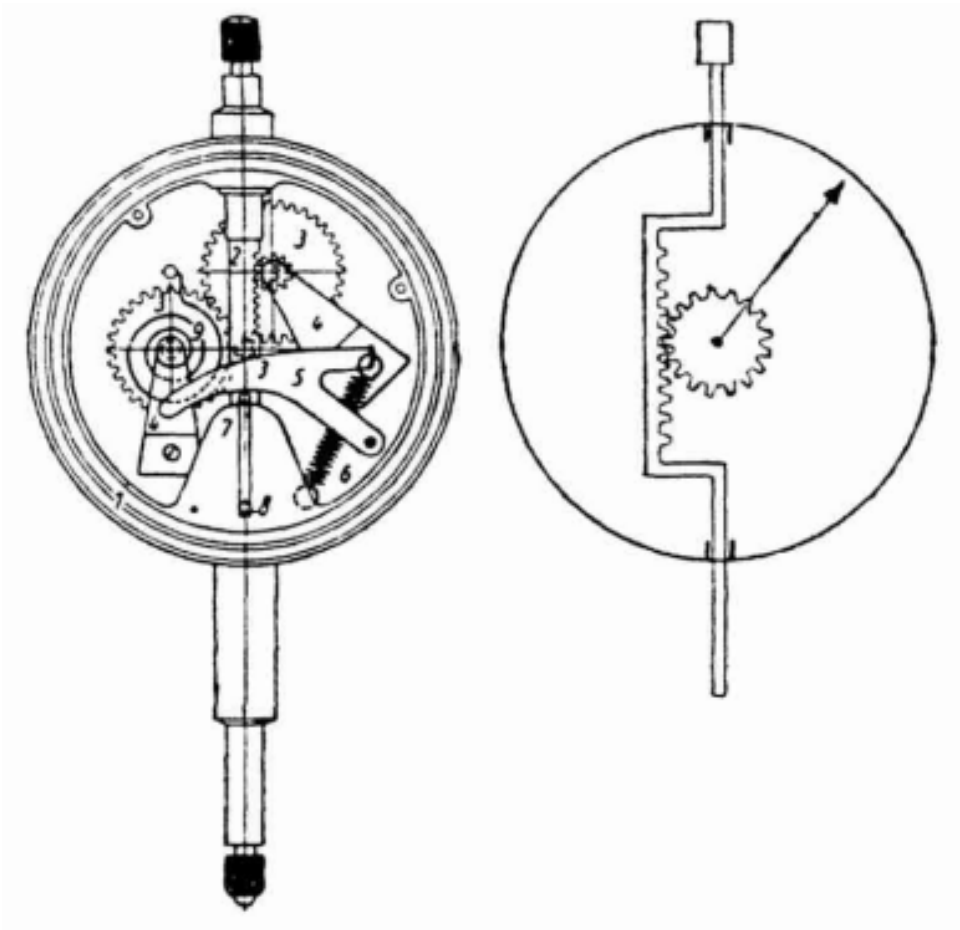

Figure 4.1: An example of Didactic Reduction for the mechanism of a dial indicator [Grü67, p. 419]

Chapter 5

Evaluation

5.1 Integration in Backstage

For evaluating the proof editors, they were implemented in the classroom communication system "Backstage", which was designed to support teaching and learning in large class lectures [BP17]. Backstage allows students to ask questions and exchange information during the lecture without interrupting it [BP17]. As teachers can see and answer these questions as well, they also get feedback for instance about the understandability of their talk [BP17]. Furthermore, Backstage aims to "abridge the gap between classroom learning and homework" [BP17, p. 112], therefore Backstage also provides exercises for students to train and repeat the content of the lecture. The logic editors were implemented as additional exercises for exam preparation of the course "Logic and Discrete Mathematics". As the functionalities of Backstage already provide different ways for feedback, the feedback users receive while solving an exercise in a proof editor is extended by the architecture of Backstage as follows:

- **Peer feedback**
  As users can ask questions related to the exercises, which can be answered by peers and teachers alike, feedback directed at all levels can be provided. Both task and process related questions can be posed, and, as mentioned in Section 2.1, answering questions can improve the capacity of self evaluation.

- **Providing information to teachers**
  The platform can check the correctness of the exercises solved by a user allowing teachers to determine how many users have solved the exercise, and how many provided a wrong answer. This information can be used to adjust the lecture regrading possible problems in understanding.

- **Exemplary Solutions**
  Users are informed about the correctness of their solution by the platform as well and they receive an exemplary solution of the proof. By comparing their own work to the exemplary solution, users can determine own errors and get an impression of good performance. As described in Section 2.1, this process not only informs about the correctness of a task, it also provides feedback at process level and at the level of self regulation.

Besides, the lecture material can be found on Backstage as well, allowing users to look up definitions, strategies, or instructions for Resolution or Natural Deduction methods. Regarding the process of scaffolding, this can be seen as *supportive* information. A short sample videos about how to solve an exemplary Resolution and Natural Deduction exercise was provided to the students as well.

5.2 Design of the exercises

For Resolution there were five exercises, beginning with three simple zeroth order logic tasks followed by two first order logic tasks. The first task did not require the usage of the option "New Tree", as opposed to the other ones which required the usage of sub trees. Furthermore, all tasks were solvable within the range of four to six conclusion steps. For Natural Deduction, there were eight exercises beginning with four proofs, which heavily depend on the usage of both implication rules. Proofs of this kind were considered easy to solve, as the implication rules are intuitively understandable. These exercises were followed by two still easy to solve tasks, namely proving the commutative law, where the proof tree only has a height of three conclusions, and another simple exercise with small tree height that was discussed in lecture. The last two exercises were the proof in Figure 3.10 and the proof of the distributive law, both of which are more difficult and not always intuitive as they require more complex deduction rules such as the $\vee$ elimination rule.

5.3 Results

As being a course of approximately 600 students, at least a third of the students used the proof editors for their exam preparation. The number of individual responses per exercise ranges from 100 to 180. However, students often submitted more than one answer. For instance, if the first answer was not correct, many students tried solving it once again. Therefore, there were sometimes up to 500 answers submitted for a single exercise. The editor for Resolution was used slightly more often than the editor for Natural Deduction, which could be due to the fact that Resolution proofs are usually

easier to solve. As Figure 5.2 shows, the percentage of correct answers is almost equal for all exercises of zeroth order Resolution. Compared to the percentages of exercise 4 and 5, which were exercises of first order logic, it clearly can be seen that Resolution in first order logic is more challenging for students. Moreover, in Figure 5.4 it can bee seen, that in both task classes the exercise with the lowest number of average attempts are at the last task of the task class (exercise 3 and 5). This may be indicating that there has been a learning process for some students during the solving of the previous exercises. Figure 5.3 underlines this hypothesis as exercise 3 and 5 have the highest percentage of correct first try responses in their task classes. The low percentage in Figure 5.3 for the second exercise might be explainable by the unfamiliar usage of the "New Tree" option, which was not necessary in the first exercise. However, another possibility for these results could be, that students who were not able to successfully solve an exercise stopped doing the following exercises. As shown in Figure 5.1 the number of participants decreases at the end of each task class. If only the students, who successfully solved the exercise continued with the further ones, the hypothesis of the learning process is questionable.
As for Natural Deduction Figure 5.6 shows that the task class of the implication rule exercises (1-4) was solved almost as good as the task class with the two exercises with small tree height. It is hardly surprising that students had the most problems with the last task class containing two rather difficult exercises. When comparing the average number of attempts for the Resolution exercises 5.4 with those for the Natural Deduction exercises 5.8 it can be seen that, apart from the last two difficult exercises, the solving of Natural Deduction exercises required less attempts than the solving of Resolution exercises. This is an interesting result because, as mentioned before, Natural Deduction proofs usually are more challenging for students. Just as for Resolution the progress students made while working on the tasks cannot be determined with certainty. In Figure 5.7 and Figure 5.8 it can be seen that from exercise one to exercise four the average attempts decreases and the percentage of correct answers at the first try increases. As these tasks were almost the same in their complexity and in the approach of proving, this could be indicating that students made efforts in understanding the principle of Natural Deduction. Still it is possible that only the students who successfully solved an exercise continued with the following exercises, as the number of participants for an exercise decreases if the average number of attempts decreases (see Figure 5.5 and 5.8). To determine the exact learning progress, further research has to be made with a constant number of participants.

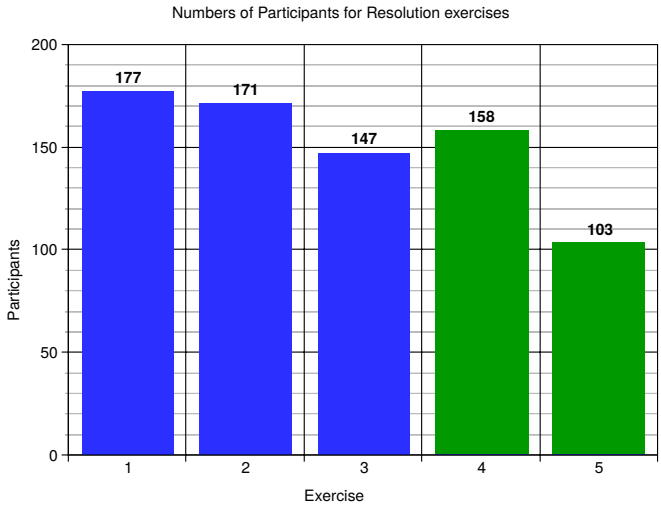

Figure 5.1: Numbers of participants for the Resolution exercises in zeroth order (blue) and first order logic (green)

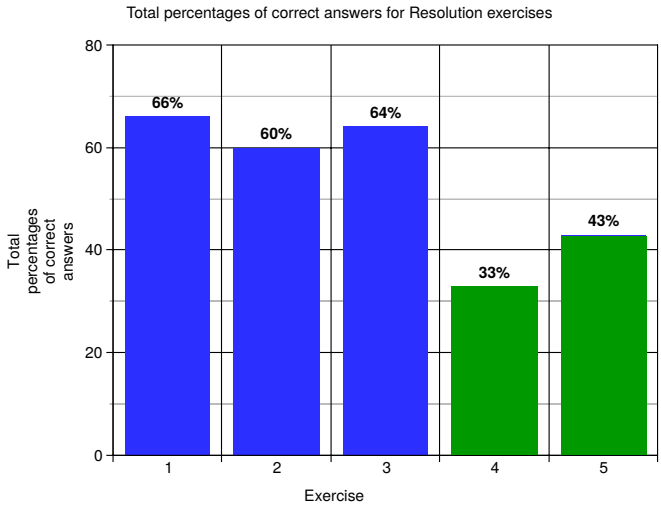

Figure 5.2: Total percentages of correct answers for the Resolution exercises in zeroth order (blue) and first order logic (green)

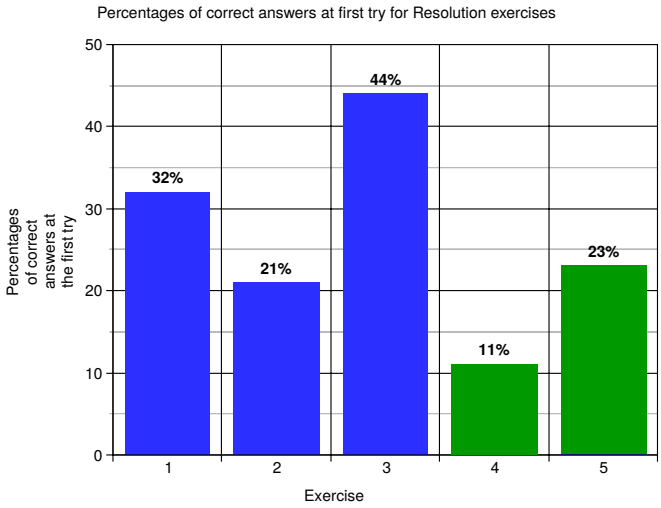

Figure 5.3: Percentages of correct answers at the first try for the Resolution exercises in zeroth order (blue) and first order logic (green)

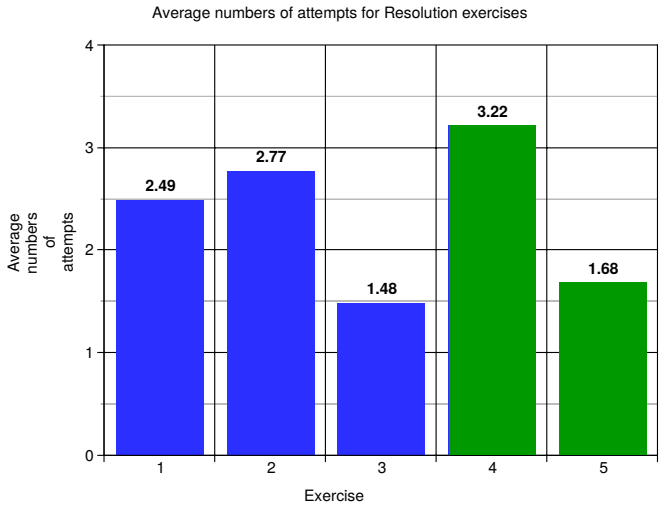

Figure 5.4: Average numbers of attempts for solving the Resolution exercises in zeroth order (blue) and first order logic (green)

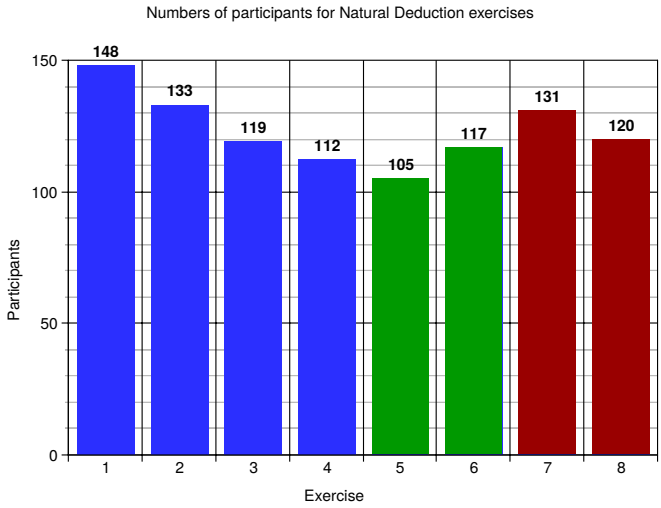

Figure 5.5: Numbers of participants per exercise for Natural Deduction grouped by task classes

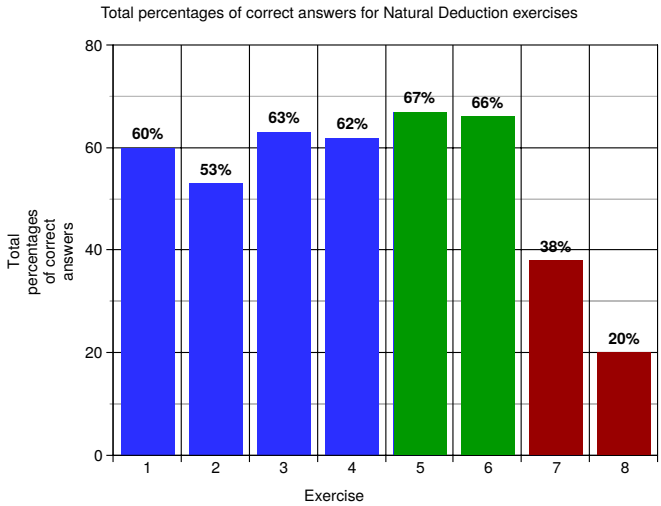

Figure 5.6: Total percentages of correct answers for the Natural Deduction exercises grouped by task classes

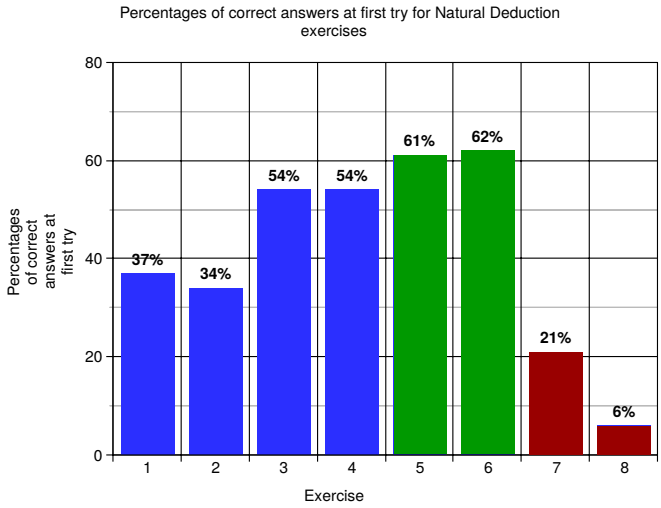

Figure 5.7: Percentage of correct answers at the first try for the Natural Deduction exercises grouped by task classes

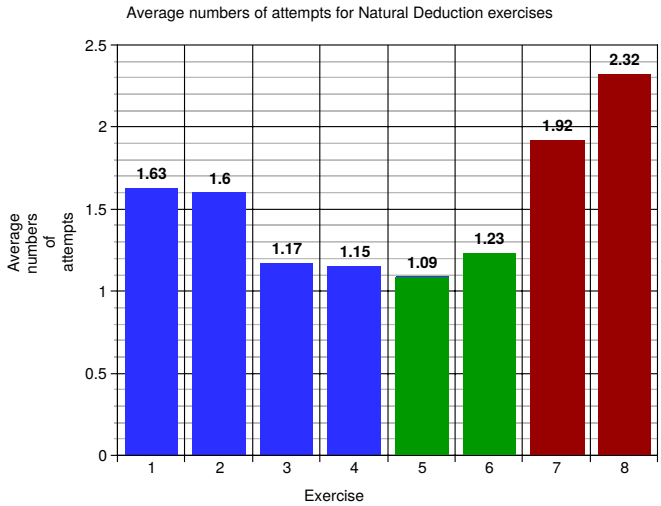

Figure 5.8: Average numbers of attempts for solving the Natural Deduction exercises grouped by task classes

Chapter 6

Conclusion and Future Work

This thesis reports on the design of two proof editors for learning, one for the Resolution method and one for the Natural Deduction method, allowing students to prove logic formulas. Both editors were not only designed to assist students in understanding these two methods, but also to enhance the understanding of formal methods in general. The Resolution editor allows users to prove the validity of a conclusion $F \vDash G$, where F and G are formulas in conjunctive normal form, by deriving the empty clause from $(F \wedge \neg G)$. Resolution in both zeroth and first order logic is possible, as the editor for the latter extends the functionality by allowing users the application of substitutions and factorizations. The Natural Deduction Editor allows users to prove the validity of any statement $F \vDash G$, by building up a proof tree as defined by Gentzen. Currently only proofs in zeroth order logic are supported. In order to make logical reasoning easier, proof trees in Natural Deduction can be built up both from the top and from the bottom. To make the learning process as easy as possible, the editors were designed close to the material provided in lecture and users are additionally supported by the didactic methods scaffolding and feedback. Scaffolding, as a means of reducing cognitive load during task solving, is mainly provided by error messages, the automatic application of deduction rules, the fact that users do not have to write the proof tree on their own, and didactic reduction. The process of fading can be achieved by grouping exercises by their difficulty, starting with already filled out parts or reducing some of the scaffolding mentioned. Feedback, mainly directed at the task and process level, is provided by error messages, visual feedback about the correct application and the statement of correctness users receive when solved an exercise successfully. The editors were evaluated in the environment of the classroom communication system Backstage, whose architecture extends the feedback provided. An evaluation of 200 students results solving exercises with the editors showed that the editors were well received by the students and although further investigations have to be made about the impact on

learning, an enhancement of student skills regarding Resolution and Natural Deduction is visible. Compared to the editors already existing, these editors are definitely more apt to assist students in understanding formal methods.

6.1 Improvement of the Editor's Functionalities

Due to limited time and capacities when working on the implementation, there is still much possibility to further improve the editors. The functionality of the Resolution editor covers almost everything of the lecture content, except that it is not possible to perform a substitution for function symbols with arities higher than 1 in first order logic. For example the substitution $p(f(x), y)[a/x] \vDash p(f(a), y)$, is currently not possible, as x is not recognized as a variable by the program. This limitation could be overcome by using a simple linear strategy extending the substitution process. The string of arguments has to be searched for occurrences of the expression to be replaced (here x) and the occurrences can be replaced by the given term (here a). The Natural Deduction editor can be improved in two ways. First, it is only possible to negate atoms, and not complete formulas or literals. For instance, it is neither possible to display a formula like $\neg\neg(A \Rightarrow B)$, nor come to a conclusion result with such a formula. Therefore, the inner architecture of the formula representation in the program has to be changed that way, that each formula can be negated as well and the rules have to be changed dealing with negated formulas. Second, the solving of exercises in first order logic would be desirable because the lecture covers this topic as well. This extension could be rather difficult to implement as formulas in first order logic are far more complex than in zeroth order logic. Apart from adding two more rules dealing with $\exists$ and $\forall$ quantifiers, the internal representation of formulas would have to be changed significantly and the rules would have to be adjusted to first order logic.

6.2 Scaffolding Approaches

More scaffolds could be used to further improve the support users get when solving an exercise. It is already possible to facilitate exercises by having the begin of an proof tree already filled out so that users only need to solve the last part of a proof. Still, these exercises could be facilitated even more by having users fill out gaps instead of applying rules. This can be done by letting users write a formula in a text field which subsequently is inserted into the proof tree or display various buttons with possible formulas and letting users decide which formula has to be inserted into the gap by clicking on the correct button. This functionality would enhance the possibilities of fading, because it allows a better adaption to student abilities as the different degrees of difficulty are extended. Users can start by filling out gaps, then

working with partly worked out examples and finally solve a complete proof. Another simple but useful scaffold would be to provide more procedural information regarding the conclusion rules in both Resolution and Natural Deduction. Currently, users can apply rules, but they have to look up the formal definition and mechanism of that rule in lecture material. Especially for Natural Deduction there are many conclusion rules, so that it would be useful to have that information placed right in the editor in order to facilitate looking up the definition of a rule. This can be achieved for instance by generating tooltips for the rule buttons containing the formal definition. When hovering over a button such as ${}^{(n)} \Rightarrow_B$, the formal definition

$$\frac{(A \Rightarrow B) \qquad A}{B} \; {}^{(n)} \Rightarrow_B$$

would be displayed as tooltip. In the process of fading this feature can be removed as well, allowing to further differentiate the levels of difficulty.

6.3 Improvement of the Integration in Backstage

The integration in Backstage gives further room for improvement. While feedback is already covered very well by Backstage, further improvement can be made to maintain student motivation through scaffolding. In order to better adapt the exercises to student capabilities, the sequence of exercises could be dynamic instead of static. If a student fails to solve an exercise, the next exercise should be an easier one in the same task class or even below the current task class. Otherwise, if a student solves an exercise easily, the next exercise should be more difficult. Additionally this can be combined with a progress bar informing students about their learning progress and the current level of difficulty. In doing so students with small knowledge of the methods will be less frustrated, because at some point they will always get an exercise which they can solve, and students who know the methods well wont be bothered with useless exercises regarding their learning process. While for this work the editors were implemented as exam preparation in Backstage, it is also possible to extend the use of these editors by using them during lecture. As the course “Logic and Discrete Mathematics” is already interactive allowing students to work on smaller tasks in form of quizzes during lecture, the editors could be used as part of these quizzes as well. This would give students the opportunity to immediately try out what they have learned enhancing both student understanding and teacher feedback about understandability.

Bibliography

[AH05] Roger Azevedo and Allyson F. Hadwin. Scaffolding self-regulated learning and metacognition: Implications for the design of computer-based scaffolds. *Instructional Science*, 33:575–577, 2005.

[BP17] François Bry and Alexander Yong-Su Pohl. Large class teaching with Backstage. *Journal of Applied Research in Higher Education*, 9(1):105–128, 2017.

[Bry15a] François Bry. Vorlesung "Logik und Diskrete Strukturen", Kapitel Aussagenlogik - Teil 1. *Vorlesung Logik und diskrete Strukturen, Lehrstuhl für Programmier- und Modellierungssprachen, Institut für Informatik, Ludwig-Maximilians-Universität München*, 2015.

[Bry15b] François Bry. Vorlesung "Logik und Diskrete Strukturen", Kapitel Natürliches Schließen. *Vorlesung Logik und diskrete Strukturen, Lehrstuhl für Programmier- und Modellierungssprachen, Institut für Informatik, Ludwig-Maximilians-Universität München*, 2015.

[Bry15c] François Bry. Vorlesung "Logik und Diskrete Strukturen", Kapitel Prädikatenlogik erster Stufe - Teil 1. *Vorlesung Logik und diskrete Strukturen, Lehrstuhl für Programmier- und Modellierungssprachen, Institut für Informatik, Ludwig-Maximilians-Universität München*, 2015.

[Bry15d] François Bry. Vorlesung "Logik und Diskrete Strukturen", Kapitel Prädikatenlogik erster Stufe - Teil 2. *Vorlesung Logik und diskrete Strukturen, Lehrstuhl für Programmier- und Modellierungssprachen, Institut für Informatik, Ludwig-Maximilians-Universität München*, 2015.

[Bry15e] François Bry. Vorlesung "Logik und Diskrete Strukturen", Kapitel Resolution. *Vorlesung Logik und diskrete Strukturen,*

Lehrstuhl für Programmier- und Modellierungssprachen, Institut für Informatik, Ludwig-Maximilians-Universität München, 2015.

[BW98] Paul Black and Dylan Wiliam. Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74, 1998.

[CW96] Edmund M Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39(1):176–210, 1935.

[Gib02] Pauline Gibbons. *Scaffolding language, scaffolding learning: Teaching second language learners in the mainstream classroom*, volume 428. Heinemann Portsmouth, NH, 2002.

[Grü67] Gustav Grüner. Die didaktische Reduktion als Kernstück der Didaktik. *Die Deutsche Schule*, 59(7/8):414–430, 1967.

[HT07] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007.

[KD96] Avraham N. Kluger and Angelo DeNisi. The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological bulletin*, 119(2):254, 1996.

[Kul77] Raymond W. Kulhavy. Feedback in written instruction. *Review of Educational Research*, 47(2):211–232, 1977.

[KWT+85] Raymond W. Kulhavy, Mary T. White, Bruce W. Topp, Ann L. Chan, and James Adams. Feedback complexity and corrective efficiency. *Contemporary Educational Psychology*, 1985.

[NMD06] David J. Nicol and Debra Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, 31(2):199–218, 2006.

[OM13] Paul Orsmond and Stephen Merry. The importance of self-assessment in students' use of tutors' feedback: A qualitative study of high and non-high achieving biology undergraduates. *Assessment & Evaluation in Higher Education*, 38(6):737–753, 2013.

[Pra06] Dag Prawitz. *Natural deduction: A proof-theoretical study.* Courier Dover Publications, 2006.

[PZ02] Paul R. Pintrich and Akane Zusho. The development of academic self-regulation: The role of cognitive and motivational factors. *Development of Achievement Motivation*, 2002.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

[VdPVB10] Janneke Van de Pol, Monique Volman, and Jos Beishuizen. Scaffolding in teacher–student interaction: A decade of research. *Educational Psychology Review*, 22(3):271–296, 2010.

[VMKK03] Jeroen JG. Van Merriënboer, Paul A. Kirschner, and Liesbeth Kester. Taking the load off a learner's mind: Instructional design for complex learning. *Educational Psychologist*, 38(1):5–13, 2003.

[Vyg80] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes.* Harvard university press, 1980.

[WBR76] David Wood, Jerome S. Bruner, and Gail Ross. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17(2):89–100, 1976.