The Sequent Calculus Trainer with Automated Reasoning – Helping Students to Find Proofs

Arno Ehle Norbert Hundeshagen Martin Lange
School of Electrical Engineering and Computer Science
University of Kassel, Germany
post@arnoehle.de hundeshagen@uni-kassel.de mlange@uni-kassel.de

The sequent calculus is a formalism for proving validity of statements formulated in First-Order Logic. It is routinely used in computer science modules on mathematical logic. Formal proofs in the sequent calculus are finite trees obtained by successively applying proof rules to formulas, thus simplifying them step-by-step.

Students often struggle with the mathematical formalities and the level of abstraction that topics like formal logic and formal proofs involve. The difficulties can be categorised as syntactic or semantic. On the syntactic level, students need to understand what a correctly formed proof is, how rules can be applied (on paper for instance) without leaving the mathematical framework of the sequent calculus, and so on. Beyond this, on the semantic level, students need to acquire strategies that let them find the right proof.

The Sequent Calculus Trainer is a tool that is designed to aid students in learning the techniques of proving given statements formally. In this paper we describe the didactical motivation behind the tool and the techniques used to address issues on the syntactic as well as on the semantic level.

1 Introduction

Complaints among computer science students about the perceived difficulty and toughness of theoretical modules in their curricula are ubiquitous. This can typically be attributed to the comparably more formal content of those courses, which require a deeper understanding of the taught material in order to successfully solve exercises and ultimately pass exams.

This paper addresses one particular such issue, occurring in courses on formal logic. Most university curricula in computer science include such a course as a basic module, as part of either the mathematical foundations or the theoretical computer science strand. In fact, recommendations on the content of computer science courses usually include mathematical logic in the form of a stand-alone module or as part of a more general module on discrete structures, c.f. [1, 16], and this should include the first-order predicate calculus besides basic propositional logic.

Teaching mathematical logic typically includes the study of a syntactic proof system like natural deduction [14, 17], resolution [6, 19], Hilbert proof systems or the sequent calculus [14, 22]. Some typical basic-level exercises ask for a formal proof of a given formula of first-order logic. One of the greatest challenges for the students there is to understand the connection between semantic and syntactic deduction. Even if the student has an intuitive idea of why the given formula is valid, turning this intuition into a formal proof still requires them to have obtained the understanding of this deep logical connection. So finding a proof for a given formula requires the students to have acquired the ability to reason using both syntactic and semantic tools – the former for a rigorous formulation of proof steps and the latter for a deeper understanding of the underlying mathematical structures involved (c.f. [23]).

P. Quaresma and W. Neuper (Eds.): 6th International Workshop
on Theorem proving components for Educational software (ThEdu'17)
EPTCS 267, 2018, pp. 19–37, doi:10.4204/EPTCS.267.2

It is worth noting that this connection between semantic and syntactic reasoning is enabled mathematically through the existence of completeness theorems for the underlying calculi, in the sense of Hilberts programm, Gödel's Completeness Theorem [15], Gentzen's Hauptsatz [14, 22] and so on. One of the main reasons for teaching completeness as a central concept is to form this connection in students' minds. This is beneficial not only for learning formal logic but for the computer science education in general given that the essence of finding algorithmic solutions to any kind of problem lies in a syntactical characterisation of this problem for otherwise computers would not be able to carry out the solution by any means of symbolic manipulation of variable values, memory contents, etc.

The Bachelor's curriculum for computer science at the University of Kassel contains a mandatory 2nd-year course on formal logic, which focuses on the model and proof theory of first-order logic with equality. Propositional logic is presented as a true fragment of first-order logic. The course has been rigorously shaped with the aim of improving learning outcomes and therefore reducing failure rates. A central point of its organisation is the use of constructivistic learning theory, i.e. students are expected to learn formal logic in a highly self-organised and self-regulated way. This includes machanisms like electronic feedback systems, tool support, highly structured learning material and – to a smaller degree – the use of the inverted-classroom model (c.f. [18]). This model focuses on learning, literally, as a self-organised activity; consequently, the course engages 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* (SCT). In [12] a first version has been introduced, which solely focused on reducing mistakes made by students on the syntactic level. Empirical data in the form of exam results suggest that this earlier version of SCT indeed helps students to master the challenges of this level. Namely, a reduction in syntactical mistakes in corresponding exam papers could be noticed, and it could also be linked to the use of the SCT tool.

However, this feature alone is not enough to train students adequately; the semantical level still needs to be achieved. In other words, understanding what a correct proof is, is only the first step in finding one. Consequently, the Sequent Calculus Trainer has been extended to tackle the problem of helping students to master the semantical level as well. It uses a simple feedback system which can give the user a hint about how to construct a proof, first of all by directly issuing a warning when a bad step has been taken (i.e. a rule has been applied by the user such that the resulting subgoals have been identified to be unprovable), but also by being able to make suggestions about which rule how to apply next in order to get closer to finishing the construction of the formal proof.

The aim of this paper is to present the background and internal technology of the Sequent Calculus Trainer, now capable of assisting students to find the right proof and thus also addressing the aforementioned semantic level of correct reasoning.

In the following the word calculus refers to the sequent calculus, as presented in Section 2.

2 The Sequent Calculus for First-Order Logic with Equality

This section recalls the definition of the syntax and semantics of First-Order Logic with Equality over uninterpreted function symbols and a formal proof system for validity known as the Sequent Calculus.

2.1 First-Order Logic with Equality

A *signature* is a list $\tau = \langle R_1, \ldots, R_n, f_1, \ldots, f_m \rangle$ of *relation symbols* R_i and *function symbols* f_i. Each of these implicitly has an *arity* denoted by $ar(R)$, resp. $ar(f)$.

Let $\mathcal{V} = \{x, y, z, \ldots\}$ be a countably infinite set of (first-order) variables. *Formulas* φ, ψ and *terms* $t_1, t_2, \ldots$ of First-Order Logic with Equality over τ, FOL$[=, \tau]$ for short, are given by the following grammar.

$$\begin{aligned} \varphi, \psi &::= R(t_1, t_2, \ldots, t_{ar(R)}) \mid t_i = t_j \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \to \psi \mid \varphi \leftrightarrow \psi \mid \exists x\, \varphi \mid \forall x\, \varphi \\ t_1, t_2, \ldots &::= x \mid f(t_1, t_2, \ldots, t_{ar(f)}) \end{aligned}$$

where $x \in \mathcal{V}$, R is a relation symbol in τ and f is a function symbol in τ. Note that function symbols of arity 0 are terms as well, called *constants*.

Terms and formulas are interpreted in τ-*structures* $\mathscr{A}$, consisting of a set U – called the *universe*, and,

- for each $i = 1, \ldots, n$, a relation $R^{\mathscr{A}}$ over U of arity $ar(R)$, and
- for each $i = 1, \ldots, m$, a total function $f^{\mathscr{A}}$ on U of arity $ar(f)$.

A τ-*interpretation* is a pair $(\mathscr{A}, \vartheta)$ consisting of a τ-structure $\mathscr{A}$ with a universe U and a variable interpretation $\vartheta : \mathcal{V} \to U$.

Terms denote elements of τ-structures; the value of term t under a τ-interpretation $(\mathscr{A}, \vartheta)$ is denoted by $[\![t]\!]^{\mathscr{A}}_{\vartheta}$ and is obtained by successively applying the functions corresponding to the symbols in the term, starting with those elements pointed to by the variable interpretation:

$$[\![x]\!]^{\mathscr{A}}_{\vartheta} := \vartheta(x) \quad , \quad [\![f(t_1, \ldots, t_m)]\!]^{\mathscr{A}}_{\vartheta} := f^{\mathscr{A}}([\![t_1]\!]^{\mathscr{A}}_{\vartheta}, \ldots, [\![t_m]\!]^{\mathscr{A}}_{\vartheta})$$

The satisfaction of a FOL$[=, \tau]$-formula φ by a τ-interpretation $(\mathscr{A}, \vartheta)$ is denoted by $\mathscr{A}, \vartheta \models \varphi$ and is inductively explained as follows.

$$\begin{aligned}
\mathscr{A}, \vartheta &\models R(t_1, \ldots, t_n) && \Leftrightarrow && ([\![t_1]\!]^{\mathscr{A}}_{\vartheta}, \ldots, [\![t_n]\!]^{\mathscr{A}}_{\vartheta}) \in R^{\mathscr{A}} \\
\mathscr{A}, \vartheta &\models t = t' && \Leftrightarrow && [\![t]\!]^{\mathscr{A}}_{\vartheta} = [\![t']\!]^{\mathscr{A}}_{\vartheta} \\
\mathscr{A}, \vartheta &\models \varphi \wedge \psi && \Leftrightarrow && \mathscr{A}, \vartheta \models \varphi \text{ and } \mathscr{A}, \vartheta \models \psi \\
\mathscr{A}, \vartheta &\models \varphi \vee \psi && \Leftrightarrow && \mathscr{A}, \vartheta \models \varphi \text{ or } \mathscr{A}, \vartheta \models \psi \\
\mathscr{A}, \vartheta &\models \neg\varphi && \Leftrightarrow && \mathscr{A}, \vartheta \not\models \\
\mathscr{A}, \vartheta &\models \varphi \to \psi && \Leftrightarrow && \mathscr{A}, \vartheta \models \varphi \text{ implies } \mathscr{A}, \vartheta \models \psi \\
\mathscr{A}, \vartheta &\models \varphi \leftrightarrow \psi && \Leftrightarrow && \mathscr{A}, \vartheta \models \varphi \text{ iff } \mathscr{A}, \vartheta \models \psi \\
\mathscr{A}, \vartheta &\models \exists x\, \varphi && \Leftrightarrow && \text{there is } u \in U \text{ such that } \mathscr{A}, \vartheta[x \mapsto u] \models \varphi \\
\mathscr{A}, \vartheta &\models \forall x\, \varphi && \Leftrightarrow && \text{for all } u \in U \text{ we have } \mathscr{A}, \vartheta[x \mapsto u] \models \varphi
\end{aligned}$$

where $\vartheta[x \mapsto u]$ denotes the update of ϑ at position x with value u.

A FOL$[=, \tau]$-formula φ is *valid* iff for all τ-interpretations $(\mathscr{A}, \vartheta)$ we have $\mathscr{A}, \vartheta \models \varphi$. Examples of valid formulas include

$$\forall y R(z, y) \to \forall y \exists x R(x, y) \quad , \quad \forall x\, f(x) = x \to \forall x\, f(f(x)) = x \quad , \quad \exists x\, (\forall y\, Drinks(y) \to Drinks(x))$$

In the following we will simply speak of First-Order Logic, FOL in short, when we mean First-Order Logic with Equality over a particular signature τ which is derivable from the context. We further assume, that all formulas are *sentences*, i.e. all variables are bound by a quantifier. Note that models for sentences can be given as a τ-structure alone, i.e. no variable assignment is needed.

$$(\wedge_{\mathtt{L}})\ \frac{\Gamma, \varphi, \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta} \qquad (\wedge_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta} \qquad (\neg_{\mathtt{L}})\ \frac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$$

$$(\vee_{\mathtt{L}})\ \frac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta} \qquad (\vee_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi, \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta} \qquad (\neg_{\mathtt{R}})\ \frac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$$

$$(\rightarrow_{\mathtt{L}})\ \frac{\Gamma, \psi \Longrightarrow \Delta \qquad \Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \varphi \rightarrow \psi \Longrightarrow \Delta} \qquad (\rightarrow_{\mathtt{R}})\ \frac{\Gamma, \varphi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \rightarrow \psi, \Delta}$$

Figure 1: The proof rules for Boolean operators.

2.2 Sequents and Validity

The Sequent Calculus is a formal system with which one can derive the validity of a formula by purely symbolic formula manipulation. The notion of validity of a formula is suitably generalised in order to enable the application of simple logical principles in the form of proof rules. The basic data structure for this purpose is that of a *sequent* – a pair of finite multisets of formulas written $\Gamma \Longrightarrow \Delta$. The left part Γ is called the *antecedent*; the right part Δ is called the *succedent* of the sequent.

Such a sequent is *valid* if the formula $(\bigwedge\Gamma) \rightarrow \bigvee\Delta$ is valid. Hence, in a sequent $\varphi_1, \ldots, \varphi_n \Longrightarrow \psi_1, \ldots, \psi_m$, the comma separating different elements of the (multi-)set is interpreted as a conjunction in the antecedent and as a disjunction in the succedent. Note that validity of finite sequents does indeed generalise validity of formulas since the formula φ is valid iff the sequent $\emptyset \Longrightarrow \varphi$ is valid. Likewise, validity of finite sequents can be expressed as validity of a formula as it is done in the definition here.

A sequent $\Gamma \Longrightarrow \Delta$ of τ-formulas is consequently *invalid* if there is a τ-interpretation $(\mathscr{A}, \vartheta)$ that fulfils all formulas of the antecedent but none of the succedent, i.e. $\mathscr{A}, \vartheta \models \varphi$ for all $\varphi \in \Gamma$ and $\mathscr{A}, \vartheta \not\models \psi$ for all $\psi \in \Delta$. Such an interpretation is also called a *countermodel* for $\Gamma \Longrightarrow \Delta$.

2.3 Formal Proofs

The Sequent Calculus is a formal proof system that characterises the semantic notion of validity of sequents (and therefore validity of formulas) through the existence of a purely syntactic object, namely a *formal proof* for a sequent $\Gamma \Longrightarrow \Delta$ under consideration. Such a formal proof is a finite tree whose nodes are labeled with sequents, such that

- the tree's root is labeled with $\Gamma \Longrightarrow \Delta$, and
- the labels at each node, together with the labels on its children, form a substitution instance of one of the proof rules depicted in Figures 1–3.

We let proof trees grow upwards, i.e. the sequent under consideration is shown at the bottom of the tree.

Rules are of the form

$$(N)\ \frac{\Gamma_1 \Longrightarrow \Delta_1 \quad \ldots \quad \Gamma_n \Longrightarrow \Delta_n}{\Gamma \Longrightarrow \Delta}$$

for some $n \in \{0, 1, 2\}$. (N) is the rule's name simply used to identify it when reasoning about proofs. Each $\Gamma_i \Longrightarrow \Delta_i$ is called a *premiss* of the rule, and $\Gamma \Longrightarrow \Delta$ is called the *conclusion*. Note that some rules have no premisses – they are called *axioms* – and they are the only rules that can be used to close a branch of a proof tree.

$$(\exists_{\mathtt{L}})\ \frac{\Gamma, \varphi[c/x] \Longrightarrow \Delta}{\Gamma, \exists x\, \varphi \Longrightarrow \Delta}\ c \text{ fresh} \qquad (\exists_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi[t/x], \Delta}{\Gamma \Longrightarrow \exists x\, \varphi, \Delta}\ t \text{ ground} \qquad (\mathtt{Contr}_{\mathtt{L}})\ \frac{\Gamma, \varphi, \varphi \Longrightarrow \Delta}{\Gamma, \varphi \Longrightarrow \Delta}$$

$$(\forall_{\mathtt{L}})\ \frac{\Gamma, \varphi[t/x] \Longrightarrow \Delta}{\Gamma, \forall x\, \varphi \Longrightarrow \Delta}\ t \text{ ground} \qquad (\forall_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi[c/x], \Delta}{\Gamma \Longrightarrow \exists x\, \varphi, \Delta}\ c \text{ fresh} \qquad (\mathtt{Contr}_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi, \varphi, \Delta}{\Gamma \Longrightarrow \varphi, \Delta}$$

$$(\mathtt{Subst}_{\mathtt{L}})\ \frac{\Gamma, \varphi[s'/x] \Longrightarrow \Delta}{\Gamma, s = s', \varphi[s/x] \Longrightarrow \Delta} \qquad (\mathtt{Subst}_{\mathtt{R}})\ \frac{\Gamma \Longrightarrow \varphi[s'/x], \Delta}{\Gamma, s = s' \Longrightarrow \varphi[s/x], \Delta} \qquad (\mathtt{Eq}_{\mathtt{L}})\ \frac{\Gamma, s = s \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

Figure 2: The proof rules for quantifiers and equalities.

$$(\mathtt{Ax})\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta} \qquad (\mathtt{Eq}_{\mathtt{R}})\ \frac{}{\Gamma \Longrightarrow s = s, \Delta}$$

Figure 3: The axioms.

Proof search in the sequent calculus thus amounts to the construction of a finite tree, starting with the sequent to be proved, then selecting rules and applying them to the current sequent. This may create further proof obligations in the form of one or two premisses of the currently applied rule, which then need to be handled in the same way until all created branches of the proof tree are closed by the application of axioms.

The general intuition behind the format of the rules is the following. There is one rule for each potential occurrence of a logical operator at top-level in the antecedent and the succedent. These rules then try to simplify the sequent at hand by eliminating this logical connector. This is at least true for the rules shown in Figure 1, which handle Boolean operators. Note, for instance how the implicitly understood meaning of a sequent – the antecedent is interpreted conjunctively, the succedent disjunctively – is used by rules $(\wedge_{\mathtt{L}})$ and $(\vee_{\mathtt{R}})$ to eliminate the occurrence of a conjunction in the antecedent or a disjunction in the succedent.

Conversely, a conjunction in a succedent can be handled using rule $(\wedge_{\mathtt{R}})$ which creates two "smaller" sequents to be proved. This rule incorporates the distribution law for conjunctions and disjunctions by turning a statement about a disjunction including a conjunction into two statements about disjunctions that both need to be fulfilled.

Quantifiers are handled in a similar way, in that they are also eliminated in an application of the respective rules as they are shown in Figure 2. It is important to obey the requirements that in an application of rule $(\exists_{\mathtt{L}})$ or $(\forall_{\mathtt{R}})$, the respective variable is replaced by a fresh constant symbol c which does not occur anywhere in the sequent of the conclusion. Such constants are also sometimes referred to as *Skolem constants*. Moreover, rules $(\forall_{\mathtt{L}})$ and $(\exists_{\mathtt{R}})$ require the replacement of the respective variable by a ground term, not an arbitrary term. It is not hard to construct examples of non-valid formulas which would be provable without these requirements.

On the other hand, one can equally construct examples of valid formulas which are not provable with those rules alone. Sometimes, the formal proof of a statement from some assumption requires some assumption to be used more than once, in particular when it is a universally quantified formula. This is

$$\dfrac{\dfrac{}{a = f(c), E(a,c), E(b,c) \implies a = f(c)}\,(\mathtt{Ax})}{\dfrac{a = f(c), b = f(c), E(a,c), E(b,c) \implies a = b}{(**)}\,(\mathtt{Subst_R})}$$

$$\dfrac{\dfrac{}{a = f(c), E(a,c), E(b,c) \implies E(b,c), a = b}\,(\mathtt{Ax}) \qquad (**)}{\dfrac{a = f(c), E(b,c) \to b = f(c), E(a,c), E(b,c) \implies a = b}{(*)}\,(\to_{\mathtt{L}})}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{E(b,c) \to b = f(c), E(a,c), E(b,c) \implies E(a,c), a = b}\,(\mathtt{Ax}) \qquad (*)}{E(a,c) \to a = f(c), E(b,c) \to b = f(c), E(a,c), E(b,c) \implies a = b}\,(\to_{\mathtt{L}})}{E(a,c) \to a = f(c), \forall y.\, E(b,y) \to b = f(y), E(a,c), E(b,c) \implies a = b}\,(\forall_{\mathtt{L}})}{E(a,c) \to a = f(c), \forall x \forall y.\, E(x,y) \to x = f(y), E(a,c), E(b,c) \implies a = b}\,(\forall_{\mathtt{L}})}{\forall y.\, E(a,y) \to a = f(y), \forall x \forall y.\, E(x,y) \to x = f(y), E(a,c), E(b,c) \implies a = b}\,(\forall_{\mathtt{L}})}{\forall x \forall y.\, E(x,y) \to x = f(y), \forall x \forall y.\, E(x,y) \to x = f(y), E(a,c), E(b,c) \implies a = b}\,(\mathtt{Contr_L})}{\forall x \forall y.\, E(x,y) \to x = f(y), E(a,c), E(b,c) \implies a = b}\,(\wedge_{\mathtt{L}})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x \forall y.\, E(x,y) \to x = f(y), E(a,c) \wedge E(b,c) \implies a = b}{\forall x \forall y.\, E(x,y) \to x = f(y) \implies E(a,c) \wedge E(b,c) \to a = b}\,(\to_{\mathtt{R}})}{\forall x \forall y.\, E(x,y) \to x = f(y) \implies \forall z.\, E(a,z) \wedge E(b,z) \to a = b}\,(\forall_{\mathtt{R}})}{\forall x \forall y.\, E(x,y) \to x = f(y) \implies \forall y \forall z.\, E(a,z) \wedge E(y,z) \to a = y}\,(\forall_{\mathtt{R}})}{\forall x \forall y.\, E(x,y) \to x = f(y) \implies \forall x \forall y \forall z.\, E(x,z) \wedge E(y,z) \to x = y}\,(\forall_{\mathtt{R}})$$

Figure 4: A formal proof for the sequent given in Example 1.

what rule $(\mathtt{Contr_L})$ is for. Likewise – but perhaps less obviously – one may need several copies of an existentially quantified formula in the succedent of a sequent which can be obtained with rule $(\mathtt{Contr_R})$.

At last, equality possesses some fundamental principles which require three more rules which break the symmetry that is present in the set of rules introduced so far. $(\mathtt{Subst_L})$ and $(\mathtt{Subst_R})$ can be used to replace a term by another, provided that their equality is assumed, i.e. is part of the sequent's antecedent. Note that these rules are only applicable if the replacement of term s by term s' in φ can be carried out harmlessly, i.e. no new variable bindings are created in this way. Finally, equality is reflexive and it is therefore always possible to assume that some term equals itself using rule $(\mathtt{Eq_L})$.

At last, there are two axioms shown in Figure 3. Their intuition is easily derived from the properties of equality and the meaning of a sequent. Axiom $(\mathtt{Ax})$ for instance allows a branch of a formal proof to be closed when antecedent and succedent of the current sequent contain a common formula. Such a sequent is clearly valid, given the conjunctive interpretation of the antecedent and the disjunctive interpretation of the succedent. I.e. when every formula in the antecedent is satisfied, and one of them also occurs in the succedent, then some formula of the succedent is also satisfied. The other axiom incorporates the reflexivity principle of equality: when the goal is to show that some term equals itself or something else holds, then nothing more is to be proved in fact.

Example 1 Figure 4 depicts a formal proof of the valid sequent

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

Intuitively, this sequent formalises the following assertion: if the inverse of the relation E is included in the function f then no element z can have two different predecessors in the relation E.

In order to formally prove this valid statement using the sequent calculus, can proceed as follows.[1] First we introduce fresh constants a,b,c for the universally quantified variables in the antecedent using rule $(\forall_{\mathtt{R}})$ thrice. This amounts to showing that

$$E(a,c) \wedge E(b,c) \to a = b$$

holds for arbitrary a,b and c, provided that the antecedent – untouched up until then – holds, too. It allows us to use an implicit property of functions, namely that they map values to values uniquely, for the relation E. However, we need to use the assumption twice, once for a and c but also for b and c. This is why the proof uses rule $(\mathtt{Contr}_{\mathtt{L}})$ and duplicates this assumption before we instantiate the universally quantified formulas in the antecedent with these two pairs of constants (which are now available as ground terms). This is an insight one needs to have at this point for otherwise one will not be able to close the proof. Note that the application of rule $(\mathtt{Contr}_{\mathtt{L}})$ is not driven by any syntactical need like a top-level operator in the sequent.

The rest of the proof – which is broken into three parts in order to fit the page width here – uses a bit of simple Boolean reasoning eliminating the implication connectors, followed by very simple equational reasoning basically implementing a standard pattern to show that equality is transitive using rule $(\mathtt{Subst}_{\mathtt{R}})$.

2.4 The Sequent Calculus as a Formal Proof System

We briefly discuss two aspects of the sequent calculus. The first one is purely mathematical at first sight and can be phrased as follows.

Proposition 1 ([14, 10]) *The Sequent Calculus is sound and complete with respect to validity of sequents in First-Order Logic with Equality.*

Soundness means that any sequent which can be derived, i.e. for which there is a proof, is indeed valid. In other words, one cannot prove false statements using the sequent calculus. It can be shown by induction on the height of a proof tree, making use of the fact that all the rules are valid in the sense that a conclusion is valid if all its premisses are valid. In the special case of axioms this means that conclusions are valid straight away. Completeness means that any valid sequent is provable. This is more difficult to show, c.f. [10].

These meta-logical principles about the sequent calculus have some effect on didactical questions. Completeness of the calculus guarantees that a proof exists for a sequent which may – by informal reasoning – be seen as valid. So any student's difficulty in being able to construct a proof is down to the student's abilities and experience with this calculus. It does not introduce an additional level of difficulty by requiring a student to understand for which valid sequents formal proofs could be constructed and for which it simply is impossible. Soundness is of course at least equally important as working with an unsound proof calculus would severely impede students' abilities to distinguish valid from invalid statements and therefore also to learn how to read off the intuitive meaning of formulas.

The second aspect worth mentioning here concerns the format of proofs in the sequent calculus as opposed to other proof calculi for First-Order Logic with Equality. The application of a rule in the sequent calculus is purely local, as opposed to natural deduction for instance where rule application can

[1] The proof shown in Figure 4 is not the only but one of the shortest and simplest.

span over the entire history of formal statements from the beginning of the proof attempt. The rules of the sequent calculus carry all assumptions and proof goals through to the premisses such that a decision on which rule to apply next can (at least in principle) be taken purely based on a current sequent alone, disregarding all other sequents of a partially constructed formal proof.

3 Learning the Sequent Calculus

3.1 Syntactic Rule Manipulation vs. Semantic Understanding

A standard exercise in sequent calculus asks for a proof of a given sequent. We reconsider the valid sequent $\mathscr{S}_0$ presented in Example 1 above:

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

The students' difficulties when trying to find a formal proof for such sequents, as the one given in Figure 4 for instance, as well as mistakes frequently made when trying to construct such a proof on paper can generally be put into two categories – the syntactic and semantic ones mentioned in the introduction.

(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 – say a, b, c – and then decompose the Boolean operators on the right side, yielding $\mathscr{S}_1 :=$

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

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 [12].

(2) The second category concerns the semantical understanding of a proof, which we here address as the ability of *finding the right proof*. We exemplify the difficulties in finding a proof for sequent $\mathscr{S}_0$ from above. As mentioned above, one has to start by applying a "purely syntactic" strategy, which eventually yields the sequent $\mathscr{S}_1 :=$

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

At this point, the next step is not obvious. Clearly, based on results on semi-decidability and completeness of the sequent calculus [10] there is a strategy that will always find a proof if one exists: it consists of applying every possible rule instance at some point. This is not a good strategy for students in an exam or homework-assignment, though, as it simply takes far too long to find a proof, and the resulting proofs are too large to be overseen by the human mind. Thus, this generic strategy is hardly useful to yield semantic understanding, let alone solve homework or exam exercises.

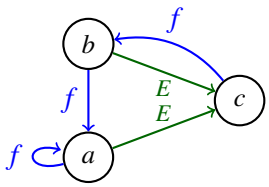

Figure 5: Countermodel for $\mathscr{S}_2$.

A closer look at the sequent can give the right intuition needed to find a short proof. In analogy to the explanations in Section 2.3 the sequent $\mathscr{S}_1$ can be rephrased as follows: *Given a graph $G = (E,V)$, if the inverse of the edge-relation E is functional and there are edges from a to c and b to c, then a has to equal b.*

Now, a naïve approach to prove sequent $\mathscr{S}_1$ is to continue with obvious rule applications like instantiating the universal quantifiers in the premiss, e.g. resulting in the sequent $\mathscr{S}_2 :=$

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

This sequent has a simple countermodel, shown as a directed graph in Figure 5 with edges depicting the relation E in green and the function f in blue.

Thus, the valid sequent $\mathscr{S}_1$ has turned into the invalid sequent $\mathscr{S}_2$ by an application of a particular proof rule, and it should be clear that such a rule application is to be deemed as a bad step in trying to construct a formal proof for a valid sequent.

Actually, the existence of a countermodel to $\mathscr{S}_2$ provides the right intuition to prove sequent $\mathscr{S}_1$. We have to use the premiss $\forall x \forall y.\ E(x,y) \to x = f(y)$ on all edges of the given graph. The advisable next step is therefore to double the formula $\forall x \forall y.\ E(x,y) \to x = f(y)$ using rule ($\texttt{Contr}_\texttt{L}$)and then apply it also to the edge from a to c. The resulting sequent is $\mathscr{S}_3 :=$

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

which is easily proved by decomposing Boolean operators and substituting equal terms using the corresponding rules.

3.2 Tool-Supported Learning

There is a clear dependency between the two challenges mentioned in Section 3.1: the syntactic ones described in (1) need to be met before the semantic ones in (2); 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 mathematics (e.g. in [20]). The phenomenon is accurately described in [13] 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 (SCT) comes into play. It is supposed to aid the students in facing the aforementioned challenges of constructing correct proofs and finding the right proof. SCT therefore adheres to two design principles: it is an easy-to-use and simple assistant for building proof trees in the sequent calculus, while also providing compiler-like feedback on syntactical rule applications which is known for its benefits in tutorial teaching environments [2]. Moreover, it offers an interactive proof mode, meant to guide the students' focus on the underlying semantics of a sequent.

3.3 Related Tools

There are other high quality interactive proof systems that can be used to train the construction of proofs in the sequent calculus. Here we only mention three, which draw our attention through their accessibility and usability.

The tool that meets the prerequisites laid out here best is LOGITEXT[2]; others worth mentioning are JAPE [5] and KEY[3]. One major drawback of the two mentioned first is that they don't treat first-order logic with equality out of the box, though, JAPE offers the possibility to add rules for treating equality to its sequent calculus. Although, the KEY-System is meant as a verification tool for JAVA-Programs it can be used as an interactive prover of sequents in first-order logic.

However, none of those existing tools is general enough to serve the subtle didactical purposes described above – to act as a tool that is meant to trigger semantical understanding in learning how to proof in an error-guided fashion. Furthermore, in a setting where we also 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. Thus, one 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.

4 The Sequent Calculus Trainer

We provide the Sequent Calculus Trainer (SCT) as an open source application under the BSD-3 license. The source code as well as the binaries are publicly available.[4] Figure 6 shows the graphical user interface which is kept fairly simple.

SCT comes 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 text file or via a text field 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.

In the following we introduce the key features of the Sequent Calculus Trainer distinguished by their purpose of helping students develop syntactical understanding and semantical understanding for the task of finding formal proofs in the sequent calculus.

[2] `http://logitext.mit.edu`
[3] `http://www.key-project.org`
[4] `http://www.uni-kassel.de/eecs/fachgebiete/fmv/projects/sequent-calculus-trainer.html`

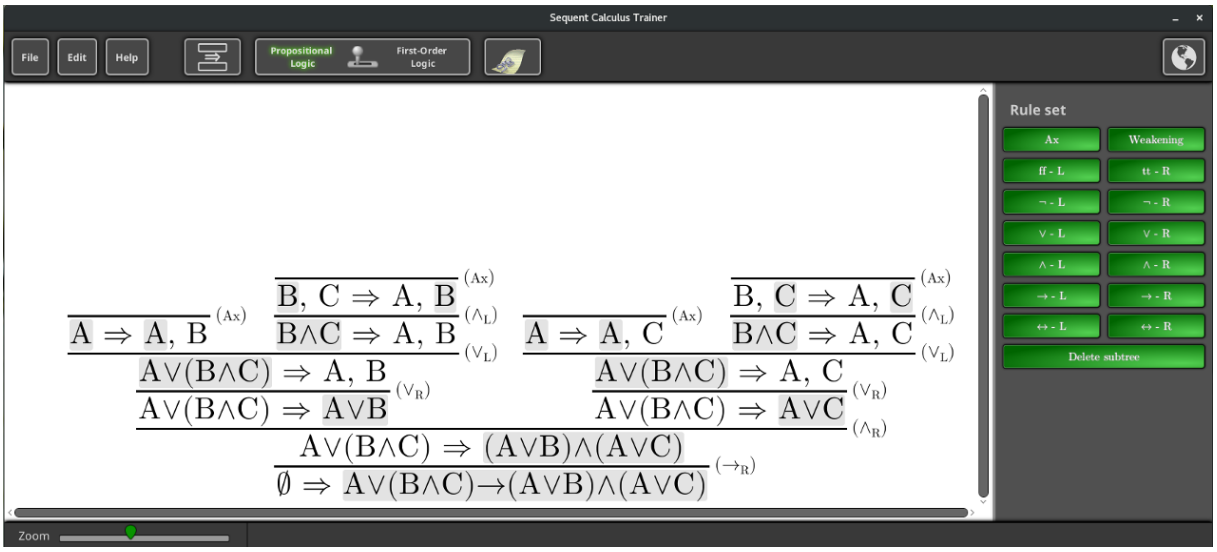

Figure 6: The user-interface.

4.1 Support for Constructing Syntactically Correct Proofs

The Sequent Calculus Trainer can be used as a simple assistant for constructing proof trees, without providing any hints on which rules to apply. In the following we will briefly introduce the two key features of this mode.

- Nearly every user action leads to a response by the program. Figure 7 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 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 proof tree 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 8 shows how the substitution rule works. After the rule for substitution on the left-hand or 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.

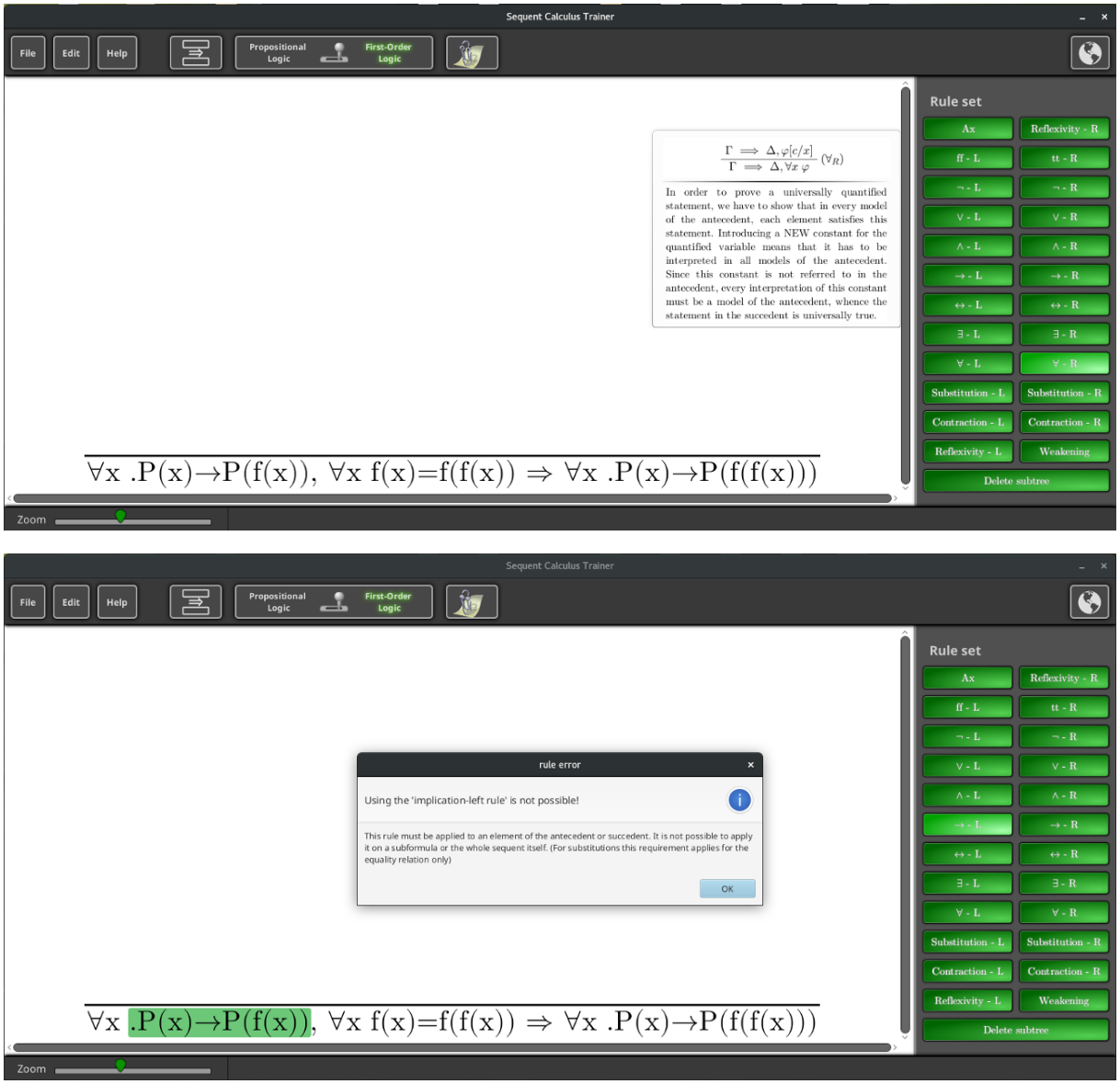

Figure 7: The feedback system.

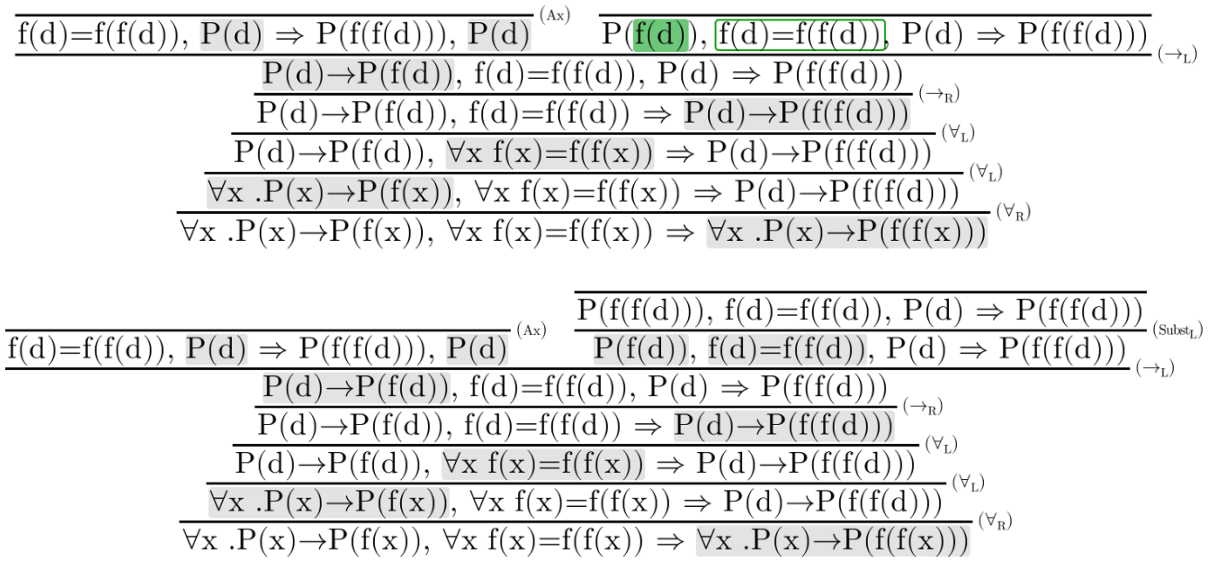

Figure 8: Applying the substitution rule in a proof: before (top) and after (below).

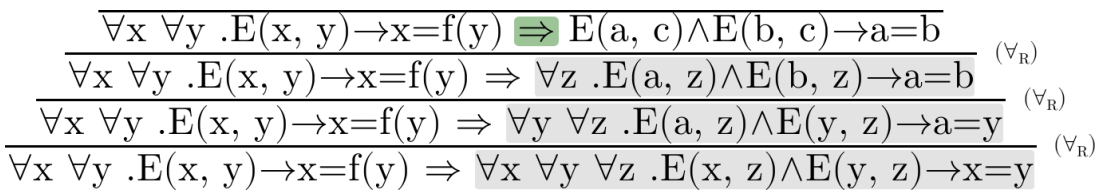

Figure 9: The beginning of a proof for the sequent $\mathscr{S}_1$.

4.2 Support for Finding the Right Proof

The interactive proof mode uses a traffic-light-like system to provide students with feedback to indicate whether the current sequent is valid (green), invalid (red), or of unknown status (yellow). Fig. 9 shows the situation for the sequent $\mathscr{S}_0$ from Section 3.1.

As soon as one quantifier in the premiss is resolved without doubling the premiss first the sequent becomes unprovable. This is announced to the user by turning the indication light red, as shown in Fig. 10.

So the interaction between SCT and a user is thus governed by a very simple protocol: SCT immediately reports to the user what the effect of the last rule application was in terms of chances to finish the formal proof in this way. The information transported through this traffic-light system is the following: as soon as a valid sequent turns into an invalid one there is no hope of completing the proof in this way; so the student has to rethink the last step. In general, he/she cannot simply try out all possible rule applications at this point because there is no a priori bound on

- the number of duplicates one may need using rules ($\texttt{Contr}_\texttt{L}$) or ($\texttt{Contr}_\texttt{R}$), and on
- the number of ground terms that suffice for instantiations using rules ($\forall_\texttt{L}$) and ($\exists_\texttt{R}$).

Furthermore, as it is clear that no syntactic mistake has been made at this point the best strategy for completing a proof is to understand what went wrong in the rule application that turned a green light

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall y\ .E(a,\ y)\to a=f(y) \Rightarrow E(a,\ c)\wedge E(b,\ c)\to a=b}}{\forall x\ \forall y\ .E(x,\ y)\to x=f(y) \Rightarrow E(a,\ c)\wedge E(b,\ c)\to a=b}\,(\forall_L)}{\forall x\ \forall y\ .E(x,\ y)\to x=f(y) \Rightarrow \forall z\ .E(a,\ z)\wedge E(b,\ z)\to a=b}\,(\forall_R)}{\forall x\ \forall y\ .E(x,\ y)\to x=f(y) \Rightarrow \forall y\ \forall z\ .E(a,\ z)\wedge E(y,\ z)\to a=y}\,(\forall_R)}{\forall x\ \forall y\ .E(x,\ y)\to x=f(y) \Rightarrow \forall x\ \forall y\ \forall z\ .E(x,\ z)\wedge E(y,\ z)\to x=y}\,(\forall_R)$$

Figure 10: Creating an invalid sequent by unwise instantiation.

red. This initiates a process of deeper semantical understanding, for instance by forcing the student to consider a counter model for the invalid sequent or understand the intuitive reason for the validity of the implication present in the previous sequent.

For valid sequents the Sequent Calculus Trainer also possesses a mode in which information on which rule to apply next is provided to the students.

5 Behind the Scenes

The Sequent Calculus Trainer is a Java implementation; this guarantees accessibility for a wide range users. It uses the GUI framework JavaFX[5] which is integrated in the Java Standard Library since the emerging of Oracle's Java 8.

In the following, we will describe the main ideas behind automated reasoning algorithm used to determine the display colour for a sequent in SCT's interactive helper mode. Remember that validity for sequents of First-Order Logic is undecidable (c.f. [10]). Hence, there is no obvious algorithm – let alone any – for determining validity and turning its answer into either of the colours red or green. On the other hand, the validity problem is semi-decidable (c.f. [10]) which is typically shown in a way that boils down to arguing that the set of finite proof trees is recursively enumerable. However, a brute-force method of enumerating all proof trees, suitably combined with a time-out, hardly yields a practical procedure for displaying the validity status of a sequent through the colours red, green and yellow. The didactical purpose that SCT serves requires the validity test to be efficient and yield the colour yellow in as few cases as possible, for otherwise SCT faces the danger of not being accepted by students as a helpful tool.

The main techniques used in the implementation of the validity test concern an efficient instantiation of quantified variables followed by a decision procedure for the validity of quantifier-free first-order formulas with equality. The latter problem is well-known to be decidable, for instance using a result on the decidability of the congruence closure of equality terms (e.g. in [3]). Hence, we focus on the instantiation problem here. Figure 11 shows the algorithm's general structure. A comprehensive reflection of all techniques used can be found in [11].

The principle idea underlying the validity test uses the following connection between invalidity of sequents and satisfiability of formulas:

$$\mathscr{S} = \Gamma \Longrightarrow \Delta \text{ invalid} \quad \Longleftrightarrow \quad \varphi_{\mathscr{S}} := \bigwedge_{\psi\in\Gamma} \psi \wedge \bigwedge_{\psi'\in\Delta} \neg\psi' \text{ is satisfiable.}$$

Hence, it is principally possible to use a satisfiability solver for First-Order Logic for this task. This solver needs to be able to handle quantifiers and uninterpreted function symbols. The former is supported by

[5] `https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm`

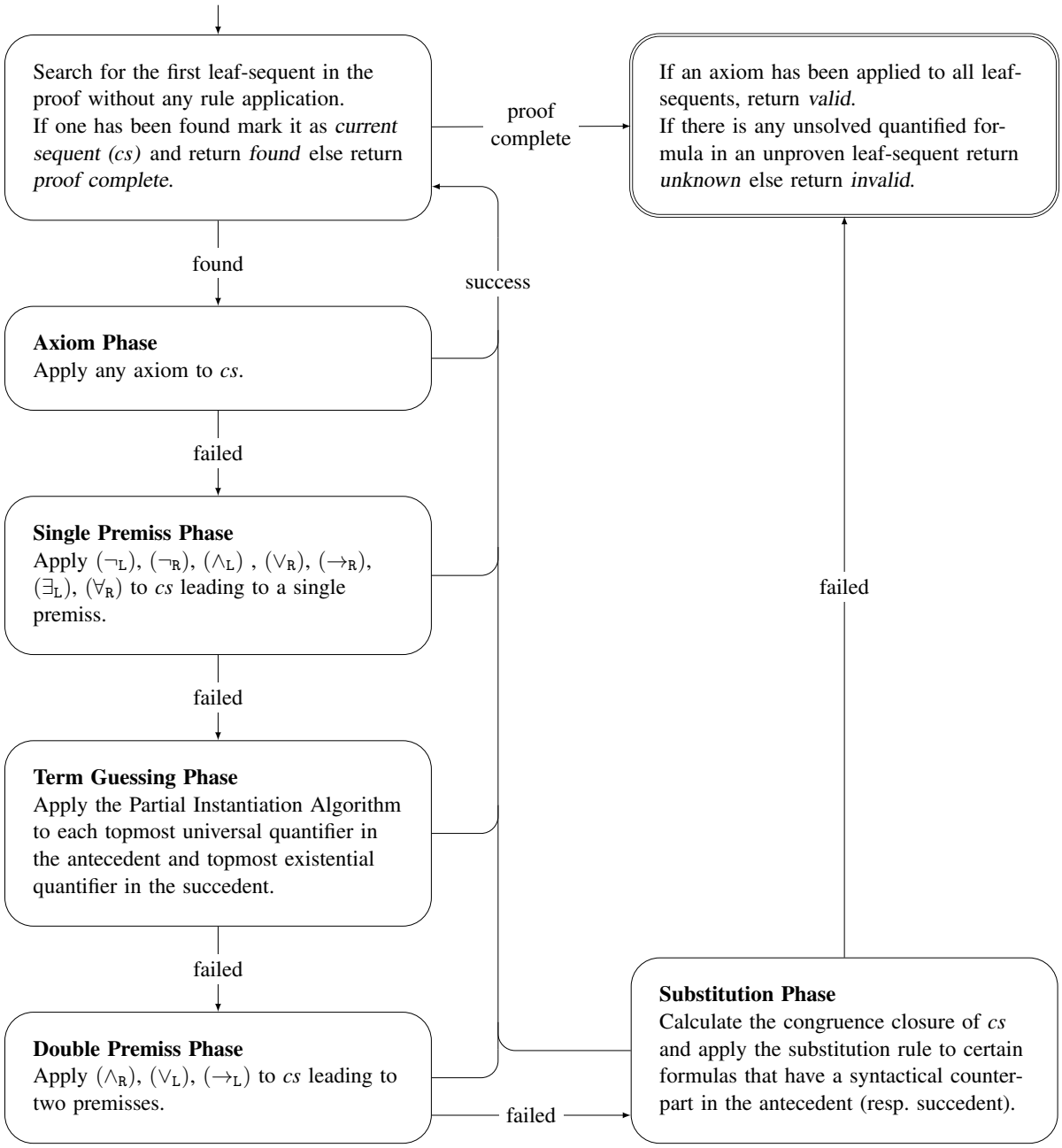

Figure 11: Partial Instantiation Algorithm.

some, the latter by most modern SMT solvers.[6] We make use of Microsoft Research's high performing solver Z3 [7] in version 4.5.0.

However, there is an apparent mismatch in using this reduction. A satisfiability solver like SMT either reports unsatisfiability, a time-out or satisfiability. In the latter case, Z3 produces a model for the formula; in the former case no witness for unsatisfiability is given, though. This does not go well with the requirements laid out for SCT's interactive helping mode (and also has to do with results on the semi-decidability of validity, resp. of satisfiability over finite models for FOL): it is the case of validity in which a witness would be needed in order to turn this into a hint at which rule to apply next and how to do so. The witness obtained from the SMT solver in case of satisfiability can of course only be used for a hint about invalidity; in fact, such a witness reported by the SMT solver is simply a counter model for the sequent.

In order to facilitate a hint at which rule to apply to a valid sequent next we use the following trick. A quantified variable (universally in the antecedent or existentially in the succedent) is instantiated with a term that is not a ground term. We call such an instantiation *partial*. Note that any non-ground term can be seen as a partially constructed ground term; those parts which have not been constructed yet are being abbreviated by a variable. By successively refining partial instantiations it is possible to approximate ground terms as the following simple example shows. The procedure building on this principle in order to test sequents for validity – the Partial Instantiation Algorithm – is presented in Figure 11.

Example 2 Consider the valid sequent

$$\forall y.P(g(e,f(y))),Q(f(c)),R(h(d,c),e) \implies \exists x\, P(x)\ .$$

The last two formulas of the antecedent clearly do not contribute to the validity of this sequent; they are added just to blow up the underlying signature which helps to exemplify the problem with an approach based on a brute-force enumeration of all ground terms. Note that this sequent can be proved using rule $(\exists_{\mathtt{R}})$ with the instantiation $x \mapsto g(e,f(d))$ for example, followed by using rule $(\forall_{\mathtt{L}})$ with the instantiation $y \mapsto d$. In fact, any instantiation $x \mapsto g(e,f(t))$ and $y \mapsto t$ is sufficient for any ground term t.

Consider the following enumeration of all ground terms over the signature $\langle f^{(1)}, g^{(2)}, h^{(2)}, c^{(0)}, d^{(0)}, e^{(0)}\rangle$ which is obtained by using a standard diagonal search through the multi-dimensional space of all ground terms:

$$c,d,e,f(c),f(d),g(c,c),f(e),g(d,c),h(c),g(e,c),h(d),f(f(c)),h(e),\ldots$$

The first term that can be used to instantiate x is $g(e,f(c))$, and it occurs in position 1049 of this enumeration. Even though Z3 can speedily detect unsatisfiability of the instantiated formula

$$\forall y.P(g(e,f(y))) \wedge Q(f(c)) \wedge R(h(d,c),e) \wedge \neg P(t) \tag{1}$$

for any ground term t that is not of the form $g(e,f(\ldots))$ this is not in any way efficient since the overhead for building formulas and calling Z3 costs too much time.

Partial instantiation tries to find the correct instantiation value for variables like x in the previous example by building such terms in a goal-directed fashion. Instead of going through all ground terms according to some fixed enumeration we consider the variable at hand as an already partially instantiated term. Then we take its subterm which is highest in the syntax tree and consists of a variable (i.e. at the

[6] ... even though the name *Satisfiability Modulo Theories* [4] indicates that they were predominantly designed to solve satisfiability problems for formulas over interpreted function symbols like addition on numbers for instance.

beginning the term itself) and replace it successively by terms of the form $f(x_1,\ldots,x_n)$ for some n-ary function symbol f and variables $x_1,\ldots,x_n$. This way, the correct term for the instantiation is found by successive refinement.

Example 3 (Continued) Using partial instantiation on the example formula above the SMT solver Z3 is successively asked to check satisfiability of the formula (1) for the following partially instantiated values of t:

$$z, f(z), g(z,z'), h(z), f(f(z)), f(g(z,z')), f(h(z)), g(f(z),z'), g(g(z,z''),z'), g(h(z),z'), g(z,f(z')),\ldots$$

where z,z',z'' are variables. Note how they are being obtained: starting with the maximally partially instantiated term z, we replace its top-most variable by $f(z)$, $g(z,z')$ and $h(z)$ since f,g,h are the only three function symbols in the underlying signature. The next three terms are obtained by replacing z in $f(z)$ with these three again. Then we get 9 more terms obtained by refining $g(z,z')$, replacing each z and z' with either of these three, and so on.

As one can see, the (partially instantiated) term $g(z,f(z'))$ that leads to unsatisfiability of (1) occurs in position 11 of this list. It is then not hard to see that a proper ground term can be obtained from a partially instantiated term that causes unsatisfiability, by replacing its variables with some combination of constants. Taking this into account, the first term that can be used to prove validity using $(\exists_{\mathtt{R}})$ is found after trying out 74 instantiations only. Hence, only a fraction of the number of calls to Z3 is needed – at least in this case – in comparison to the brute-force method of enumerating all ground terms.

Once Z3 reports unsatisfiability in this way, a term has been constructed by successive refinements, and this term can be used to give the student a hint as to how to use rule $(\exists_{\mathtt{R}})$(and $(\forall_{\mathtt{L}})$in general, too).

Obviously, the chance of finding a proof depends on the success rate of the used SMT solver. Tests show that the running time of Z3 increases especially with the number of different function symbols and the alternation of these symbols in terms. The smallest example we have found that makes Z3 fail to compute an answer is the formula $\forall x\, P(f(g(f(x)))) \wedge \neg P(f(g(h(c))))$. Such cases are handled in the implementation by a fixed timeout; the GUI's feedback to the user is the yellow marker then indicating an unknown validity statement of the sequent at hand.

6 Conclusion and Future Work

In [12] we presented some statistical findings hinting that already the first version of the Sequent Calculus Trainer led to a significant improvement of student's learning outcomes in constructing correct proofs. Although the extension introduced here has not yet been tested in the setting of a major course on formal logic, we strongly believe that the current enhancement will initiate the desired thought processes which are needed for understanding the semantic structure behind a proof.

The description of the Partial Instantiation Algorithm gives rise to a more or less obvious future extension. Remember that this procedure is used to derive feedback in terms of a hint for the user in cases when he/she is faced with a valid sequent (marked green) but is clueless as to which rule to apply next. Equally, the user should receive feedback when he/she uses a rule that does not lead to a completed proof because it creates an invalid sequent (marked red). Since invalidity is detected via satisfiability of the transformed formula, any model generated by the underlying SMT solver Z3 can be presented to the user as a counter model. For didactic purposes, Z3's output format would have to be turned into a suitable presentation of such a counter model, for instance a graphical one. A particular challenge would be handling infinite counter models.

However, simply giving the user a counter model to convince him/her that the last step was bad may be a step too big for students who are struggling with the problem of finding the right proof. It seems like such convincing would have to take smaller steps and require more effort from the students themselves. This would also be further in line with SCT's design principles; the tool can be seen as a first and major step in developing e-learning software for teaching formal reasoning that is based on the well-known didactic principle called "Socratic Method/Dialogue".

There, learning is understood as a process driven by a teacher asking the right questions. Therefore further developments of the Sequent Calculus Trainer will include techniques that resemble the mentioned dialogue. For instance, the software can be extended in a way that requires the user to present a counter model and then to use game-based model checking techniques [21] in order to convince him/her that it does indeed falsify the sequent at hand.

Further extensions, which are not too surprising, concern the actual implementation. First, there is no need to restrict SCT's power to provide hints by the abilities of one particular SMT solver. There are others like Yices [9], CVC4 [8], etc. Judging which one is best for the purposes here is beyond the scope of this work and also not easy to answer. It is also not necessary as one could run several SMT solvers in parallel to check sequents for validity and only report an unknown status when all of them have timed out, as seen in KEY (c.f. Section 3.3).

Another piece of further work concerning SCT's implementation is a transformation into a browser-runnable application since this would simplify the running of the tool, and in particular make it more accessible for mobile devices.

References

[1] ACM/IEEE (2013): *Computer Science Curricula 2013 — Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Available at `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

[2] J. R. Anderson, A. T. Corbett, K. R. Koedinger & R. Pelletier (1995): *Cognitive Tutors: Lessons learned*. *J. of the Learning Sciences* 4(2), pp. 167–207, doi:10.1207/s15327809jls0402_2.

[3] F. Baader & T. Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9781139172752.

[4] C. W. Barrett, R. Sebastiani, S. A. Seshia & C. Tinelli (2009): *Satisfiability Modulo Theories*. In: *Handbook of Satisfiability*, *Frontiers in Artificial Intelligence and Applications* 185, IOS Press, pp. 825–885, doi:10.3233/978-1-58603-929-5-825.

[5] R. Bornat & B. Sufrin (1999): *Animating Formal Proof at the Surface: The Jape Proof Calculator*. *Comput. J.* 42(3), pp. 177–192, doi:10.1093/comjnl/42.3.177.

[6] M. Davis & H. Putnam (1960): *A Computing Procedure for Quantification Theory*. *J. ACM* 7, pp. 201–215, doi:10.1145/321033.321034.

[7] L. De Moura & N. Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08*, LNCS, Springer, pp. 337–340, doi:10.1007/978-3-540-78800-3_24.

[8] M. Deters, A. Reynolds, T. King, C. W. Barrett & C. Tinelli (2014): *A tour of CVC4: How it works, and how to use it*. In: *Proc. 14th Int. Conf. in Formal Methods in Computer-Aided Design, FMCAD'14*, IEEE, p. 7, doi:10.1109/FMCAD.2014.6987586.

[9] B. Dutertre (2014): *Yices 2.2*. In: *Proc. 26th Int. Conf. on Computer Aided Verification, CAV'14*, *LNCS* 8559, Springer, pp. 737–744, doi:10.1007/978-3-319-08867-9_49.

[10] H.-D. Ebbinghaus, J. Flum & W. Thomas (1994): *Mathematical Logic*, 2nd edition. Undergraduate Texts in Mathematics, Springer, Berlin, doi:10.1007/978-1-4757-2355-7.

[11] A. Ehle (2017): *Proof Search in the Sequent Calculus for First-Order Logic with Equality*. Master's thesis, Universität Kassel. Available via `https://www.uni-kassel.de/eecs/?id=46992`.

[12] A. Ehle, N. Hundeshagen & M. Lange (2015): *The Sequent Calculus Trainer - Helping Students to Correctly Construct Proofs*. In: *Proc. 4th Int. Conf. on Tools for Teaching Logic, TTL'15*, pp. 35–44. Available via `http://arxiv.org/abs/1507.03666`.

[13] O. Gasquet, F. Schwarzentruber & M. Strecker (2011): *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*, *LNCS* 6680, Springer, pp. 85–92, doi:10.1007/978-3-642-21350-2_11.

[14] G. Gentzen (1935): *Untersuchungen über das Logische Schliessen I.* *Mathematische Zeitschrift* 39, pp. 176–210, doi:10.1007/BF01201353.

[15] K. Gödel (1930): *Über die Vollständigkeit des Logikkalküls*. Ph.D. thesis, University of Vienna.

[16] Gesellschaft f. Informatik e.V. (2016): *Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen*. Available at `https://www.gi.de/fileadmin/redaktion/empfehlungen/GI-Empfehlungen_Bachelor-Master-Informatik2016.pdf`.

[17] S. Jáskowski (1934): *On the rules of suppositions in formal logic*. *Studia Logica* 1, pp. 5–32.

[18] M. J. Lage, G. J. Platt & M. Treglia (2000): *Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment*. *J. of Economic Education* 31(1), pp. pp. 30–43, doi:10.2307/1183338.

[19] J. A. Robinson (1965): *Machine-oriented logic based on resolution principle*. *Journal of the ACM* 12, pp. 23–41, doi:10.1145/321250.321253.

[20] B. Shneiderman & R. E. Mayer (1979): *Syntactic/semantic interactions in programmer behavior: A model and experimental results*. *Int. J. of Parallel Programming* 8(3), pp. 219–238, doi:10.1007/BF00977789.

[21] C. Stirling (1997): *Games for bisimulation and model checking*. Notes for Mathfit instructional meeting on games and computation, Edinburgh.

[22] M. E. Szabo, editor (1969): *The Collected Papers of Gerhard Gentzen*. Studies in Logic and The Foundations of Mathematics, North-Holland Publishing Company, doi:10.2307/2272429.

[23] K. Weber & L. Alcock (2004): *Semantic and Syntactic Proof Productions*. *Educational Studies in Mathematics* 56(2), pp. 209–234, doi:10.1023/B:EDUC.0000040410.57253.a1.