AXolotl: A Self-study Tool for First-order Logic

David M. Cerna
Research Institute for Symbolic Computation (RISC),
Institute of Formal Methods and Verification (FMV),
Johannes Kepler University, Linz, Austria,
Email: David.Cerna@risc.jku.at

Abstract—We introduce AXolotl, a self-study aid designed to
guide students through one of the more strenuous topics in a
typical course on formal logic, that is quantification. While the
formal inferences for quantification are quite simple and are thus
easy absorbed by students, the properties these simple inferences
entail when used in conjunction with the other inferences usually
leaves students flummoxed. Our software isolates certain aspects
of quantification which are particularly difficult for students,
namely term unification. This is done by restricting the proof
and formula structure allowed in a derivation. Here we illustrate
the initial results of an ongoing investigation, which includes the
development of more comprehensive self-study software.

Keywords—Education; Logic; Quantification; Unification

I. INTRODUCTION

Logic has, over the past century, moved from an esoteric
subject studied and used, in its abstract form, by the few, to
a subject pervasive in the modern world. This pervasiveness
is mostly due to ubiquity of computing technology within
modern society, the foundations of which rest in the realm
of mathematical logic. With this in mind, one would expect
formal logic to encompass a significant portion of undergraduate
computer science education, however, this is unfortunately
not the case [12]. Part of this problem seems to be the
tremendous gap between logic studied in the abstract form
and its application within computing technology.

While many can see the importance of understanding
Boolean algebra when one wants to write a correct if state-
ment, the formal theory of propositional logic seems far
removed from issues like program failure or buggy software,
yet it is precisely in these situations where it has helped
Helping students see these connections is obviously beyond
the scope of this work, rather we focus on the problem of
helping students feel comfortable with formal systems and the
required manipulations therein.

We are not the first to notice nor investigate this understand-
ing gap described above. There are even venues dedicated to
the topic, such as Tools for Teaching Logic (TTL) and ThEdu.
Thus, there have been an assortment of proposed solutions over
the past twenty years, however, none of which, to the best of
our knowledge, takes the approach to the problem we discuss
here. While some software does focus on restricted fragments
of formal logic, these fragments are usually too weak or
spurious to be any use to the development of understanding of
formal systems. A good example of such a self-study aid is the
mobile phone app Quantifiers! [5]. While, similar to our work,
it focuses on quantification, the examples of quantification are
spurious, writing the statements without quantification would
be significantly easier. Furthermore, the examples are trivial in
that term construction is never needed, this tends to be one of
the more difficult concepts for students to understand. Other
examples of similar software are Emojiic [9] which completely
abstracts logic away and focuses on image-word associations,
and Lewis Carroll [18] which is written in Scratch [13] and
focuses on natural language syllogisms.

As one may expect from a subject with its roots in philos-
ophy, much of the existing software is aimed at philosophy
education. For example, the mobile app Andor [10] focuses
on the understanding of natural language statements logically.
This is one type of exercise which the interactive textbook
Carnap.io [11] provides. Similarly, Terrance Tao developed an
interactive textbook [19] for understanding the logic behind
mathematical theorems. Integrating both the more natural
language interpretation with mathematical understanding is
tackled by Lurch, a mathematical text typesetter [3] with an
integrated prover.

While the above outline covers many of the outliers con-
cerning self-study logic education software, the majority,
which we have yet to mention, focus on derivation con-
struction and proof assistance. A quite important example
of such derivation construction tools is the Sequent Calculus
Trainer [7] which provides a user-friendly interface for the
construction of sequent calculus proofs as well as a hint engine
powered by the Z3 [4] smt solver. The already mentioned
interactive textbook, carnapi [11] also includes a proof con-
struction interface, but for natural deduction. However, unlike
the Sequent Calculus Trainer which has buttons for each rule
the one requires free-form text input from the user following
a particular style of natural deduction proof representation
(i.e. Finch style, Montague style, etc.). This leaves more
room for error on the students part and furthermore, such
proof representations are not appropriate outside of philosophy
education, thus limiting the applicability of the system.

Some other worthwhile mentions are the mobile app Natural
Deduction [8], NaDea: Natural Deduction assistant [20], The
Incredible Proof Machine [2], and SPA: Students’ Proof Assis-
tant [16]. Unlike most other mobile apps Natural Deduction
provides a natural deduction finch style proof system as well as

This work was funded by the Linz Institute of Technology project
LOGTECHEDU.
a theorem prover. There exist mobile logic assistants which include theorem proving technology, for example Logic++ [22], but they tend to put less focus on the assisting aspect and more on the proving. NaDea is of similar design as Natural Deduction in that the student can construct proofs and have the system prove the statement for them, however, NaDea can also provide hints to the students, essentially guiding them through the proof. The system is implemented on top of Isabelle [15] and thus benefits from its further development and expressive power. Of similar design as NaDea and also implemented in Isabelle is SPA which aids students through the process of developing a proof assistant. It is essentially a proof assistant within a proof assistant.

The final self-study software we will discuss is the incredible proof machine which is a proof construction tool with a novel interface design. Rather than constructing traditional proofs users build circuits which match the inference rules. While this is pretty standard with respect to propositional logic, it provides an intuitive interface for more difficult calculi such as Hilbert Systems [14] and the Lambda Calculus [1].

Concerning our contribution to the plethora of existing software, AXolotl, we focus on the problem of unification within the sequent calculus framework. The users is given a pair of initial terms and using a set of rules must show that there exists a sequence of rule applications to initial terms which result in a proof tree without open branches. Our rule set consist of formula with a universal prefix and a matrix consisting of an implication with a single Atomic formula in its consequent and antecedent. These restrictions guarantee that application of each inference rule results in a derivation with a single axiomatic branch and a single open/axiomatic branch. Thus, the proof state is extremely simple to represent and term construction can be performed in a calculator-like setting similar to the sequent calculus trainer [7].

The rest of the paper is as follows: In Section II we discuss the background knowledge needed for understanding the educational scenarios addressed by our software. In Section III we discuss our implementation of AXolotl and how the software address the educational scenarios discussed in Section II. In Section IV we discuss our plans for extending the addressed educational scenarios and alternative implementations for the software for mobile devices such as tablets and smart phones.

II. PRELIMINARIES

We provide a brief description of a restriction of Gentzen’s Sequent calculus for first-order logic which covers the precise logical problems we address. For a more detailed and complete description please see [17]. Let $\mathcal{P}$ be a countably infinite set of predicate symbols each of which has a unique arity $n \geq 0$ and $\mathcal{X}$ be a countably infinite set of variable symbols. Terms $T$ are constructed from a countably infinite set of function symbols $F$ each of which has a unique arity $n \geq 0$ together with the variables of $\mathcal{X}$ as follows:

- let $x \in \mathcal{X}$, then $x \in T$
- let $a \in F$ such that $a$ has arity 0, then $a \in T$

- let $t_1, \ldots, t_n \in T$ and $f \in F$ s.t. $f$ has arity $n$, then $f(t_1, \ldots, t_n) \in T$

Furthermore, by $v(t)$ for $t \in T$ we denote the variables occurring in $t$. Concerning formula, we restrict ourselves to the operators $\{\forall, \rightarrow\}$ being that we will not require more complex constructions. Formula $\mathcal{F}_{or}$ are constructed as follows:

- let $P \in \mathcal{P}$ of arity $n$ and $t_1, \ldots, t_n \in T$, then $P(t_1, \ldots, t_n) \in \mathcal{F}_{or}$
- let $P(t_1, \ldots, t_n), Q(s_1, \ldots, s_m) \in \mathcal{F}_{or}$ and be atomic, then $P(t_1, \ldots, t_n) \rightarrow Q(s_1, \ldots, s_m) \in \mathcal{F}_{or}$
- let $\forall x_1, \ldots, x_n(P(t_1, \ldots, t_n) \rightarrow Q(s_1, \ldots, s_m)) \in \mathcal{F}_{or}$, $x_0 \in (\bigcup_i(\{v(t_i) \cup v(s_i)\}) \setminus \{x_1, \ldots, x_n\}$, then $\forall x_0, x_1, \ldots, x_n(P(t_1, \ldots, t_n) \rightarrow Q(s_1, \ldots, s_m)) \in \mathcal{F}_{or}$

We will refer to formula $P(t_1, \ldots, t_n) \in \mathcal{F}_{or}$ such that $\bigcup_i v(t_i) = \emptyset$ as terminal Atom and formula $\forall x_0, \ldots, x_n(P(t_1, \ldots, t_n) \rightarrow Q(s_1, \ldots, s_m))\in \mathcal{F}_{or}$ such that $\bigcup_i(v(t_i) \cup v(s_i)) = \{x_1, \ldots, x_n\}$ as rules. Sequents are pairs of multi-sets of formula which will be denoted by uppercase Greek letters as follows: $\Pi \vdash \Delta$. We restrict ourselves to so called implicational sequents which are of the form $A, \Delta \vdash B$ where $A$ and $B$ are terminal atoms and $\Delta$ is a set of rules. The idea behind implicational sequents is that the rules occurring in $\Delta$ can be used to transform $A$ and $B$ until they are equivalent. So far AXolotl uses a simplified version of implicational sequents which we refer to as P-implicational sequents, that is an implicational sequent where the only predicate symbol occurring is $P$.

**Example 1.** The sequent is a $P$-implicational sequent:

$$P(a), \forall x(P(x) \rightarrow P(r(x))) \vdash P(r(a))$$

while the following is only an implicational sequent:

$$P(a), \forall x(P(x) \rightarrow Q(r(x))) \vdash Q(r(a))$$

Given that our concept of well-formed formula is so restricted, we do not need the full expressive power of the Gentzen sequent calculus for first-order logic. The following logical rules suffice:

$$
\frac{A, C, \Delta \vdash B}{A, \Delta \vdash B} \quad \text{c} \quad \frac{A, \Delta \vdash B}{A, C, \Delta \vdash B} \quad \text{w}
$$

$$
\frac{A \vdash C \quad D, \Delta \vdash B}{A, C \vdash D, \Delta \vdash B} \quad \rightarrow 
\frac{A \vdash F(t), \Delta \vdash B}{A, \forall x F(x), \Delta \vdash B} \quad \forall
$$

Given a valid P-implicational sequent $S$ we can construct a proof of $S$ using the above rules. In Figure 1 we provide an example proof of an extension of the first sequent provided in example 1. Notice that while one can apply the rules as one may wish there does exist a particular proof strategy which may be employed to construct a proof of any valid P-implicational sequent $S$ which is as follows:

1) Contract (inference c) the rule $R$ from $\Delta$ which will be applied
2) Apply the $\forall$ inference to $R$ exhaustively choosing terms which match either $A$ or $B$. 

Axiomatic branch. Thus, the proof state is extremely simple to represent with a single axiomatic branch and a single open/axiomatic branch. These restrictions guarantee that application of each inference rule results in a derivation with a single axiomatic branch and a single open/axiomatic branch. Thus, the proof state is extremely simple to represent and term construction can be performed in a calculator-like setting similar to the sequent calculus trainer [7].
3) Apply the → inference to R resulting in two branches $B_1$ and $B_2$.
4) w.l.o.g we assume that $B_1$ contains the same terminal atom in the antecedent and succedent. Thus, we apply weakening (inference w) to $\Delta$ until $\Delta$ is empty.
5) Branch $B_2$ once again contains a P-implicational sequent which we can apply the same procedure to, or $B_2$ also contains the same terminal atom in the antecedent and succedent and thus we apply 4) to $B_2$ as well.

This procedure is precisely what is implemented in AXolotl, albeit in a more visually pleasing way.

### III. IMPLEMENTATION

We implemented AXolotl as a stand-alone Java program with an AWT/Swing user interface (Figure 2). An implementation for implicational sequents is possible, though we have only implemented a version for constructing proofs of P-implicational sequents. We plan to develop a mobile version of the software which will include proof construction for the full implicational fragment. The current version of AXolotl may be found on the Author’s homepage along with a simple manual and example AXolotl files.

In order to use AXolotl for proof construction, an AXolotl file must be loaded which includes the allowed function symbols, variables and rules as well as problems associated with the allowed symbols. See Figure 3 for an example AXolotl file.

![AXolotl User Inference](image)

We can relate the user interface presented in Figure 2 with the procedure outline in Section II which we refer to as Linear Proof Search (or LPS). We can ignore the contraction and weakening aspects of LPS being that they are implicit. However, what is important to take notice of concerning the procedure outlined in Section II is that after one iteration, either both branches are axioms or one of the branches is once again a P-implicational sequent. While this is not necessarily the case being that one can instantiate the quantifiers in such a way that it forces non-axiomatic branching, any valid P-implicational sequent is provable without such double branching. This is precisely the reason we chose this fragment. We enforce this property of LPS within AXolotl by only allowing rule application when the instantiated rule’s antecedent or succedent matches one of the terminal atoms. Furthermore, a beneficial property of LPS concerning user inference design is that the proof state can be represented by a single P-implicational sequent from which one can easily undo the last rule application.

Notice that the above paragraph entails that the implication inference also becomes implicit within the AXolotl-LPS framework being that once a rule is fully instantiated it becomes clear how it ought to be applied and whether or not it can even be applied. Thus, the only part of the proof state a student has to manipulate is the instantiation of the rules, what is referred to as the substitution set in Figure 2.

![Image](image)

---

1 http://www3.risc.jku.at/publications/download/risc_5887/AXolotl.jar
2 http://www3.risc.jku.at/publications/download/risc_5887/Man.pdf
3 http://www3.risc.jku.at/publications/download/risc_5887/axFiles.zip
substitutions in the substitution set, an instantiated form of the rule is displayed in the instantiation window below the problems list. Thus, prior to applying the rule a student can visually check if the application is sensible or not. Notice, that unlike LPS as outlined in Section II, quantifiers need not be instantiated in a particular order, this is an artifact of the sequent calculus and partially due to eigenvariable conditions which do not exist in our restricted logic.

In order to construct substitutions students must construct terms using the drop down menu under the substitution set. This menu is highlighted in Figure 4. The black • denotes the next term to be filled in by the student and the white dot ◦ denotes subsequent holes to be filled. Once the the term in the display to the left of the drop down menu is void of •s and ◦s one can choose a variable and create a substitution. We chose this method of input rather than allowing the students to enter free form text because it minimizes the number of errors on the student’s part, especially errors which have nothing to do with the logical problem they are trying to solve.

Beyond simple examples similar to those presented in Section II and Figure 2, The entirety of propositional logic, including propositional proof trees may be encoded within AXolotl. Figure 4 illustrates the application of modus ponens within the propositional Hilbert system for classical logic.

IV. Future work

In this work we introduced the self-study aid AXolotl which addresses problems students commonly face when first working with quantifiers. Currently AXolotl is released as a java program which can be downloaded from http://www3.risc.jku.at/publications/download/risc_5887/AXolotl.jar. We plan to strengthen the expressive power and extend the proof situations it addresses in future versions, as well as design a smart phone app based on the current implementation. The app version will have a touch interface further simplifying user interaction.

REFERENCES