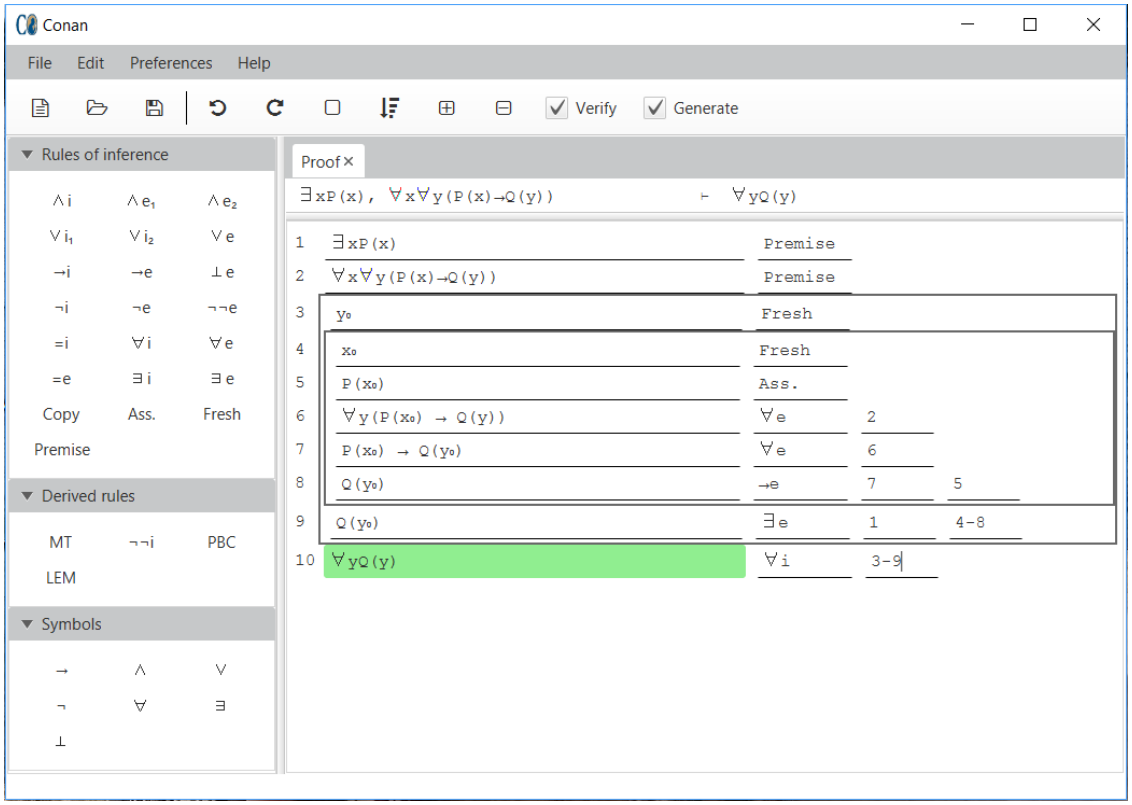
CHALMERS | GÖTEBORGS UNIVERSITET



Proof Editor for Natural Deduction in First-order Logic

The Evaluation of an Educational Aiding Tool for Students Learning Logic

Bachelor's thesis in Computer Science

ELIN BJÖRNSSON, FREDRIK JOHANSSON, JAN LIU, HENRY LY, JESPER OLSSON, ANDREAS WIDBOM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Proof Editor for Natural Deduction in First-order Logic

The Evaluation of an Educational Aiding Tool
for Students Learning Logic

ELIN BJÖRNSSON, FREDRIK JOHANSSON, JAN LIU,
HENRY LY, JESPER OLSSON, ANDREAS WIDBOM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Computer Science
Chalmers University of Technology
Göteborg, Sweden 2017

Proof Editor for Natural Deduction in First-order Logic
The Evaluation of an Educational Aiding Tool for Students Learning Logic
ELIN BJÖRNSSON, FREDRIK JOHANSSON, JAN LIU,
HENRY LY, JESPER OLSSON, ANDREAS WIDBOM

Supervisor: Fredrik Lindblad, Department of Computer Science and Engineering
Examiner: Niklas Broberg, Department of Computer Science and Engineering

Bachelor's Thesis 2017
Department of Computer Science and Engineering
Computer Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualisation of the correct natural deduction steps for the first-order logic proof $\exists x P(x), \forall x \forall y P(x) \rightarrow Q(y) \vdash \forall y Q(y)$, constructed in the developed proof editor.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2017

Proof Editor for Natural Deduction in First-order Logic
The Evaluation of an Educational Aiding Tool for Students Learning Logic
ELIN BJÖRNSSON, FREDRIK JOHANSSON, JAN LIU,
HENRY LY, JESPER OLSSON, ANDREAS WIDBOM
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The subject of this thesis is the presentation and evaluation of *Conan*, an editor for writing natural deduction proofs in first-order logic. The intent is for the editor to serve as a supplementary tool alongside a course in logic. For this reason, emphasis was put on making sure the editor would have a low learning curve, ensuring that learning to use it would not take away from the limited time in a typical university course. Though editors for writing this kind of proof already exist, they are often cumbersome or difficult to use. A pre-study was conducted to determine that there is indeed a lack of editors that fulfil the requirements set forth in this thesis. The interface of Conan was evaluated both heuristically using Jakob Nielsen's heuristics and also through limited user tests. This evaluation suggested that the interface was easy to use, quick to learn and that students were positive toward using the editor for writing proofs. We also present arguments for why Conan would be an aid from the perspective of pedagogy, though no in-depth research on this matter was conducted.

Keywords: Proof editor, First-order logic, Predicate logic, Natural deduction.

Sammandrag

Syftet med den här rapporten är att presentera och utvärdera *Conan*, en bevisredigerare av typen naturlig deduktion för första ordningens logik. Conans syfte är att verka som ett kompletterande verktyg i en logikkurs. Av denna anledning betonades vikten av att redigeraren skulle ha en låg inlärningskurva, för att försäkra att tiden det tar att lära sig verktyget inte tar för mycket från den begränsade tid en typisk unversitetskurs tar. Det existerar sedan tidigare redigerare för att skriva den här typen av bevis, men de upplevs ofta som omständiga eller svåranvända. En förstudie genomfördes för att utvärdera behovet av bevisredigerare som uppfyller de krav som sätts upp i den här rapporten. Conans gränssnitt utvärderades både heuristiskt utifrån Jakob Nielsens heuristiker och genom begränsade användartester. Utvärderingen antyder att gränssnittet är enkelt att använda och lätt att lära sig, samt att studenter är positivt inställda till att använda redigeraren för att skriva bevis. Vi framför även argument för varför den utvecklade redigeraren skulle vara ett hjälpmedel utifrån ett pedagogiskt perspektiv, men ingen djupgående forskning genomfördes på området.

Nyckelord: Beviseditor, Första ordningens logik, Predikatlogik, Naturlig deduktion.

Acknowledgements

We would like to thank Fredrik Lindblad, Koen Lindström Claessen, K. V. S. Prasad, Thierry Coquand, John Hughes and our testers for their contribution to enhancing the end product. Fredrik Lindblad performed excellently as a supervisor. His commitment and responsiveness significantly benefited the team and the end product. Koen Lindström Claessen provided stellar constructive feedback on the process. His assistance greatly helped us clarify our picture of objective. K. V. S. Prasad carefully reviewed both the thesis and the product and he voiced important critique. His time and effort helped us view our work in a more down-to-earth perspective, which made our assessments less bold and optimistic. Thierry Coquand evaluated the proof editor from the perspective of an examiner in a course for logic in computer science. His input was highly valuable and particularly beneficial to future development. John Hughes held an inspiring seminar on the topic of correctness and property based testing. Although, unfortunately not used in this project, his offer of obtaining *QuickCheck* licences was well-received. Finally, we would like to thank our testers for taking the time to evaluate the end product and for the feedback they provided.

Elin Björnsson, Fredrik Johansson, Jan Liu, Henry Ly, Jesper Olsson, Andreas Widbom, Göteborg, May 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Students learning natural deduction in first-order logic are today faced with the choice of using pen and paper or unsatisfactory educational aiding tools which has been concluded by a pre-study (see appendix A). The lack of satisfactory educational aiding tools suggest that improvements can be made to education in logic.

The purpose of this thesis is to evaluate a developed proof editor that is intended to cater to the needs of students learning natural deduction in first-order logic. The most prominent advantage pen and paper have over existing proof editors is that a student does not have to learn how to use pen and paper. The way proofs are edited in the developed proof editor, which we named *Conan*, resemble how proofs are written with pen and paper. Because of this, together with efficient controls and graphical user interface, the time the student has to spend learning how to use Conan is shorter than existing proof editors. Thus enabling the student to focus on their actual studies. Appraisal of Conan was performed on the basis on a number of different criteria (see section 1.1) emanating from the criteria outlined in the pre-study.

1.1 Satisfaction Criteria for the Proof Editor

This thesis aims to answer the following question unambiguously:

> Is Conan a satisfactory educational aiding tool for students learning natural deduction in first-order logic?

The rest of this section is devoted to clarify how this question should be interpreted, starting with the definitions presented below.

Definition 1. A satisfactory educational aiding tool for students learning natural deduction in first-order logic provides the necessary functionality of a proof editor, is easily accessible and is suitable for students.

Definition 1.1. The necessary functionality of a proof editor is the ability to edit proofs, to save and load proofs and to verify the correctness of all deduction steps.

Definition 1.2. A program is easily accessible if it can be downloaded from the Internet, is executable on Windows, OSX and Linux and the average user can install the program without the aid of others.

Definition 1.3. A program is suitable for students if it has a low learning curve, is user friendly and is pedagogical for students (see section 1.1.1).

Definition 1.3.1. A proof editor for first-order logic has a low learning curve if the average user can construct a proof that requires quantifiers and subproofs to reach the conclusion without the help of others.

Definition 1.3.2. A program is user friendly if no remarks more severe than minor (see section 5.2) were found using a heuristics evaluation with Jakob Nielsen's heuristics (see section 5.2.1).

1.1.1 Pedagogical Framing

Although pedagogy is an essential part of an educational aiding tool, in-depth investigations lie outside the scope of this thesis (see section 1.2). Instead, a few assumptions about pedagogy are made below. These assumptions were based on our experience as students.

Assumption 1. Providing students with instant feedback by verifying each inference rule application would aid the students' learning.

Assumption 2. It is pedagogically sound for an educational aiding tool to represent the subject matter in a fashion similar to the representation in the learning material.

Assumption 3. It is pedagogically sound for an educational aiding tool to include no more material than is required by the corresponding field.

Drawing from the assumptions above, we make the following definition.

Definition 1.3.3. An educational aiding tool that provides the choice of instant feedback, presents the material in the same fashion as the textbooks and limits its contents to the subject field is pedagogical for students.

1.2 Delimitations

This thesis is focused on educational aiding tools that supplement conventional learning. However, pedagogical theories were not investigated since those lie outside

the subject fields studied by the authors. Instead, the pedagogical evaluation was based on the assumptions (see section 1.1.1), set up by the authors.

This thesis will solely concern propositional logic and first-order logic and will not address usability for various kinds of disabilities because it is outside the authors' area of expertise.

1.3 Outcomes and Implications

The pre-study (see appendix A) suggests that there currently is a lack of proof editors that would function as a suitable aid to students learning to write natural deduction proofs in first-order logic. The primary shortcomings of existing editors lie in the user interface and documentation. There also exists more advanced applications for various kinds of logic proofs that were deemed unsuitable because of their complexity.

The heuristic evaluation (see section 6.2) showed only minor remarks for Conan's interface and the user tests (see section 6.1) indicated a low learning curve and a positive attitude towards the usefulness of the editor.

The question formulated in section 1.1 was: Is Conan a satisfactory educational aiding tool for students learning natural deduction in first-order logic? In section 7 it was concluded that this is indeed the case, based on the aforementioned evaluation and user tests and the assumptions laid out in section 1.1.1, and argued for in section 7.2.2 regarding pedagogy. In order to draw this conclusion, it must be assumed that Conan's verification is correct. Although some testing was performed, a rigorous testing process was not prioritised. For the editor to actually be used as part of a course in logic, such testing would have to be conducted to ensure the correctness of the verification of proofs.

The significance of the outcomes lies in the potential for a better logic education. If a student can get better feedback when writing proofs it could potentially help them to faster learn the material. It could perhaps help students more quickly recognise misunderstandings of the material and allow students to more easily test different approaches for writing a proof. It could also reduce the reliance on learning material, since solutions to exercises would not necessarily be needed. Because the editor makes larger proofs more manageable, it could also make it feasible for students to write larger and more complicated proofs.

Chapter 2

Theory

This section presents the mathematical theory that is required to understand this thesis to its full extent. Concepts that are central in mathematical logic are explained in order of complexity.

2.1 Propositional Logic

Propositional logic concerns itself with statements that are either true or false [1], where instead of using English sentences, we can replace them with formulas. 'Water is wet and the ocean is blue.' can be expressed as '$P \wedge Q$', where P replaces 'Water is wet' and Q replaces 'the ocean is blue'.

Here the letters denote *atoms*, which are formulas that cannot be deconstructed further [1], whereas $P \wedge Q$ can be deconstructed into the atoms P and Q.

We only concern ourselves about the truth value of the statement rather than the content of the statement. Instead of making deductions for a single statement like 'Water is wet', we can make general deductions for a set of statements that include the former.

We introduce our first *connective*, implication, denoted by the symbol $\rightarrow$. By using it in the following way, where P and Q are both propositions, we get a new proposition

$$P \rightarrow Q.$$

Connectives are used in conjunction with formulas, which creates new formulas, any time we use a connective correctly, we end up with a single new proposition. $\rightarrow$ is called a binary connective, since it takes two propositions and turns them into one proposition. An example of a unary connective is negation, denoted by $\neg$, which takes only one proposition and also turns it into one proposition.

$$\neg P.$$

The truth value of this new proposition depends on the truth value of P. We can enumerate these possible values using a truth table, where we let F denote falsehood, and T denote truth.

Table 2.1: Truth table for the $\neg$–connective.

P	$\neg P$
F	T
T	F

We will now list some of the logical connectives of propositional logic, with truth tables.

Table 2.2: Truth table for the $\wedge$– and $\vee$–connectives.

P	Q	$P \vee Q$	$P \wedge Q$
F	F	F	F
F	T	T	F
T	F	T	F
T	T	T	T

We note that $P \to Q$ may also be expressed as $\neg P \vee Q$, which is demonstrated in the following table:

Table 2.3: Truth table for the $\to$–connective.

P	Q	$P \to Q$	$\neg P \vee Q$
F	F	T	T
F	T	T	T
T	F	F	F
T	T	T	T

Any proposition we create by applying the connectives correctly, will take us from valid propositions to other valid propositions. If a proposition can be constructed this way, we say it is *well-formed* formula [1]. E.g. if ϕ, is a well-formed formula, then so is $\neg\phi$.

Returning to our building blocks we note that P, or any atom is by itself a well-formed formula, by extension any proposition we construct from these atoms using the connectives correctly will also be well-formed formulas.

Note that

$$\neg \wedge PQ,$$

is not well-formed since the formula can not be reached using the connectives correctly.

We note that $\neg P \wedge Q$ is equivalent to $(\neg P) \wedge Q$, and not equivalent to $\neg(P \wedge Q)$. Since the $\neg$–connective is applied before the $\wedge$–connective. As the exponentiation operation in mathematics is applied before multiplication, and as multiplication is applied before addition, there is also a order of operations here, which is determined by the connectives' precedence.

We have from highest to lowest precedence of the connectives we introduced:

$$\neg, \wedge, \vee, \rightarrow$$

2.2 Natural Deduction

The term natural deduction was introduced by the German mathematician and logician Gerhard Gentzen [2]. Natural deduction was constructed as a formalism that came close to actual reasoning, hence the name.

In natural deduction, conclusions are inferred from premises by applying *inference rules* [1]. Natural deduction is a way to prove that a logical reasoning is valid. Meaning that it is impossible for its premises to be true while having a conclusion that is false [3].

An example of an inference rule is the and-introduction:

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge i$$

Where the premises are at the top, with the conclusion at the bottom. The abbreviated name of the rule is given on the right side of the rule, where e stands for elimination and i stands for introduction. For some of the logical symbols the inference rules comes in two forms, namely introduction and elimination. The introduction rules introduces a logical symbol on the conclusion while the elimination rules eliminates a logical symbol from the premise. In the case of the and-Introduction rule it introduces a $\wedge$ when the conclusion $\phi \wedge \psi$ is drawn by using ϕ and ψ as premises. The inference rules follows the convention used by Huth and Ryan, where the logical formulas are represented as greek lower case letters.

Presented below are all basic inference rules in propositional logic and first-order logic (see figure 2.1). For more explanation on first-order logic see section 2.5.

	Introduction	Elimination
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}\,\wedge i$	$\dfrac{\phi \wedge \psi}{\phi}\,\wedge e_1 \qquad \dfrac{\phi \wedge \psi}{\psi}\,\wedge e_2$
$\vee$	$\dfrac{\phi}{\phi \vee \psi}\,\vee i_1 \qquad \dfrac{\psi}{\phi \vee \psi}\,\vee i_2$	$\dfrac{\phi \vee \psi \qquad \begin{array}{c}\boxed{\phi}\\ \vdots \\ \theta\end{array} \qquad \begin{array}{c}\boxed{\psi}\\ \vdots \\ \theta\end{array}}{\theta}\,\vee e$
$\rightarrow$	$\dfrac{\begin{array}{c}\boxed{\phi}\\ \vdots \\ \psi\end{array}}{\phi \rightarrow \psi}\,\rightarrow i$	$\dfrac{\phi \qquad \phi \rightarrow \psi}{\psi}\,\rightarrow e$
$\neg$	$\dfrac{\begin{array}{c}\boxed{\phi}\\ \vdots \\ \bot\end{array}}{\neg\phi}\,\neg i$	$\dfrac{\phi \qquad \neg\phi}{\bot}\,\neg e$
$\bot$		$\dfrac{\bot}{\phi}\,\bot e$
$\neg\neg$		$\dfrac{\neg\neg\phi}{\phi}\,\neg\neg e$
$\forall$	$\dfrac{\begin{array}{c}\boxed{x_0}\\ \vdots \\ \phi[x_0/x]\end{array}}{\forall x \phi}\,\forall x i$	$\dfrac{\forall x \phi}{\phi[t/x]}\,\forall x e$
$\exists$	$\dfrac{\phi[t/x]}{\exists x \phi}\,\exists x i$	$\dfrac{\exists x \phi \qquad \begin{array}{c}\boxed{x_0 \ \phi[x_0/x]}\\ \vdots \\ \theta\end{array}}{\theta}\,\exists e$
$=$	$\dfrac{}{t = t}\,= i$	$\dfrac{t_1 = t_2 \qquad \phi[t_1/x]}{\phi[t_2/x]}\,= e$

Figure 2.1: Basic rules of inference in natural deduction for propositional logic and first-order logic.

2.3 The Structure of Proofs

For proofs in natural deduction, we start with a set of zero or more premises and prove a formula which is provable from this set. In the scope of a proof we can justify each formula by using either inference rules or some other justification. We will use the Huth and Ryan style for representing proofs, which will be described further in section 2.4.

1.	P	premise
2.	Q	premise
3.	$P \wedge Q$	$\wedge$i, 1,2

Figure 2.2: The proof $P, Q \vdash P \wedge Q$, using the conjunction introduction rule.

Here, our formulas P, Q are justified by being premises. The formula $P \wedge Q$ on line 3 is justified by the conjunction introduction rule (see table 2.1).

We may denote the proof in figure 2.2 with what is known as a *sequent*:

$$P, Q \vdash P \wedge Q \tag{2.1}$$

The symbol $\vdash$ is called a turnstile, and can be read as 'proves'. This line can be thought of as the conclusion, $P \wedge Q$, being provable, given our premises: P, Q. The proof in figure 2.2 is therefore a proof of the sequent 2.1.

A proof is valid when every line up to and including the conclusion is justified. If there is no justification for a line, then any line that relies on that line for its own justification cannot be justified. Starting from the premises, any well-formed formula can be stated as is, if its existence is justified as a premise. From these premises we may use the inference rules to justify new formulas, and eventually end up at our conclusion.

Another justification that is not an inference rule or a premise is justification by *assumption*. This justification is valid as the first line of a *subproof*, which is a proof in a proof. However, the formula that is assumed is only justified inside of the subproof. E.g.

1.	$P \wedge Q \to (R \to S)$	premise
2.	P	premise
3.	Q	assumption
4.	$P \wedge Q$	$\wedge$i, 2,3
5.	$R \to S$	$\to$ e, 4, 1
6.	$Q \to (R \to S)$	$\to$ i, 3–5

Figure 2.3: The proof $P \wedge Q \to (R \to S), P \vdash Q \to (R \to S)$. Showcasing the use of a subproof.

Q is assumed inside the subproof, and may not be referenced outside of the scope of the box. As a subproof is a proof, it might also contain a subproof.

2.4 Visual Representations of Proofs

There are several different notation styles for representing proofs in natural deduction. Since it is outside the scope of this thesis to account for them all, only two broad remarks will be made. The first remark is that the most common styles present proofs in either tree form or tabular form. The second remark is that the most common styles present formulas either in sequent form (see section 2.3) or not. Figure 2.4 illustrates a tabular and non-sequent style that is sometimes referred to as Fitch-style notation. Figure 2.5 illustrates a tree and sequent styles that is sometimes referred to as Gentzen-style notation.

1	$P \wedge Q \to R$	
2	$\quad P$	
3	$\quad\quad Q$	
4	$\quad\quad P \wedge Q$	$\wedge$I, 2, 3
5	$\quad\quad R$	$\Rightarrow$E, 1, 4
6	$\quad Q \to R$	$\Rightarrow$I, 3–5
7	$P \to (Q \to R)$	$\Rightarrow$I, 2–6

Figure 2.4: Fitch-style for representing the proof $P \wedge Q \to R \vdash P \to (Q \to R)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P \wedge Q \to R, P, Q \vdash P \wedge Q \to R}\,\text{hyp} \quad \dfrac{\dfrac{}{P \wedge Q \to R, P, Q \vdash P}\,\text{hyp} \quad \dfrac{}{P \wedge Q \to R, P, Q \vdash Q}\,\text{hyp}}{P \wedge Q \to R, P, Q \vdash P \wedge Q}\,\wedge\text{I}}{P \wedge Q \to R, P, Q \vdash R}\,\to\text{E}}{P \wedge Q \to R, P \vdash Q \to R}\,\to\text{I}}{P \wedge Q \to R \vdash P \to (Q \to R)}\,\to\text{I}$$

Figure 2.5: Gentzen-style for representing the proof $P \wedge Q \to R \vdash P \to (Q \to R)$

As can be seen in figures 2.4 and 2.5, there are a number of important differences between Fitch-style and Gentzen-style representation. Firstly, Fitch-style proofs expand only in length, whereas Gentzen-style proofs expand in both length and width. As a result, Gentzen-style proofs are in terms of physical space less suitable for a number of frequently used media, including paper. Secondly, Fitch-style proofs make use indexed row numbers whereas Gentzen-style proofs do not. As a result, Fitch-style proofs provides the option of cross-referencing different parts of the proof.

On the other hand, it can be argued that cross-referencing is a diversion from the inherent structure of the logic language.

In this thesis, the primary notation used follow the same style [1] as Huth and Ryan (see figure 2.6). This style is a descendant of Fitch-style notation and presents subproofs (see section 2.3) enclosed in boxes as opposed to being indented.

1.	$P \wedge Q \to R$	premise
2.	P	assumption
3.	Q	assumption
4.	$P \wedge Q$	$\wedge$i, 2,s
5.	R	$\to$ e, 1, 4
6.	$Q \to R$	$\to$ i, 3–5
7.	$P \to Q \to R$	$\to$ i, 2–6

Figure 2.6: Huth and Ryan-style for representing the proof $P \wedge Q \to R \vdash P \to (Q \to R)$

2.5 First-Order Logic

First-order logic is an extension of propositional logic (see section 2.1) [4]. In first-order logic, *logical functions* take zero or more elements from a set of objects under consideration commonly referred to as the universe, U, and returns an element from the universe. This can formally be denoted as

$$f : U^n \to U, \quad n \in \mathbb{N}.$$

A *predicate* is a function that takes zero or more elements from the universe and returns either true or false, this can formally be expressed as,

$$P : U^n \to \{\text{True}, \text{False}\}, \quad n \in \mathbb{N}.$$

Let us clarify with an example.

$$\begin{aligned} U &:= \{\text{cat}, \text{dog}, \text{lion}, \text{wolf}\}, \\ P(x) &: x \text{ is feline}, \\ f(x) &: x\text{'s closest biological relative}. \end{aligned} \tag{2.2}$$

Predicates can be turned into statements, by being *specified* or *quantified* [4]. Specifying a predicate means that we fixate values for the variables, which turns the predicate to a propositional statement [4]. For example, $P(\text{cat}) : True$.

Quantifying a predicate means that we reason about for which x the predicate is true. The two most common [4] ways to quantify a predicate is to reason about

whether or not the predicate is true for all x in the universe, denoted as $\forall x P(x)$, and whether or not the predicate is true for at least one x in the universe, denoted as $\exists x P(x)$. Continuing on the example 2.2, $\exists x P(x)$ would be true since the universe contains at least one feline.

There are many similarities between proofs in natural deduction calculus for first-order logic and proofs in natural deduction for propositional logic. The difference is that first-order logic requires additional rules of inference in order to deal with the equality symbol and quantifiers [1]. All established inference rules in propositional logic are still valid in natural deduction calculus, but we add additional rules for quantifiers and equality. The additional rules are divided into introduction and elimination rules and can be found in the bottom part of figure 2.1.

An example of an inference rule for a quantifier is the for all-elimination rule:

$$\frac{\forall x \phi}{\phi[t/x]} \forall x e$$

Where the premises are at the top, with the conclusion at the bottom. The abbreviated name of the rule is given on the right side of the rule, where e stands for elimination. In order to use this rule we have to introduce a way of changing a variable to a term, this is defined in the course literature *Logic in Computer Science-Modelling and Reasoning about Systems* [1] as 'Given a variable x, a term t and a formula ϕ we define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t'.

Another important concept in first-order logic is to introduce a new variable, this concept is called *fresh*. A fresh variable is a variable that we do not have any previous knowledge about. This means that any conclusions we draw about x must hold for all possible objects. For example, if x is fresh and we can prove P(x), then it must be true that $\forall x P(x)$. An example of a proof in first-order logic with quantifiers can be seen in figure 2.7.

1.	$\forall x(P(x) \to Q(x))$	premise
2.	$\exists x P(x)$	premise
3.	x_0	fresh
4.	$P(x_0)$	assumption
5.	$P(x_0) \to Q(x_0)$	$\forall$e, 1
6.	$Q(x_0)$	$\to$ e, 5, 4
7.	$\exists x Q(x)$	$\exists$i, 6
8.	$\exists x Q(x)$	$\exists$e, 2, 3–7

Figure 2.7: The proof $\forall x(P(x) \to Q(x)), \exists x P(x) \vdash \exists x Q(x)$ in first-order logic. Note in particular, the presence of quantifiers in the proof.

Chapter 3

The History and Current State of Proofs Assistants

This section gives an account for previous research and development within the area of proof assistants. The intention is to provide a context for this thesis in order to help establish its relevance. It is outside the scope of this thesis to provide an exhaustive description of the history of the area. Instead, a few historical milestones will be presented, as well as shortcomings in the area found by independent research groups.

For clarifying purposes, a few definitions are made in this chapter in order to differentiate proof assisting tools by their primary purpose.

Definition 3.1 A *proof assistant* is any tool that can be used for assistance in deductive reasoning. This includes abstract, physical and digital machines.

Definition 3.2 A *theorem prover* is a specialised proof assistant with the primary purpose of performing research. These are most frequently used by expert logicians and are powerful tools that often handles many different kinds of logic and the intended proofs are large.

Definition 3.3 A *proof editor* is a specialised proof assistant with the primary purpose of aiding the learning of logic. These are most frequently used by students and novice logicians, are tools that handles a select few kinds of logic and the intended proofs are smaller than those in theorem provers.

3.1 Early Proof Assistants

The early development of proof assistants dates back to the mid 1950s [5]. By that time, proof assistants were divided in two categories. The first category was proof assistants that were intended to emulate the process of proof construction [5]. Allen

Newell, Herbert A. Simon and Cliff Shaw were pioneers in this category [5] with their program *Logic Theorist*, finished by 1956 [6]. The second category was proof assistants that were intended to use a systematic approach, whereby the problems were reduced to standard forms and then solved by known algorithms [5]. Dag Prawitz was a pioneer in this category [7] and developed a mechanical procedure for first-order logic proofs in 1957 [8].

3.2 Modern Proof Assistants

Since the mid 1950s, available proof assistants have surged (see appendix E), in part due to the popularity of formal methods in software engineering [9]. Interactive theorem provers in particular have seen notable successes [10]. Of these, *Isabelle* (a descendant of *Edinburgh LCF*) and *Coq* (a descendant of *Automath*) are two of the most popular in modern time [10]. Proof editors, on the other hand, have seen moderate success. Of these, *Jape* is the most popular [11]. Evidence does suggest that students develop sophisticated proof strategies more effectively when using *ItL Jape* (a certain implementation of *Jape*) compared to pencil-and-paper [12]. However, it has been argued that *Jape* needs improvement in order to fit beginners [13].

Prior to this thesis, a pre-study was conducted (see appendix A) in order to establish whether or not there was a deficit in proof editors aimed at students learning natural deduction in first-order logic. The data sample consisted of proof editors available on the Internet and those deemed most relevant to the purpose were investigated further. The result of the pre-study suggests that existing editors are lacking in various relevant aspects. Therefore, it was concluded that although satisfactory tools may have been overlooked, there exists potential value in developing additional proof editors with a particular focus on presenting an easy-to-use interface for students learning natural deduction in first-order logic.

Chapter 4

Description of the Developed Editor

Conan was developed to function as a supplementary aid alongside a course in logic. More specifically, it is supposed to help students in practising writing natural deduction proofs. It was therefore aimed at beginners and not researchers or experts and did not aim to be a powerful theorem prover (see section 3).

Since it was supposed to be a supplement rather than a standalone tool, Conan did not aim to teach all the relevant material for writing proofs. Instead, students were expected to learn the relevant logic theory from the course material. Conan should instead function as an environment where this theory can be applied, where the process and strategies of proof writing can be practised.

In order to fit well within the limited time of a typical university course, an interface that was quick to learn and easy to understand was very highly prioritised.

The program is easily accessible and is available online[1] in a jar file format.

[1] `http://web.student.chalmers.se/~jespolss/Conan.html`

4.1 Notable Features

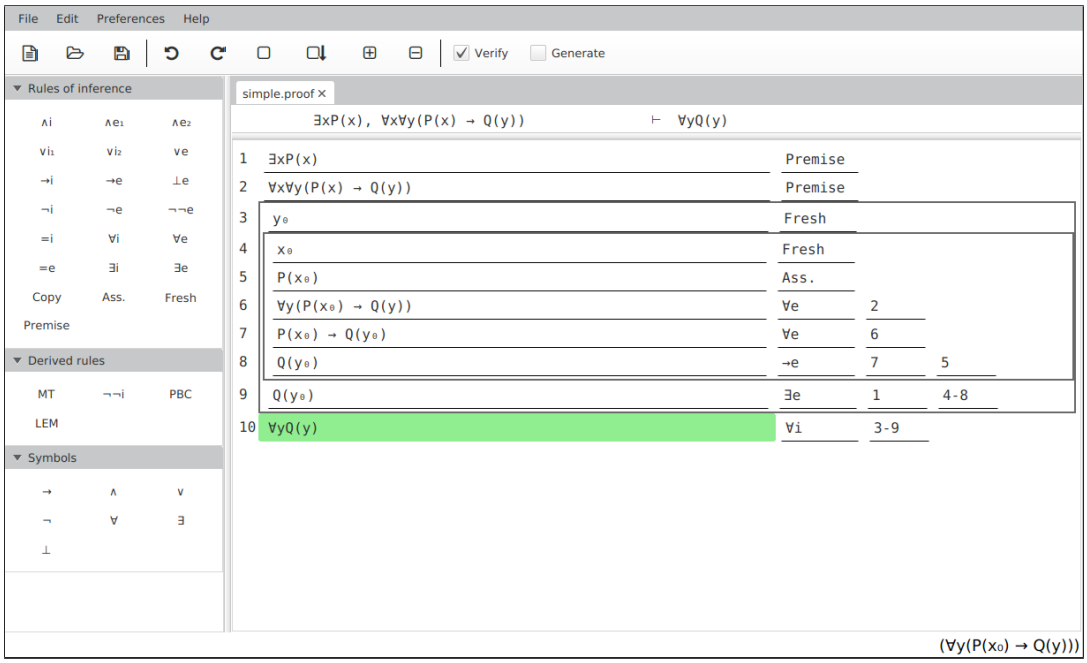


Figure 4.1: The graphical user interface of Conan.

Conan offers a design that is akin to text editors, which allows the user to familiarise themselves with the program faster from their common use of the former. The user is allowed to jump to any row of the proof and is able to insert new rows or delete and edit old rows to correct mistakes or even introduce new ones.

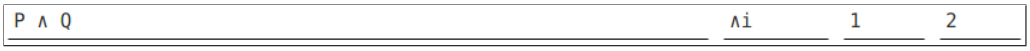


Figure 4.2: A row has a formula field, a rule field and reference fields.

A row consists of a formula field, a rule field and up to three reference fields (see figure 4.2). The formula field is where the user inputs the desired formula and the rule field should contain the rule that justifies it. References fields are only shown when the rule requires them and are used in conjunction with that specific rule.

Figure 4.3: The formula is not well-formed and therefore the field turns pink to inform the user.

The user is free to input any type of formula, including formulas that are not well-formed. However, while the formula is not well-formed, the formula field turns pink, instantly alerting the user of the mistake they introduced (see figure 4.3).

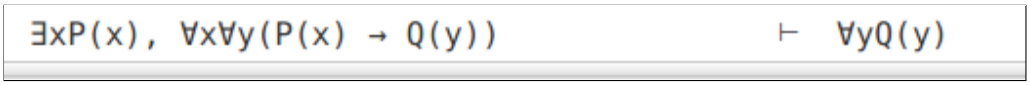


Figure 4.4: The sequent in Conan

At the top of the proof we have the sequent (see figure 4.4), which will be updated automatically, adding all the premises of the proof. The consequent of the sequent is used to indicate when the goal has been reached in a valid way. This is indicated with a green colour (see figure 4.1).

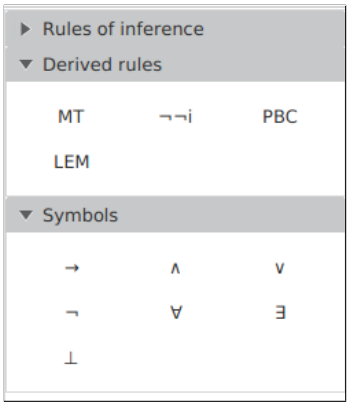


Figure 4.5: Collapsible panes in Conan.

There are collapsible panes of buttons showcasing the rules of natural deduction, a few derived rules and the symbols which are used in the program. The user may press these buttons to input the rule or symbol to the active field or row. Collapsing the pane allows the user to hide the buttons and work on their recall at will.

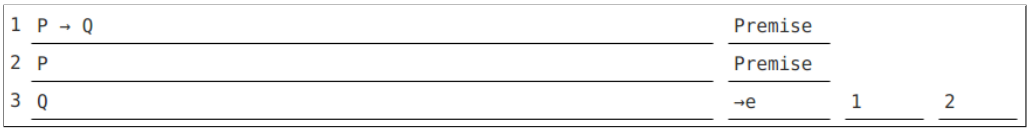


Figure 4.6: Three verified rows in Conan.

In the short proof in figure 4.6, we have that the first two rows are verified since both of them are well-formed formulas that are justified by being premises. The third row is verified since implication elimination takes one implication as a reference and the left hand side of the implication as the other reference. A final check makes sure that the right hand side of the implication matches the row in question.

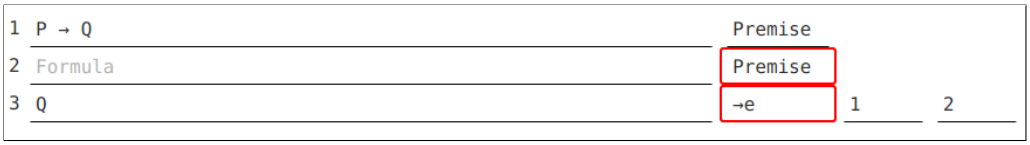


Figure 4.7: Introducing an error to figure 4.6, making two rows unverified.

Erasing the formula in row two makes the row unverified and affects the third row. Red frames appear, surrounding the rules, alerting the user of the error.

For this proof, we may also generate the formula of the third row by first inputting two verified rows and then inputting both the rule and the correct references in the third row, generating the formula automatically resulting in the same proof as figure 4.6.

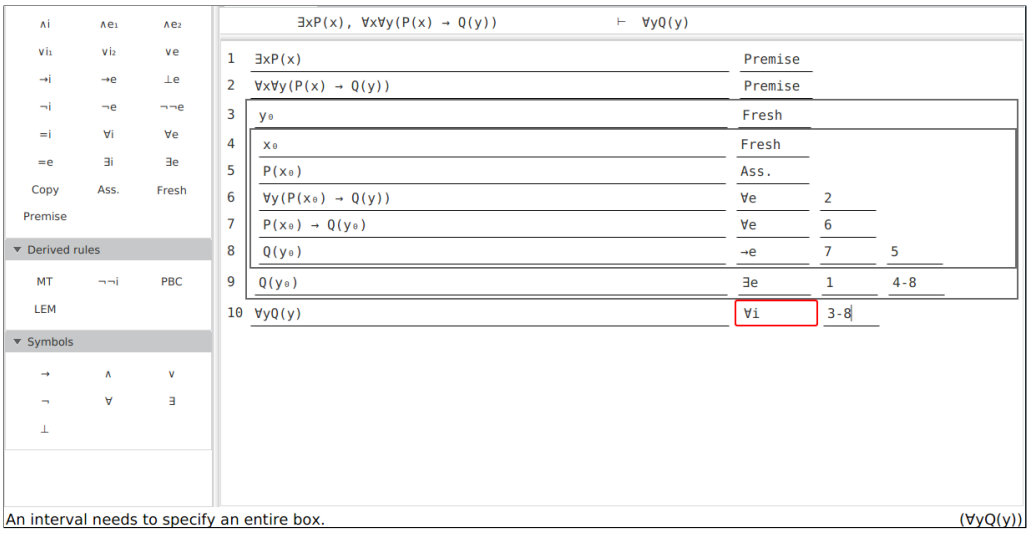


Figure 4.8: Row specifying an incorrect interval as a reference, which the user is alerted to by the left status field. The right status field shows how the formula in the current row is interpreted by the program.

The status bars are located at the bottom of the program. The left status field is used to indicate errors and the right status field shows how the formula is interpreted in the program (see figure 4.8). The errors in the status bar are minimal and only indicate basic slips, instead of indicating errors in proof strategy or faulty reasoning.

4.2 Graphical User Interface

The graphical user interface was determined to be of critical importance since that was the major flaw of existing editors (see appendix A). The design choices were based on our own experiences as students and general program users. One option that was considered early on was to use the generic front-end for proof assistants called Proof General[2]. It is an Emacs-based interface and it was decided that it would better to provide our own interface with a heavier focus on ease of use and a low learning curve.

The importance of using proper unicode notation for the logical expressions was

[2] `https://proofgeneral.github.io`

determined both because we wanted the interface to be consistent with literature and to avoid the users having to learn a particular syntax for the editor.

In order to make the unicode symbols easy to input, clearly visible buttons were added in the left-side menu. Though having buttons makes it easy to find how to insert these symbols, it does not lead to a smooth workflow. To remedy this, faster ways to insert the desired characters were added, like typing '&' or 'an' for '∧' as well as '->' or 'im' for '→' and so on.

To further improve the workflow, alternative input options for other actions were also added. For example, deleting a row can be done either by clicking a button in the toolbar, accessing the option under the Edit menu, right-clicking a particular row or by simply pressing CTRL+D when having a row focused.

Throughout the development process the overall goal of the design was to make sure that available options were easy and intuitive to find for new users and to allow more advanced users to maintain a nice workflow.

4.3 Saving and Loading of Proofs

For a student practising writing proofs it would be important to be able to save proofs and later load the proof. A student might get stuck on a particular proof and wish to continue it later or might want to save finished proofs to look at, perhaps when preparing for an exam. It could also facilitate assignment hand-ins where a student can upload a proof file rather than turn in a piece of paper or have to scan it.

4.4 Exportation to LaTeX

This feature was implemented since it allows users to display their proofs in reports and hand-ins. It was one of different ways suggested for exportation, among these were exportation to a picture format and portable document format. However, exportation LaTeX is the most flexible of these options, as it allows the user to convert the result more easily to any of the previously mentioned ones than vice verse. Instead of developing our own package, we used the existing package, logicproof [3], to have a proof format similar to the program.

[3] `https://www.ctan.org/pkg/logicproof`

4.5 Reference Material for the Implemented Logic in Conan

According to the assumptions about pedagogy (see section 1.1.1), it is sound to follow the style of the course material. However, the notation style (2.4) used for representing natural deduction varies depending on the textbook. In order to decide what source material to follow when implementing the proof editor, an investigation was performed in order to find the most commonly used textbook in university courses for learning natural deduction.

We used Google to search the Internet for the phrases 'logik naturlig deduktion', 'Logic natural deduction course' and 'Formal Methods Natural Deduction Course'. Results that returned websites for courses where natural deduction were provided by a department of computer science at a university were investigated. In total, 25 universities were investigated. The data is available in appendix B.

As can be seen in appendix B, 18 of 25 universities used the book *Logic in Computer Science: Modelling and Reasoning about Systems* in their teaching either as the main material or recommended it as a supplementary material, making it one of the most commonly used textbook for teaching natural deduction. Therefore, it was decided that Conan should mimic the style of this book. This implies that Conan should follow the conventions used in the textbook, including its visual representation of proofs (see section 2.4), naming conventions for terms and predicates, order of precedence and associativity rules.

4.6 Limitations of Conan

Some limitations result directly from the time limit set by the project, such as the inability to switch visual display between tree form and the linear box form that is now in the program. Creating and applying custom rules was planned as an extra feature, which would allow the user to save a proof as a rule and reuse it for other proofs.

Extensive testing is another limitation of the editor. There may exist cases where the verification is not correct which could have been found by testing. More testing would make such errors less likely.

Finally, the editor does not allow total freedom in editing; a box is not allowed to start in a box, and any action that results in this is disallowed. Therefore the user may not delete the first row in a box, if the second row is the start of another box. This is an intentional limitation, however there is no explicit message that informs the user of this.

Chapter 5

Methods

This chapter presents the methods used in this thesis in order to obtain empirical data. The purpose of this section is to provide an objective account for the set-up, specific enough to enable reproducibility of the data or judge the quality of the data. This chapter provides an explanation of the methods used when assessing the learning curve and user-friendliness of Conan. This data was gathered through user tests and a heuristic evaluation respectively.

5.1 Assessment of Learning Curve Through User Tests

In order to assess if Conan had a low learning curve, user tests were performed. As defined in section 1.1, a proof editor for first-order logic has a low learning curve if the average user can construct a proof that requires quantifiers and subproofs to reach the conclusion without the help of others.

There were two test groups participating in this study: one with beginners and one with experienced users, each group consisting of five students. The beginners had not taken any course in logic and consisted of a mixed group from the two bachelor programmes in Computer and Software Engineering and from the Master programme Algorithms, Languages and Logic at Chalmers University of Technology[1]. The experienced group had completed the course *Logic in Computer Science* [2] at Chalmers University of Technology and were all attending the master programme Algorithms, Languages and Logic at Chalmers University of Technology.

The difference between the user tests for the two test groups was that the experienced group answered five additional questions (see the end of this section) relevant to the

[1] `https://www.chalmers.se/en/Pages/default.aspx`
[2] `http://www.cse.chalmers.se/edu/year/2016/course/DAT060/index.html`

logic course they had taken.

For each test, one or two developers of the system were assigned the task as test supervisor and hence guiding the testers throughout the user tests while also taking notes. Each tester performed the user test independently and gave oral feedback throughout the user test. It should be noted that the user tests were not held in a controlled environment and that the testers did not receive any help while constructing the proofs.

All testers were given a paper with three complete proofs to construct in the program. The testers had the entire solution of the proofs in front of them while performing the test to ensure that only the learning curve of the program was tested, and not the testers ability to solve proofs. The testers started the test by constructing a short proof in the program with no quantifiers and just one box, as can be seen in figure 5.1. The testers then continued by constructing a larger proof in the program with quantifiers and boxes within boxes, as can be seen in figure 5.2.

1.	B	premise
2.	A	assumption
3.	B	copy, 1
4.	$A \to B$	$\to$ i, 2–3

Figure 5.1: The first proof, $B \vdash A \to B$, that the testers constructed in the user test.

1.	$\neg\forall x P(x)$	premise
2.	$\neg\exists x \neg P(x)$	assumption
3.	y	Fresh
4.	$\neg P(y)$	assumption
5.	$\exists x \neg P(x)$	$\exists$i, 4
6.	$\bot$	$\neg$e, 5, 2
7.	$P(y)$	PBC, 4–6
8.	$\forall x P(x)$	$\forall$i, 3–7
9.	$\bot$	$\neg$e, 8, 1
10.	$\exists x \neg P(x)$	PBC, 2–9

Figure 5.2: The second proof, $\neg\forall x P(x) \vdash \exists \neg x P(x)$, that the testers constructed in the user test.

After constructing the second proof, the testers were asked to edit a third proof that had several errors in it, as can be seen in figure 5.3. The testers task was to edit the proof until it looked like the correct one that was presented to them in the paper.

The testers were finished with the third proof when it looked like the proof in figure 5.4.

The test supervisors wrote down the testers' comments about the program while also taking notes on how long it took for each tester to finish each proof.

1.	$\exists x P(x)$	premise
2.	$\forall x H(x)$	premise
3.	$\forall x \forall y (P(x) \to Q(y) \wedge H(x))$	premise
4.	y_0	Fresh
5.	x_0	Fresh
6.	b	Fresh
7.	A	assumption
8.	$b \to A$	$\to$ i, 6–7
9.	$P(x_0)$	assumption
10.	$\exists x P(x) \wedge \forall x \forall y (P(x) \to Q(y) \wedge H(x))$	$\wedge$i, 1,s
11.	$\forall y (P(x_0) \to Q(y) \wedge H(x_0))$	$\forall$e, 3
12.	$P(x_0) \to Q(y_0) \wedge H(x_0)$	$\forall$e, 11
13.	$Q(y_0)$	$\to$ e, 12, 9
14.	$Q(y_0)$	$\exists$e, 1, 5–13
15.	$\forall y Q(y)$	$\forall$i, 4–14

Figure 5.3: The third proof, $\exists x P(x), \forall x H(x), \forall x \forall y (P(x) \to Q(y)) \wedge H(x)) \vdash \forall y Q(y)$ that had several errors in it that the testers were asked to edit.

1.	$\exists x P(x)$	premise
2.	$\forall x \forall y (P(x) \to Q(y))$	premise
3.	y_0	Fresh
4.	x_0	Fresh
5.	$P(x_0)$	assumption
6.	$\forall y (P(x_0) \to Q(y))$	$\forall$e, 2
7.	$P(x_0) \to Q(y_0)$	$\forall$e, 6
8.	$Q(y_0)$	$\to$ e, 7, 5
9.	$Q(y_0)$	$\exists$e, 1, 4–8
10.	$\forall y Q(y)$	$\forall$i, 3–9

Figure 5.4: The third proof, $\exists x P(x), \forall x \forall y (P(x) \to Q(y)) \vdash \forall y Q(y)$, that the testers constructed in the user test.

After finishing the three proofs, the testers were asked questions about the program.

All the testers were asked the following three questions:

1. How user-friendly do you think Conan is in a scale from 1-5?
2. How intuitive do you think Conan is in a scale from 1-5?
3. Do you have anything more to add?

The experienced group were also asked the following five questions:

4. Do you think Conan would have been a useful tool during the course *Logic in Computer Science*?
5. Have you used any other proof editors before?
6. If yes: Do you think that the other editor was more useful than Conan?
7. Do you think it would be useful if the teacher could hand out assignments by using Conan?
8. Do you think that Conan would have been useful during assignments in the course *Logic in Computer Science*?

Conan was updated after the user tests were performed, small changes were made to the interface and new functionality was added[3].

5.2 Assessment of User-Friendliness Through a Heuristic Evaluation

In order to assess the user-friendliness of Conan, a heuristic evaluation was performed using Jakob Nielsen's heuristics. According to the definition in 1.1, Conan is defined as user-friendly if the heuristic evaluation finds no remarks more severe than *minor* (see section 5.2.2). First, Jakob Nielsen's heuristics are described (see section 5.2.1). Then, an account is given for how the evaluation was performed (see section 5.2.2).

5.2.1 Description of Jakob Nielsen's heuristics

This section provides an explanation of Jakob Nielsen's heuristics [14]. The aim of this section is to elucidate what was evaluated during the heuristic evaluation of Conan (see section 6.2). Nielsen's heuristics consists of ten general principles for interaction design. The following ten principles forms an excerpt from Jakob Nielsens heuristics [15]:

[3]For the sake of reproducibility, the version used in the user tests can be found at `https://github.com/nonilole/Conan/releases/tag/a994f7b`

1. Visibility of system status

   'The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.'

2. Match between system and the real world

   'The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.'

3. User control and freedom

   'Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.'

4. Consistency and standards

   'Users should not have to wonder whether different words, situations, or actions mean the same thing.'

5. Error prevention

   'Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.'

6. Recognition rather than recall

   'Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.'

7. Flexibility and efficiency of use

   'Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.'

8. Aesthetic and minimalist design

   'Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.'

9. Help users recognise, diagnose, and recover from errors

'Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.'

10. Help and documentation

    'Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.'

5.2.2 The Process of Performing the Heuristic Evaluation

The heuristic evaluation was conducted by the developers of the program. In order to determine to what degree Conan followed Jakob Nielsen's usability heuristics (see section 5.2.1), the developers of Conan defined a scale. The scale consisted of four levels: no negative remarks, minor remarks, significant remarks and major remarks. Minor remarks means that the program violates the heuristic, but not in a way that it significantly impacts usability. Significant remarks means that the program violates the heuristic in such a way that it affects the user experience and major remarks means that the program violates the heuristic in a way that it makes it hard to use the program. Each heuristic principle received either no negative remarks, minor remarks, significant remarks or major remarks.

The developers were more thorough in this heuristic evaluation compared with the ones performed in the pre-study of existing proof editors (see appendix A) since this heuristic evaluation had a great impact on answering our problem statement. The heuristic evaluation was performed after the completion of the user tests and is based on Conan's current developing state[4] (see chapter 4).

[4]For the sake of reproducibility, this version can be found at `https://github.com/nonilole/Conan/releases/tag/1.0`

Chapter 6

Results

This chapter gives an account for the empirical results found by the user tests and the heuristic evaluation.

6.1 Results of the User Tests

This section presents the result of the user tests. In section 5.1, information about how the user tests were performed and which questions were asked can be found. Figures 6.1, 6.2 and 6.3 are scatter plots showing how much time it took to finish the first, second and the third proof, both for the developers and the students who participated in the user tests. The plots also show how much time it took for the developers of the editor to finish each proof. For the exact times, see Appendix C. All testers managed to construct a proof without the help of others that required quantifiers and subproofs to reach the conclusion.

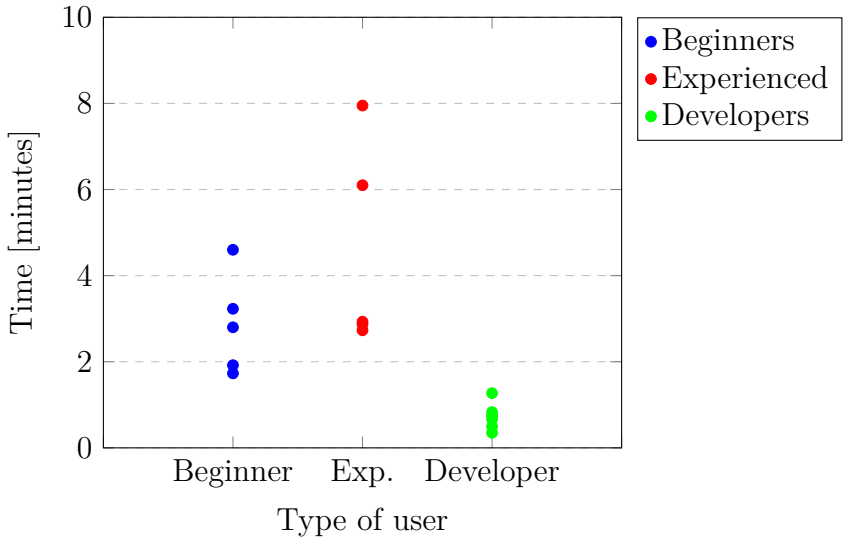


Figure 6.1: Graph showing the time it took to complete the first proof in the user test.

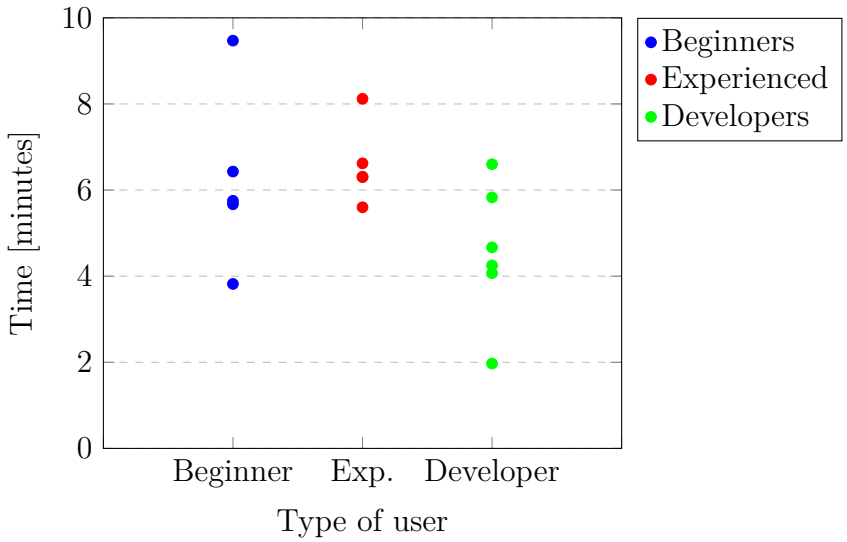


Figure 6.2: Graph showing the time it took to complete the second proof in the user test.

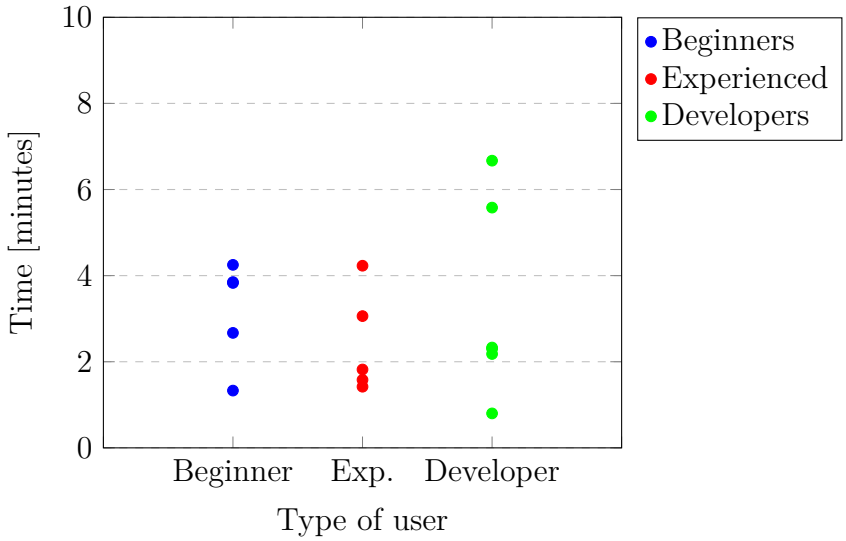


Figure 6.3: Graph showing the time it took to complete the third proof in the user test.

The results of question 1 'How user-friendly do you think the proof editor is in a scale from 1-5?' is displayed in Figure 6.4. The results of question 2 'How intuitive do you think the proof editor is in a scale from 1-5?' is displayed in Figure 6.5.

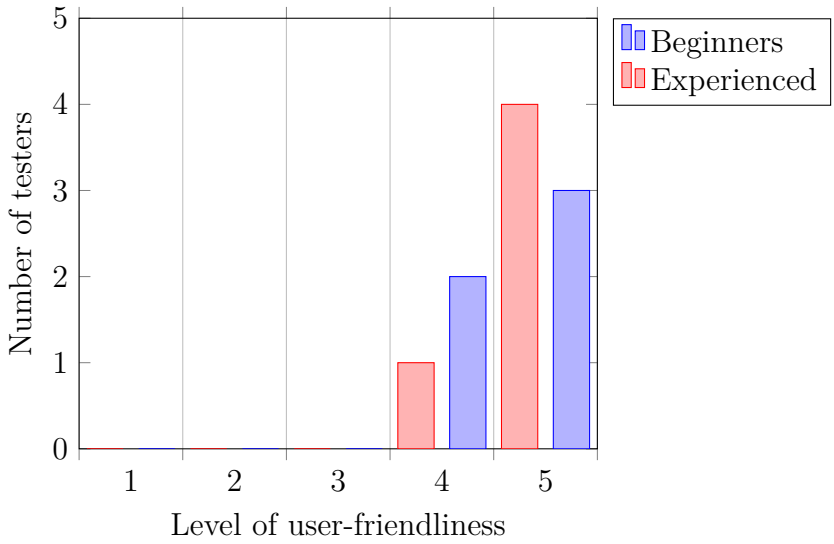


Figure 6.4: The result of the first question 'How user-friendly do you think the proof editor is in a scale from 1-5?' that was asked during the user test. The scale is from level 1 meaning not user-friendly to Level 5 meaning very user-friendly.

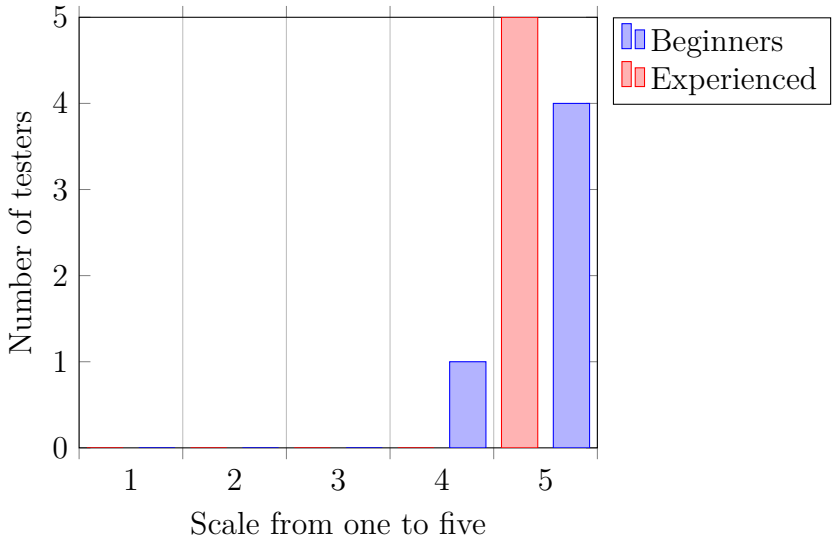


Figure 6.5: The result of the second question 'How intuitive do you think the proof editor is in a scale from 1-5?' that was asked during the user test. The scale is from Level 1 meaning not intuitive to Level 5 meaning very intuitive.

The experienced group completed five additional questions. As can be seen in table 6.1, all experienced users answered yes to question 4, 7 and 8. Out of those who answered yes to question 5, all answered no to question 6.

Table 6.1: The five experienced users' answers to questions 4-8 in the user test survey. Note in particular the coherence in the answers to question 4, 7 and 8.

	Yes	No
Question 4: Do you think the proof editor would have been a useful tool during the course *Logic in Computer Science*?	5	0
Question 5: Have you used any other proof editors before?	3	2
Question 6: If yes: Do you think that the other editor was more useful than Conan?	0	3
Question 7: Do you think it would be useful if the teacher could hand out assignments by using the proof editor?	5	0
Question 8: Do you think that the proof editor would have been useful during laborations in *Logic in Computer Science*?	5	0

Common feedback during the user tests and the written answers to the third question 'Do you have anything more to add?' was that it was too hard to figure out how to add a row outside a box in a proof. The icon for the button that had that functionality was also difficult to find because of the icon's appearance. Ninety percent of the testers thought that the editor was easy to use and fast to learn. The shortcuts for writing and editing a proof was appreciated by all testers who were

able to detect that the editor had shortcuts. All testers liked that they could export proofs to LaTeX and thought that the editor had a good design.

6.2 Results of the Heuristic Evaluation

In order to determine to what degree the proof editor followed Jakob Nielsen's usability heuristics (see section 5.2.1), an heuristic evaluation was performed.

Conan received no negative remarks on five out of ten general heuristic principles: *Error prevention, Recognition rather than recall, Flexibility and efficiency of use, Aesthetic and minimalist design* and *Help and documentation.*

Error prevention: There were no remarks on constructing the proofs since the user should be allowed to make mistakes and be informed about them when writing a proof so that they can learn from their mistakes.

Recognition rather than recall: Conan received positive feedback saying that all actions were easily accessible and visible, that all frequently used actions were available in the menu and finally that instructions were easy to reach.

Flexibility and efficiency of use: The positive aspects were that the user could use shortcuts that appeared to be very flexible due to multiple input options. The editor also had text replacement to simplify writing proofs. Conan did not allow the user to tailor frequently used actions, but since Conan had multiple ways of performing an action, the evaluators did not believe that this remark was important enough to generate a minor remark.

Aesthetic and minimalist design: This principle received positive feedback saying that Conan had an aesthetic and minimalist design.

Help and documentation: The positive aspects regarding the principle were that the documentation in Conan was concise. Conan could easily be operated without reading the documentation.

The other five principles: *Visibility of system status, Match between system and the real world, User control and freedom, Consistency and standards* and *Help and documentation,* had minor remarks.

Visibility of system status: The principle received minor remarks since Conan did not inform the user whether or not a proof was saved. The positive aspects regarding *Visibility of system status* was that the user got a reminder to save proofs before closing the program and that the user received feedback while constructing a proof.

Match between system and the real world: Since Conan used the word *scope* in error messages, the principle received minor remarks due to the fact that the word

may not be established amongst students learning first order logic. The principle received positive feedback saying that the construction and visualisation of proofs in the proof editor closely resembled of those in the course literature, *Logic in computer science: Modelling and Reasoning about Systems*. Conan only uses the logic that its users are expected to learn.

User control and freedom: Conan received minor remarks on this heuristic because the user in some cases could not get back deleted text by clicking undo. The positive feedback was that Conan did not have a state where the user needed an emergency exit.

Consistency and standards: The principle received minor remarks since the welcome screen does not appear each time the user opens a new proof, which can be considered to be inconsistent since this requires the user to fill out the premise and conclusion in different ways. Conan's menu did not have mnemonics[1] but it had shortcuts for most frequently used actions.

Help users recognize, diagnose, and recover from errors: The principle received minor remarks on the fact that the user did not receive an exact solution suggestion in the error message.

[1]For the purpose of clarification, in this instance *mnemonics* refers to `https://en.wikipedia.org/wiki/Mnemonics_(keyboard)`

Chapter 7

Discussion and Conclusion

This chapter discusses the results presented in this thesis and argue for their validity. Furthermore, this chapter attempts to review Conan from an environmental, economical and social perspective and also state the contribution of this project. Finally, future development of Conan and potential further research is discussed before a conclusion is drawn about whether or not the problem statement was fulfilled and implications of the result.

7.1 Fulfilment of the Definitions for a Satisfactory Educational Aiding Tool

This section assesses Conan according to the definitions (see section 1.1) for a satisfactory educational aiding tool for students learning natural deduction in first-order logic.

All in all, Conan fulfils the definition of a satisfactory educational aiding tool for students learning natural deduction in first-order logic.

It provides the necessary functionality of a proof editor, since Conan can edit proofs, save and load proofs as well as verify the correctness of all deduction steps.

Conan fulfils the criteria for accessibility. Because the proof editor is available in a jar file format (see section 4), it is executable on Windows, Mac and Linux. The jar file can be downloaded from the Internet, enable potential users to install Conan without the aid of others.

Since the tests show (see section 6.1) that both experienced and inexperienced users single-handedly could complete all example proofs in less than ten minutes, we conclude that the average user can construct a proof that requires quantifiers and sub proofs to reach the conclusion without the help of others. Therefore we can

draw the conclusion that the program has a low learning curve using our definition at 1.1.

Since no remarks more severe than minor were found using a heuristic evaluation with Jakob Nielsen's heuristics, the editor is deemed user friendly. Additional supporting evidence was found in the results of the user tests (see section 6.1).

Conan fulfils the definition of being pedagogical for students. It provides instant feedback through verification of inference rule application (see section 4.1). In Conan, the learning material is presented in the same fashion as in the most frequently used textbook (see section 4.5). Finally, Conan limits its contents to first-order logic.

7.2 Plausibility of the Thesis Premises

In this thesis, assumptions were made about the existing proof editors not being satisfactory for students learning natural deduction. The initial phase of the pre-study was to search for as many potential editors as we could find. From these, the 3 editors that were closest to what we wanted to achieve were selected for a more in-depth evaluation. The process of selecting the proof editors to appraise is by no means rigorous. The result of the pre-study reveals that the selection of the most relevant proof editors that was found had flaws, therefore the lack of a satisfactory educational aiding tool is clearly an issue. However, there is a possibility that there exists a proof editor that has been overlooked which meets our criteria (see section 1.1) of a satisfactory educational aiding tool.

In addition, it is possible that the pre-study was overly harsh because of the evaluators' vested interest in justifying the development of Conan. The time spent on the heuristic evaluations was limited and thus it is possible that the evaluators jumped to conclusions because they had not had sufficient familiarised themselves with the proof editors.

7.2.1 Satisfaction Criteria Definitions

The definitions made in the Problem Statement (see section 1.1) were based on our experience as students and the pre-study (see appendix A) of existing proof editors and their shortcomings. Our experience was in part composed of having used a large amount of editors, educational aiding tools and programs in general. As a result, the definitions were not based on any scientific standards and may be inaccurate. In order to ensure the relevance of our criteria, logic teachers should have been consulted early in the development process of Conan.

Factors that affect the validity was that we had low diversity in our fields of study.

We came from highly interconnected schools and were all Swedish citizens. We were probably not a representative sample of all students. On the other hand as students, of who some have taken the Chalmers course *Logic in Computer Science*, were also included in the target audience for Conan. Thus what we wanted the editor to provide, in terms of functionality and usability, had a high probability of being close to what other students would want.

The correctness of Conan has not been thoroughly nor systematically tested. This significantly undermines its validity and appeared to be the most prominent reason, according to Thierry Coquand (see section D.1), for which he could not make definite comments on whether Conan was suitable as an educational aiding tool.

The definitions for accessibility could be argued to not affect what constitutes a satisfactory educational aiding tool. It may also be subjective what should be considered 'easy to install' and that the criteria therefore should not carry considerable weight when evaluating proof editors.

7.2.2 Satisfaction Criteria Assumptions

In section 1.1, a number of assumptions regarding pedagogy were made. The most apparent problem with these assumptions is that they are not based on any literature study but rather our own experiences as students. We want to examine each of these assumptions, starting with the first which reads as follows:

Providing students with instant feedback by verifying each rule application would aid the students learning.

A very valuable aspect of doing exercises is that it allows you to test your understanding of the material. It allows the student to catch misunderstandings that they may have. For this purpose, textbook exercises are often accompanied by solutions with which an attempted solution can be compared.

The problem with this approach is that there could be many ways to prove the same thing. This means that a student might use a different approach than the proposed solution and would thus not be able to get feedback about the correctness of their proof. Another problem with proposed solutions are that students might look at the proposed solution without actually doing the exercise. They might then incorrectly conclude that they would have been able to solve the problem themselves. Having an editor that verifies proofs could solve these problems.

It is possible that an editor verifying proofs would provide too much help which could actually hinder learning. However, the key aspect of writing proofs is in the choice of which rule to apply. Using Conan, this choice still needs to be made by the user. It is conceivable that a student might try a trial and error approach, randomly applying rules until finding ones that verify. Theoretically it would be possible to reach the desired conclusion in this way, without having to actually understand the

logic. The editor verifying a particular rule application does not however mean that you have gotten any closer to the solution.

There is in fact an infinite amount of ways to reach a particular conclusion since there are many ways to apply rules where the result is redundant. On top of this, you not only have to choose the rule to apply, but also what lines in the proof to apply it to. This means that for each redundant line it is more likely that a random rule application will be redundant itself. For this reason, randomly guessing, with the help of the verification of the editor, is not really a feasible approach for reaching the desired conclusion.

The second assumption:

It is pedagogically sound for an educational aiding tool to represent the subject matter in a fashion similar to the representation in the learning material.

There are two primary ways in which this editor is intended to aid in the learning of writing proofs. First, it should make it easier, and perhaps more fun, to write proofs. Secondly, it should provide quick feedback to help students quickly realise if their understanding is correct or not. For this purpose it is beneficial if the transition from the learning material to the editor is as smooth as possible. The students should not have to spend unnecessary time learning special notation or syntax for the editor. The visual representation of proofs, like the box notation when making assumptions, should not introduce new obstacles for the student. It is also reasonable to think that the learning material, such as the textbook that we have modelled our editor after, has a pedagogical basis for its way of representing the material. It would then be unwise for us to choose a different representation without good support for why it might be better.

The third and final assumption:

It is pedagogically sound for an educational aiding tool to include no more material than is required by the corresponding field.

A focused interface is easier to understand since there are less options to consider and less visual clutter. This would help the student spend less time learning how to use the editor and thus have more time for actually learning the material. It could be argued that an editor with capabilities outside the scope of what the student is learning could encourage curiosity-driven exploration of related material which might in the end lead to the student learning more. This is, in our opinion, outweighed by the increased complexity that it could introduce.

7.3 Scientific Validity of Methods

The shortcomings of the methods that were used to test Conan are presented in this section.

7.3.1 User Tests

The result of the user tests was sufficient to fulfil our criteria. However, since the sample size was quite small the validity of the result could be questioned.

The user tests had additional factors that are important to consider when looking at the results. The testers might have been biased as they were all familiar with us, meaning they could have held back with their criticism. The tests were also held in an informal environment which adds to the question of how credible the testers criticism was. Furthermore all testers were Swedish citizens and students from Computer Science[1] or IT[2] at Chalmers University of Technology within the same age group. The lack of diversity further compromised the validity of the test. Another aspect to the ones who did the tests are that they were asked to do the tests. This means that these testers possibly liked testing new things and therefore might not have been representative for all students. The contrary might also be true since the testers were familiar with us they might have agreed to do the tests only to be nice.

Several of the testers said that they felt stressed when they knew they were being timed. This might have contributed to the testers, for example, feeling the urge to compete for the best time, not being able to focus on the task and not reading the tooltips properly. The lack of testing was also troublesome when doing the user tests as there was a bug in the editor. Every user tester faced this bug and had to be told that it was a bug after constructing the third proof. Two of the developers did not know about this bug in the editor which caused the two times that are outliers in figure 6.3.

The questions asked after the users had done the tests might have been ambiguous, especially the questions using a scale of 1-5 to rate certain aspects of the program. An example of this was the answer of one of the user testers: '4, because I don't know what 5 would be and it would feel weird to choose that. I don't know what it is, but it (Conan) can probably become better in some way.' There was a notable possibility that, although they did not say it, this might have been the case of the other testers too.

We are aware of that this scenario, with students entering completed proofs, is not the only way Conan will be used. Some students will try Conan with limited knowledge about logic. They will try the rules that exist and apply the most, to

[1] https://www.chalmers.se/sv/utbildning/program-pa-grundniva/Sidor/Datateknik.aspx
[2] http://www.chalmers.se/sv/utbildning/program-pa-grundniva/Sidor/Informationsteknik.aspx

them, logical rules to find their answer. Through this method the users will probably come across more or different problems than those who only wrote completed proofs. Though testing with the added factors knowledge and problem solving would give a less reliable result. It would either require a much larger sample size than the one in this thesis or an evaluation of each user tester to assess their level of knowledge regarding logic and how well they solve problems, neither of which we had time for. Furthermore it could be argued that this approach would compromise the validity of the test because the user tests were performed in order to assess if Conan had a low learning curve. Therefore, the only aspect of interest was whether or not the testers could construct a proof that requires quantifiers and subproofs to reach the conclusion without the help of others.

7.3.2 Heuristic Evaluation

Even though we were aware of the risk for bias, the heuristic evaluation (see section 6.2) was performed by us due time constraints. We followed Nielsen's Heuristics [14] and tried to be as objective as possible but the risk of us having been too critical or too lenient could have affected the conclusion regarding the satisfactory criteria. Given more time, the heuristic evaluation would have been done by other people than the developers. The reason we chose Nielsen's Heuristics was that it was one of the most used heuristic for evaluating GUIs, as is indicated by Nielsen's paper 'Heuristic evaluation of user interfaces' has been cited more than 5500 times on Google Scholar.

Since certain kinds of errors are intended to be present in Conan, it should be clarified why this heuristic received no negative remarks. The evaluators argue that two different categories of errors exist in this kind of program. First is what could be called *system errors*, which here is defined as errors that the program should handle. These could be unexpected crashes or failure to save a proof to disk. Second is *logic errors*, which here is defined as errors the user made in their construction of a proof. Logic errors are in turn divided into *shallow errors* and *deep errors*. Shallow errors are errors arising from the user misunderstanding how the program interprets their input (e.g. that a box must be referenced in its entirety). Deep errors are errors arising from the user misunderstanding the logic used. When performing the heuristic evaluation, deep errors where considered to carry pedagogical value and did therefore not affect the remarks for this heuristic.

7.3.3 Discussion of Demonstration

Details about the demonstration and the full interview can be found in appendix D.

Although Thierry Coquand[3] was a prominent person in computer science and, as

[3] `http://www.cse.chalmers.se/~coquand`

the examiner in the course *Logic in Computer Science*, suited for helping us know what was required to fulfil Conan's full potential there was no guarantee that his thoughts and ways of doing things were absolute. Due to only having a single teacher as evaluator a comparison was not possible. Coquand indicated that the program might be useful, if the verification was correct, which might be true for the course he was tutoring, while other courses need functionality that was not present in Conan.

7.4 Pedagogical Impact of Using Conan

Ideally, to truly determine the impact using Conan would have on a student's learning, a study comparing learning outcomes could be conducted where half a class of a logic course would be allowed to use Conan while the other half would stick with pen and paper (or another editor).

In section 1.1.1, the assumption that verifying proofs would be helpful for learning is made. In section 7.2.2, arguments for the validity of this assumption are made. Even with this assumption, some uncertainty for how Conan should function remain. When a rule does not verify, it might be useful to provide the user with information on why. From a usability perspective this is obvious. From a pedagogical perspective it is more problematic.

More precise information on why a rule application is not correct can certainly help a student to quickly identify a potential misunderstanding of the material. However it could also be the case that the process a student would go through to understand the error on their own might lead to a deeper understanding.

This is where the problem of selecting the appropriate level of help lies. Too much help and you take away the students need to reason on their own. Too little and learning might be needlessly inefficient. On top of this, usability has to be considered since you do not want the user to have to spend time trying to understand how to input the information correctly for Conan to function.

The level of help currently in Conan tries to balance this by only providing information about errors that could indicate a misunderstanding in how Conan functions. For example, Conan will not verify a row if it references a row that itself is not verified. This does not necessarily mean that the rule application is incorrect. Conan will therefore inform the user in this case.

Another example is the $\rightarrow$-e rule. This rule requires two references, an implication to eliminate and a formula matching the left hand side of the implication. This produces the right hand side of the implication as a result. If neither of the referenced rows contain an implication or the other reference does not match the left hand side of the implication, the user could be informed of this error. This error does not however indicate a misunderstanding of how Conan works, but rather a misunderstanding of the $\rightarrow$-e rule. Here no information will be provided, other than

the fact that the rule application is incorrect. Since the aim is for Conan to be used as a supplementary tool for students taking a course in logic, the students are expected here to consult their learning material to determine why the rule does not verify.

Conan aims to err on the side of caution here, preferring to provide too little help rather than too much.

7.5 Potential Environmental, Economical and Social Impact

We believe that this kind of educational aiding tools have potential for impacting the environment in a positive way and that their use cause a low negative impact on the environment. We argue that the energy required to power Conan specifically is almost negligible. Conan does not require substantial computing power and since the program is run locally on the user's machine, external services are only requested at download. We believe that education could have a net positive effect on the environment, because it enables more sustainable technical solutions. However, one could argue that technology thus far have had a negative impact on the environment.

We believe that Conan in most ways have a neutral to positive economical impact. The program is provided free of charge and does therefore not require students or universities to purchase additional software. However, we have not investigated to what extent universities spend resources on acquiring this kind of software and will thus refrain from stretching this argument further.

We believe that the social impact of Conan is dependant on existing standards of living and differences in social classes. Since Conan must be run on a computer, the users become more reliant on the technology. If some of the users do not have easy access to this technology, they are at a disadvantage compared to the others. Since we have no data on how using Conan affects learning, this argument should be taken with caution. It is possible that students who do not have access to Conan would learn the subject to the same extent using only traditional methods.

7.6 Differences Between Conan and Existing Proof Editors

The full evaluation of *NDProofs*, *JAPE* and *Logica* can be found in the pre-study (see appendix A). There are certain functionality that the other editors had that Conan did not such as with *JAPE* it was possible to define ones own logic and *Logica* was run directly in a web browser.

The main factors we argue Conan did better than the evaluated editors were the following:

Discrepancy between editors and representation in literature: Conan had less discrepancy, between how the editor represented proofs and how literature represented proofs, than the evaluated editors.

Intuitiveness: The evaluation of the other editors showed that if the user had no knowledge about them it was problematic to perform several common operations. They presented information and functionality with insufficient regard to whether it was important or not. Recommended points of entry were not emphasised enough to guide the user in using the editors. All evaluators had to refer to either help sections or third party help to be able to solve even basic proofs. During the ten user tests that were conducted (see section 6.1) for Conan none of the testers had to refer to the help section to be able to complete basic proofs.

Input control: Conan offered the user both buttons to enter symbols that were not commonly found on keyboards and shortcuts such as typing "ne" or "no" to get the negation sign (¬). The other editors did not have the latter functionality. The user was limited to using the mouse for the most parts which made them clunky to use, especially for the more advanced computer user.

Graphical User Interface: *NDProofs*, *JAPE* and *Logica* had two notable thing in common: they all used basic standard design and grey colours. Layout of each editor was limited. Either nothing was visible or everything was visible all the time. In contrast Conan had a GUI with two themes, commonly used buttons were visible while buttons which were used less were in drop down menus, highlights with colours whenever something was wrong or if the conclusion had been reached.

The user was much closer to having the freedom to edit the proof the same way that could be done with pen and paper. None of the existing editors were able to edit something in the middle of a proof. This means that if there was an error everything done after the error had to be removed. With Conan the user could freely edit anything in the proof.

7.7 Contributions of this Project

Since students find formal techniques difficult to learn [9], it would be beneficial if proof editors targeted to beginners were readily available. As previously discussed (see section 7.1) Conan fulfils the definitions set up for a satisfactory educational aiding tool. Because these definitions were set up in order to address the deficit in satisfactory proof editors found in the pre-study (see appendix A), this project has contributed to reducing this deficit by the development of Conan. It should be clarified that this does not imply that Conan in general should be considered a 'better' proof editor than other available ones, just better suited in certain instances.

In particular, the results show that Conan has an intuitive interface with a low learning curve that closely follow commonly used course material.

7.8 Future Development

A feature to be implemented in future development was brought to attention by Thierry Coquand (see appendix D). Instead of working forwards from the premises, the user may want to work backwards from the conclusion. The current implementation does not aid the user in working backwards. However the interface allows working backwards, and only the back-end needs to be modified. An example of generating an expression backwards would be if we have $A \wedge B$ that is justified by the $\wedge i$-rule, we could generate A and B, for the rows that the line references. This would allow the user to work backwards while having the option of generating rows. Note, that the user is still able to construct the proof from anywhere and both directions, however, currently both verification and generation works from top to down.

Here is a more explicit example:

1.		
2.		
3.	$A \wedge B \wedge C$	$\wedge$i, 1,2

would generate

1.	$A \wedge B$	
2.	C	
3.	$A \wedge B \wedge C$	$\wedge$i, 1,2

Implementing additional rules is not difficult, since the abstract Java class we defined is easy to understand and extend. However, implementing the functionality to derive rules and apply them inside of the program is not as simple. The GUI needs to be able to handle the input of any number of references that are needed for the rule, compared to the current state of a maximum of three references (interval or single row). Additional back-end logic is also needed to connect how the premises relate to the conclusion and each other.

Some features are easily added in the next possible version such as tab-key navigation and customised shortcuts. Additional polishing of the GUI and UX may be done, such as fixing small issues with the styling and making the interface more intuitive for users. Adding a tree visualisation of the proof should not be difficult to implement (see 2.4), as the underlying data structure for the box may be converted to a tree-style data structure without much effort. Exercises may be created using the saving and loading of proofs, but additional built-in tools can be helpful to track progress

and provide extra guidance for given exercises.

An interactive tutorial or video tutorial can be implemented to help the users familiarise themselves with the program faster. This might also be used to help users learn different rules, taking into consideration the difficulties the user might have with some of them [12]. Having loadable proofs tailored to give extra practice to a certain rule with or without interactive help from Conan is another possibility. If there is interest for a more helpful version of Conan numerous features can be added: informing the user of explicit errors in their proof strategy, autocompletion for references and formulas and a suggestion menu for when there might be multiple valid references. An Android and iOS version should be possible as is, through JavaFX compilation, but this has not been tested, and it might not be well optimised for those devices. Finally, a web version of the application can be made, using the underlying back-end on a server and rewriting the GUI with HTML, CSS and JavaScript.

7.9 Future Research

Further potential research that arises from this thesis is to examine the program from a pedagogical perspective in order to determine if the program is deemed pedagogical. The project was approached without any background knowledge in pedagogy, so there are potential flaws from the assumptions that were made during the development of the program. This can be done by comparing the use of the editor with pen and paper.

If Conan through future research is deemed pedagogical then there is the possibility to create satisfactory educational aiding tools for other subjects using similar approach as this project. A natural continuation would be to develop an educational aiding tool for natural deduction in other types of logic. These satisfactory educational aiding tools can then be further evaluated for its usefulness in teaching.

7.10 Conclusion

From the results in this study, it can be concluded that Conan meets our criteria (see section 1.1) of a satisfactory educational aiding tool for students learning natural deduction in first-order logic. However, there is no guarantee that these criteria correspond to the real-world requirements of a satisfactory educational aiding tool. Therefore, further investigation is required in order to determine Conan's effect on learning natural deduction in first-order logic.

Bibliography

[1] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd ed. Cambridge: Cambridge University Press, New York, 2004.

[2] G. Gentzen, "Untersuchungen über das logische schließen. i," *Mathematische zeitschrift*, vol. 39, no. 1, pp. 176–210, 1935.

[3] J. Beall and G. Restall, "Logical consequence," in *The Stanford Encyclopedia of Philosophy*, winter 2016 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2016.

[4] J. Jonasson and S. Lemurell, *Algebra och diskret matematik*, 2nd ed. Lund: Studentlitteratur AB, 2013.

[5] M. Davis, "The early history of automated deduction." *Handbook of Automated Reasoning*, vol. 1, pp. 3–15, 2001.

[6] A. Newell and H. Simon, "The logic theory machine–a complex information processing system," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 61–79, 1956.

[7] W. Bibel, *Early History and Perspectives of Automated Deduction.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 2–18. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74565-5_2

[8] D. Prawitz, H. k. Prawitz, and N. Voghera, "A mechanical proof procedure and its realization in an electronic computer," *J. ACM*, vol. 7, no. 2, pp. 102–128, Apr. 1960. [Online]. Available: http://doi.acm.org/10.1145/321021.321023

[9] D. Goldson, S. Reeves, and R. Bornat, "A review of several programs for the teaching of logic," *The Computer Journal*, vol. 36, no. 4, pp. 373–386, 1993.

[10] S. Obua, "Proofpeer-a cloud-based interactive theorem proving system," *arXiv preprint arXiv:1201.0540*, 2012.

[11] C. K. F. Wiedijk and M. H. F. van Raamsdonk, "Teaching logic using a state-of-the-art proof assistant," *Formal Methods in Computer Science Education*, p.

111, 2008.

[12] J. Aczel, P. Fung, R. Bornat, M. Oliver, T. O'Shea, and B. Sufrin, "Using computers to learn logic: undergraduates' experiences," in *Advanced Research in Computers and Communications in Education*, O. T. . G. L. Cumming, G., Ed. Amsterdam, IOS Press, 1999, pp. 875–882. [Online]. Available: https://telearn.archives-ouvertes.fr/hal-00190200

[13] H. Van Ditmarsch, "User interfaces in natural deduction programs," 1998.

[14] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94. New York, NY, USA: ACM, 1994, pp. 152–158. [Online]. Available: http://doi.acm.org/10.1145/191666.191729

[15] "Ten heuristics for user interface design: Article by jakob nielsen." [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

Appendix A

Pre-study of Existing Proof Editors

The purpose of this pre-study was to investigate if there was a deficit in proof editors targeted to students learning natural deduction in first-order logic. The initial hypothesis was that a large number of universities do not promote proof editors because the available options either lack in visibility or are unsatisfactory. In addition, the pre-study should outline usability criteria for proof editors aimed at the target group.

Because university courses are subject to time constraints, students have limited time to learn any tool they are expected to use in the course. Therefore, it is important that a proof editor is easy to use and has a low learning curve. Furthermore, it is essential that the tool does not significantly infringe on the student's learning. Although further pedagogical studies may be required, it seems safe to assume that a tool that automatically solves the student's exercises would counteract their learning. In other words, a proof editor should not be a tool that students learn to operate rather than to use.

In order to find evidence for the claim that there was a deficit in proof editors aimed at the target group, an appraisal of existing proof editors was conducted. Due to time constraints, the appraisal was not exhaustive in two aspects. First, the Internet was searched for available proof editors (see table A.1). Second, most of the found proof editors were rejected because they were deemed irrelevant for this study. Examples of reason that could cause a proof editor to be rejected include a complex interface, automatic proof solving or presenting a different type of logic. It should in particular be addressed that several frequently used interactive theorem provers (e.g. ACL2, Isabelle and HOL) were not considered relevant options, because of their perceived complexity. In the end, the appraisal was performed on three proof editors that seemed most relevant: *NDProofs* (v0.99b) [1], *JAPE* (v7d15) [2] and *Logica*

[1] `https://github.com/Ohohcakester/NDProofs`

[2] `https://www.cs.ox.ac.uk/people/bernard.sufrin/personal/jape.org`

(version unknown) [3].

Table A.1: The data sample of proof editors available on the Internet.

ACL2
APros
Aristotle
Athena Software
Bertie and Twootie
Bertrand
blobLogic
Blogic
Deriver (SoftOption)
Expression Evaluator
HOL
Isabelle
Jape
Logica
Logitext
NDProofs
proofWeb
uProve

The appraising was divided into three main parts: accessibility, overview and heuristic evaluation. The accessibility part was an estimation of how easy it would be for students to acquire and install the proof editor. The overview part was an informal judgement of how intuitive the proof editor is to use. The heuristic evaluation part is a heuristic evaluation of the proof editor's GUI, using Jakob Nielsen's heuristics [14]. In order to aggregate the heuristic evaluation, the proof editor received for every heuristic a remark signalling to what extent the editor diverged from the heuristic. The remarks were, in order of severity, labelled as minor, significant or major.

The main finding of the appraising was that the three proof editors were unsatisfactory. As a result, it was regarded to be motivated to engineer a prototype proof editor for students learning natural deduction in first-order logic. Since the proof editors displayed an unsatisfactory GUI, it is recommended to further investigate how this problem can be addressed in the prototype.

A.1 NDProofs

NDProofs is a free, open-source proof editor for constructing propositional proofs using natural deduction. The editor does not live up to the standards set by the

[3] `http://logica.stanford.edu/logica/php/fitch.php`

evaluators. All in all, the evaluators found no situation where they would have preferred the editor to pen and paper.

A.1.1 Overview

The results of the appraisal suggests that users do not intuitively understand how *NDProofs* is intended to be operated. Figure A.1 depicts the program GUI at start up. The amount of visual components could easily confuse the user. Recommended points of entry are not emphasised enough to guide the user in operating the software. Although tutorials were available, they proved to be unsuccessful in providing a clear understanding of how the software should be operated. Although it could be argued that the user over time would grow familiar enough with the interface to utilise it effectively, the evaluators do not believe that the target users are willing to spend sufficient time nor energy to grow accustomed to the interface.

The evaluators' opinion on the GUI is that it in its current form significantly inhibits the user's work flow. There are a number of problems with the program's design. Firstly, there are visual elements that are not displayed properly with executing the software on a Linux machine. As can be seen in Picture A.1, some buttons are unable to show their text, leaving the user to guess their behaviour. Secondly, visual elements that are not clearly related are considerably similar in appearance and put in close proximity of each other, without the GUI providing any hint to the underlying reason of their grouping.

A.1.2 Accessibility

The appraisal found that *NDProofs* is easily accessible. Because the software is freely available on the Internet, it is easy to acquire by potential users. The software is aggregated in an executable JAR file, which makes it require minimal effort to install on the most common operating systems. Executing JAR files prerequisite a Java virtual machine, which could be argued to negatively impact the accessibility. The stance taken by the evaluators was that the Java Runtime Environment is a common software package that, if not already installed, is sufficiently easily accessible to not make this dependency significant.

A.1.3 Heuristic Evaluation

The heuristic evaluation found remarks in the following areas: User control and freedom, Consistency and standards, Error prevention, Flexibility and efficiency of use, Aesthetic and minimalist design, Help users recognise, diagnose, and recover from errors, Help and documentation.

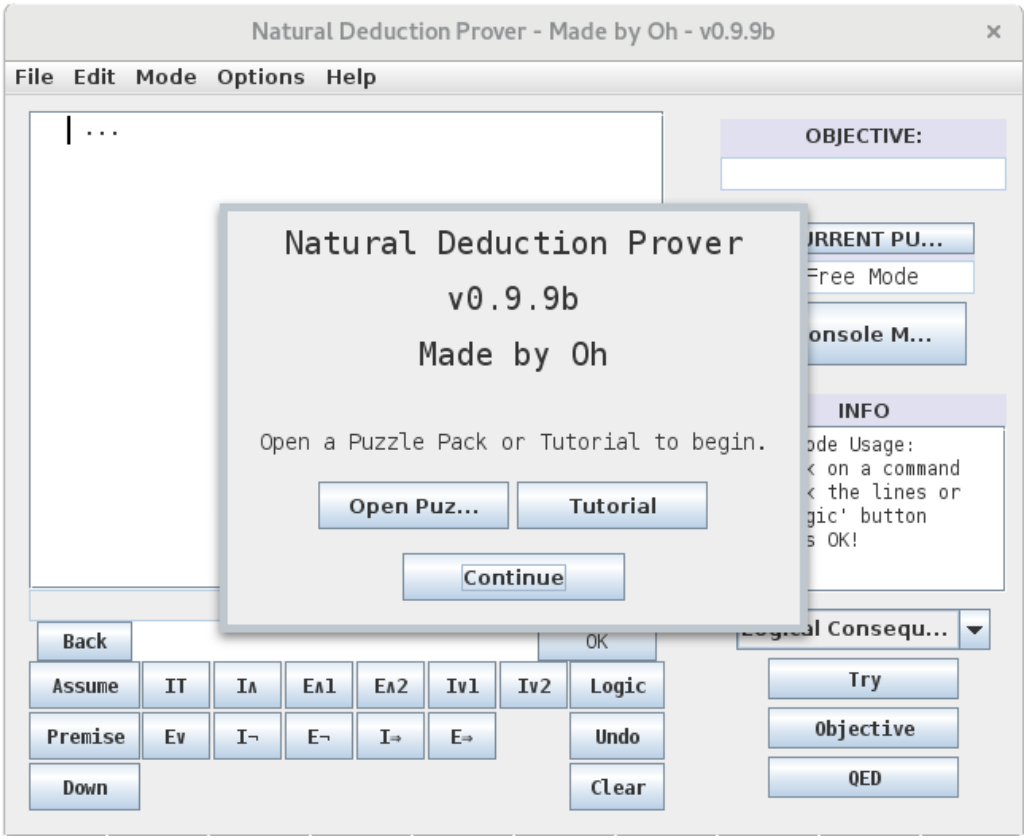


Figure A.1: A screenshot depicting the look of *NDProofs* at startup on a Linux machine.

The program received significant remarks on User control and freedom. There were no clearly marked emergency exit options for leaving unwanted states. The program did not support the redo operation, in spite of supporting the undo operation.

The program received significant remarks on Consistency and standards. Several conventional interactions were unavailable within the program. Users were among other things unable to access context menus and alter the application window size.

The program received major remarks on Error prevention. Actions that would result in errors are not greyed out, making users more prone to commit slip type errors. Similarly, taking an actions that require specific parameters do not make the remaining parameters unavailable. In order to clarify, two examples (see examples 1 and 2) are provided below.

Example 1. Implication elimination

The implication elimination rule requires the second parameter to be a line containing an implication expression. As can be seen in figure A.2, the second line does not stand out compared to lines one and three, despite being the only valid choice.

Example 2. Open Puzzle Pack

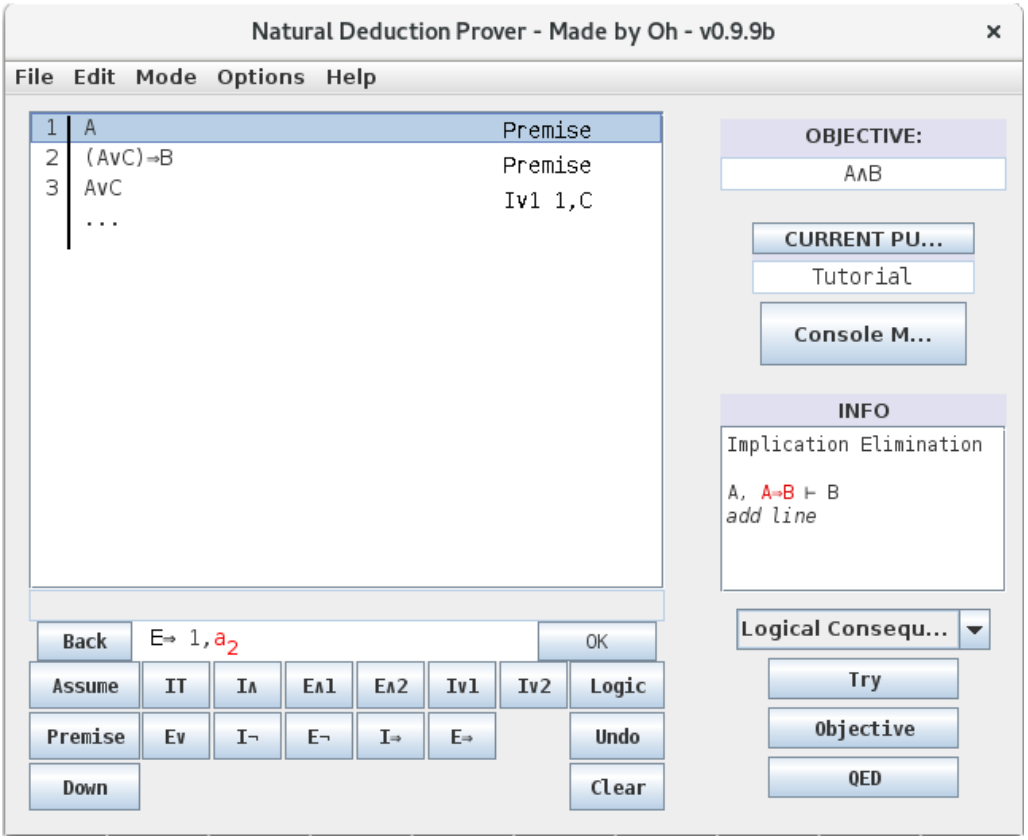


Figure A.2: A screenshot depicting the implication elimination action requiring the second parameter to be a line containing an implication expression.

The program provides the option of loading proofs from the file system, but requires the file format .ndpack. As can be seen in figure A.3, the program does not allow the user to view only files of the required format.

The program received significant remarks on Flexibility and efficiency of use. Several useful accelerators (e.g. typing a in order to perform the assumption rule action) were implemented, but were stumbled upon by chance because the GUI provided no visual queues for their existence. There were several actions that the evaluators thought would be frequently used (such as performing the undo action) that do not appear to be reachable through accelerators. Menu mnemonics were not provided.

The program received minor remarks on Aesthetic and minimalist design. The GUI put unnecessarily high cognitive load on the user. Several visual elements took up information space disproportional to their relevance.

The program received minor remarks on Help users recognise, diagnose, and recover from errors. Error messages were often too broadly stated to accurately diagnose the problem. As a result, the user was not provided with accurate suggestions for solutions.

The program received minor remarks on Help and documentation. Although a rather

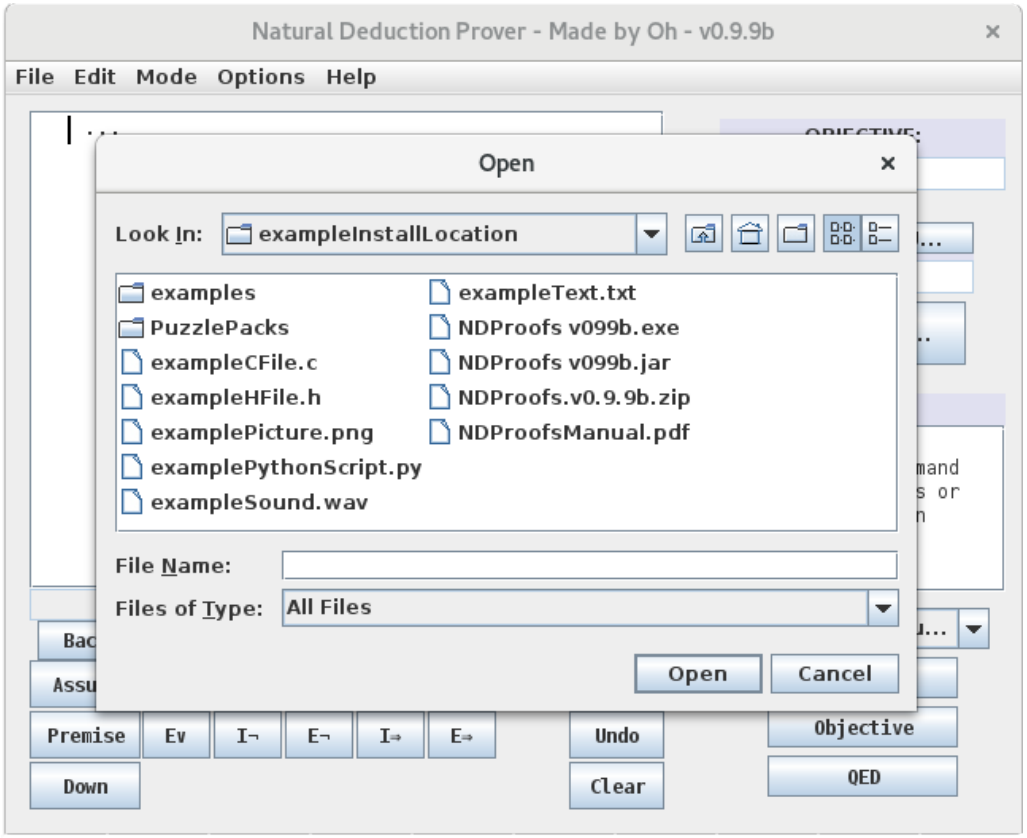


Figure A.3: A screenshot depicting the implication elimination action requiring the second parameter to be a line containing an implication expression.

extensive manual was included when the program was downloaded, large portions of its contents are unavailable through the GUI. The portions available through the GUI provided insufficient help.

A.2 JAPE

JAPE is a proof assistant for natural deduction and variants of sequent calculus created by Richard Bornard and Bernard Sufrin. Users can define their own logic, choose how to view proofs and much more. It comes with examples of proofs that can be solved. The proof assistant is a free to use Java program.

A.2.1 Overview

JAPE does what it is intended to do, nothing more. It is lacking severely in being intuitive and user friendly. It is hard to manoeuvre and in general clunky to use. *JAPE* may be preferred over pen and paper because it verifies proofs and has good error messages when trying to apply a rule that is now allowed.

A.2.2 Accessibility

JAPE can be downloaded to use on Mac, Linux and Windows from the official website . Installation on Linux proved to be troublesome due to the engine file that *JAPE* use must be placed in the users home folder even though the other files are located elsewhere. During installation on Windows the same problem was not encountered so it might have been a problem for Linux or the evaluator's computer.

A.2.3 Heuristic Evaluation

JAPE had a strong point in having a short, with the option of a longer more thorough, explanation about why a certain rule couldn't be applied. *JAPE* also supports both forward and backward reasoning, enabling for proofs to be solved in the most suitable way.

JAPE received significant remarks on Help and documentation Aesthetics and minimalist design. The basic task of selecting two expressions to apply a rule on had to be found through trial and error and perhaps only due to the fact that the evaluator has long experience with computers. When clicking the help menu the user is faced with the simple sentence: 'There is no generic help for the interface'. What kept this from being a major remark is that the program always give a concise explanation of why a rule can't be applied with the option of a longer explanation. Although the program is minimalistic, important and commonly used functions aren't visible and more accessible than unimportant and rarely used functions. Furthermore the program opens everything in separate windows which can be seen in figure A.4, that can't be closed without closing the main program is highly annoying.

Major remarks were given on Recognition rather than recall and Flexibility and efficiency of use. Significant remarks on Help and documentation. Each time a rule was to be applied the user had to drag the mouse to a drop down menu and there find the wanted rule which makes the program time consuming by requiring several clicks to perform an often repeated task. The user must remember exactly where each rule can be found as everything in the program looks the same. The only useful accelerators that were present were undo and redo. The user is forced to using the mouse. The absence of useful visible functionality can be seen in figure A.5.

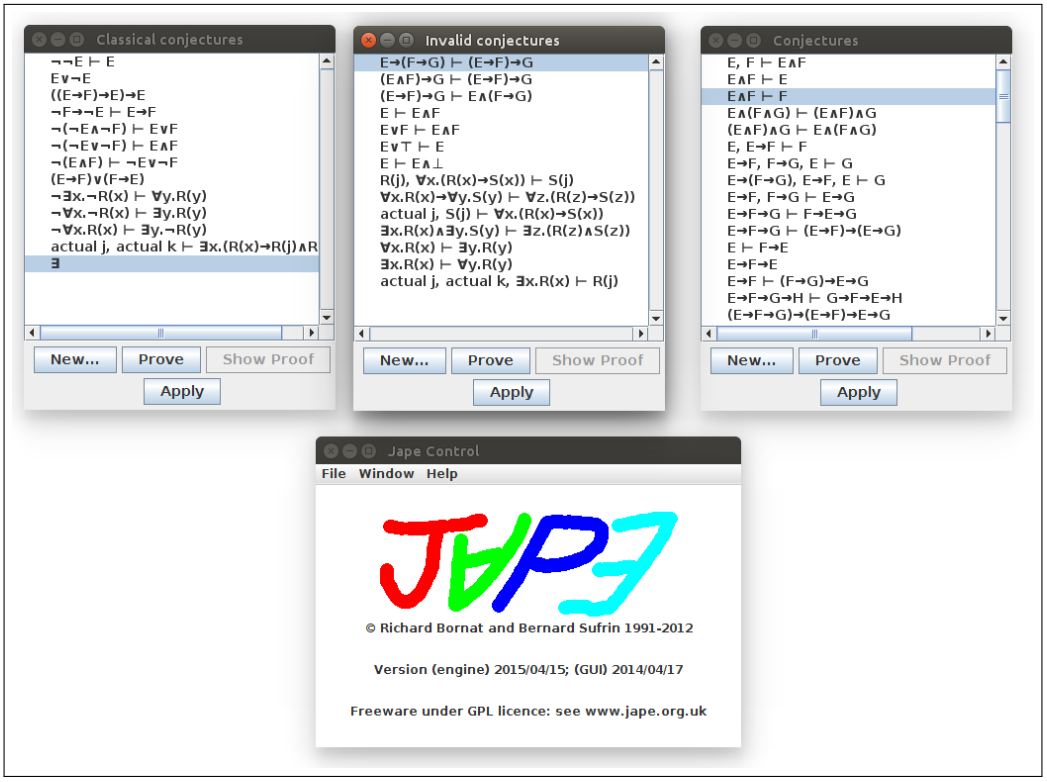


Figure A.4: A screenshot displaying how *JAPE* use multiple windows that can't be closed separately.

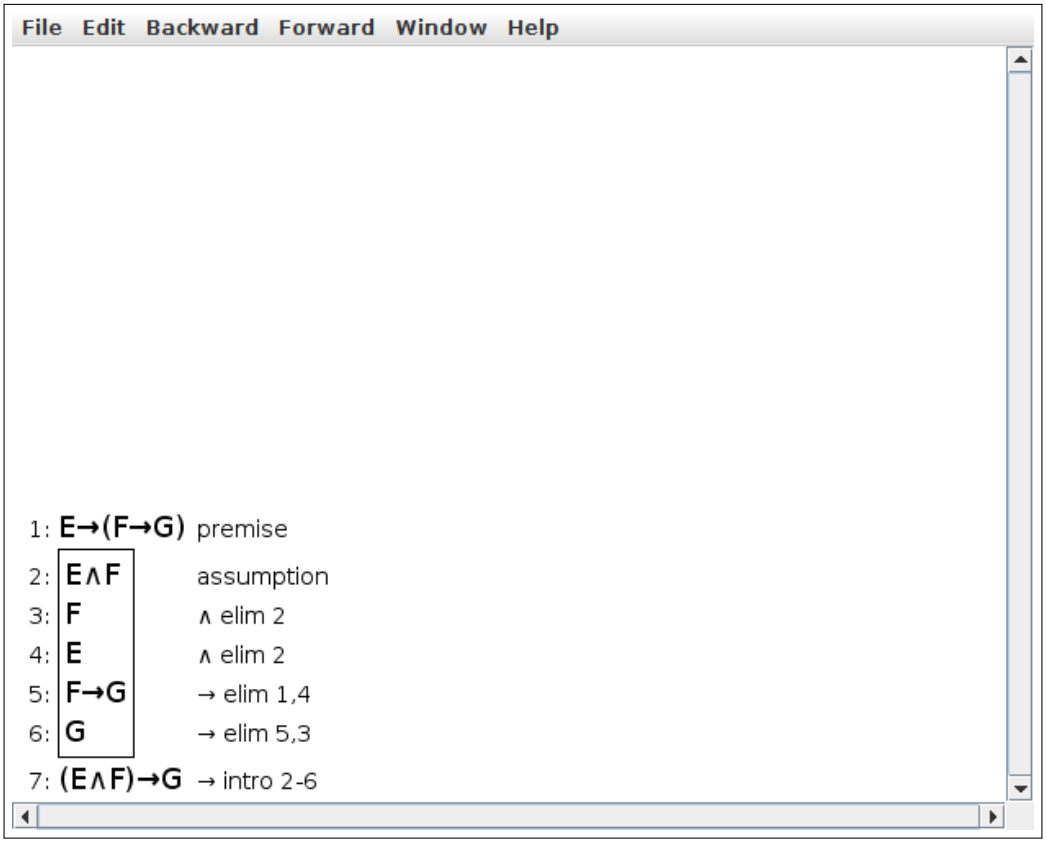


Figure A.5: A screenshot displaying how a proof looks in *JAPE*.

A.3 Logica

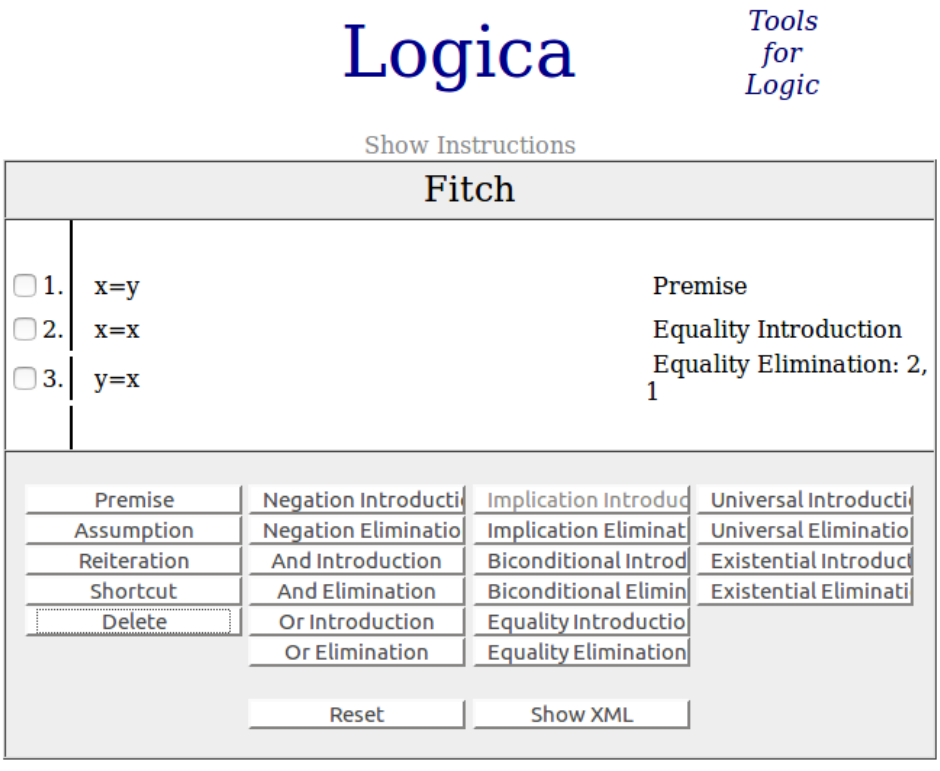


Figure A.6: A screenshot showing the interface of Logica

A.3.1 Overview

Through our evaluation, Logica was deemed unsatisfactory for writing proofs. A better alternative would be to simply use pen and paper. The interface is cumbersome and poorly documented, you can store proofs as XML but cannot load them and there is virtually no feedback when a rule application is rejected. In some cases, the evaluator was unable to finish simple proofs since there is not sufficient information on how the editor functions.

A.3.2 Accessibility

The editor is available online and runs in the browser, making it very accessible.

A.3.3 Heuristic evaluation

The primary remarks were found in the following areas from Nielsen's heuristics: User control and freedom, Flexibility and efficiency of use, Help users recognise, diagnose, and recover from errors and finally Help and documentation.

User control and freedom: There is no way to go back and edit existing lines. To correct mistakes one have to go through and delete everything that follows the incorrect line. There is no undo/redo.

Flexibility and efficiency of use: The input methods are very limited. You click to select lines and rules, entering additional information in a prompt. There seems to be no keyboard commands or shortcuts.

Help users recognise, diagnose, and recover from errors: The only error message that appeared during the evaluation was to inform the user of the need to select at least one line from the proof before applying the rule. No other information is provided about user errors.

Help and documentation: Only minimal documentation is provided and was insufficient for the evaluator to be able to finish some basic proofs.

Appendix B

Learning Materials Used by Universities

This appendix shows the data collected when investigating what course material was most frequently used.

Table B.1: Textbook recommendations for universities in a course where natural deduction is taught. The textbooks are indicated by the indices used in table B.2 and are presented in order of appearance on the course home page.

University	Index of recommended textbooks
Academia Sinica	19
Australian National University	16, 10, 5, 32, 4
Chalmers University of Technology	19
Drexel University	19, 3
Imperial College London	29, 15, 30, 2, 11, 19, 7, 20, 23
Indian Institute of Technology Bombay	19
KTH Royal Institute of Technology	19
Leiden University	19
Linköping University	22
Mälardalens University	19
National Taiwan University	9, 12, 18, 2, 17, 25, 7, 6, 27
National University Singapore	—
Stockholm University	14, 17, 8
UC Berkeley	24, 19, 13, 33
University of Bath	19, 28, 1
University of Cambridge	19, 22
University of Gothenburg	19
University of Graz	19
University of L'aquila	—
University of Liverpool	19, 31
University of Michigan	19
University of Ottawa	19
University of Oxford	19, 26
University of Tsukuba	19
Uppsala University	21

Table B.2: The books recommended by different universities in a course where natural deduction is taught sorted in an ascending order. Note that not all of them are textbooks on logic.

Index	Name
1	Basic Proof Theory with Applications
2	Beginning Logic
3	Computer Aided Reasoning: An Approach
4	Discrete Mathematics for Computing
5	Discrete Mathematics with Applications
6	First-order Logic
7	Formal Logic: Its scope and limits
8	Första ordningens logik
9	Handbook of Mathematical Logic
10	Haskell: The Craft of Functional Programming
11	How to Prove It: A structured Approach
12	Intermediate Logic
13	Introduction to Embedded Systems: A Cyber-Physical Systems Approach
14	Logic
15	Logic[1]
16	Logic and Discrete Mathematics: A Computer Science Perspective
17	Logic and Structure
18	Logic for Mathematicians
19	Logic in Computer Science: Modelling and Reasoning about Systems
20	Logic: Techniques of Formal Reasoning
21	Mathematical Logic
22	Mathematical Logic for Computer Science
23	Methods of Logic
24	Model Checking
25	Natural Logic
26	Proof and Disproof in Formal Logic
27	Proof Theory
28	Proofs and Types
29	Reasoned Programming
30	Software Engineering Mathematics: Formal Methods Demystified
31	The Essence of Logic
32	The Logic Book
33	The SPIN Model Checker: Primer and Reference Manual

[1]There exists two books titled *Logic* written by different authors. Number 14 was written by Jesper Carlström and number 15 was written by Hodges Wilfrid.

Appendix C

Times for Constructing Proofs in the User Tests

This appendix shows the empirical data collected through the user tests.

Table C.1: Times for constructing proofs in the user tests by the beginner and experienced group and construction times for the developers. The times have the format minutes:seconds.

	Proof		
Tester	Proof 1 (m:ss)	Proof 2 (m:ss)	Proof 3 (m:ss)
Beginner 1	1:55	9:28	3:50
Beginner 2	1:44	5:45	4:15
Beginner 3	2:48	6:26	1:20
Beginner 4	3:14	3:49	3:51
Beginner 5	4:36	5:40	2:40
Experienced 1	2:44	6:18	1:49
Experienced 2	6:06	6:37	4:14
Experienced 3	2:56	8:07	1:35
Experienced 4	7:57	6:19	3:04
Experienced 5	2:52	5:36	1:25
Developer 1	0:21	1:58	0:48
Developer 2	0:30	4:04	2:20
Developer 3	0:45	4:40	2:18
Developer 4	0:50	4:15	5:35
Developer 5	0:40	5:50	6:40
Developer 6	1:16	6:36	2:11

Appendix D

Interview with Thierry Coquand

This appendix presents an interview with Thierry Coquand concerning Conan.

D.1 Demonstration of Conan for an Examiner of a Logic Course

A professor examining students in a logic course possess valuable knowledge about both the proofs that Conan is used for and common mistakes that students who might use Conan do. Thierry Coquand[1] is a renowned professor in computer science and examiner in the course *Logic in Computer Science* at Chalmers University of Technology. His feedback is therefore of great value after the editor has been demonstrated for and tested by him.

Coquand had the editor shown and explained to him before letting him use the program freely. The reason was because this demonstration was intended to result in a better understanding of the editor's potential as an educational aiding tool. Afterwards a set of beforehand decided questions were asked, his thoughts on the assumptions made on pedagogy and if he had any remarks on Conan.

D.2 Results From the Demonstration of Conan

Thierry Coquand was in general positive to the editor. He stated that to make more definite comments, on whether the editor would be useful in his course, he had to test the editor further which he showed interest in. He said that the assumptions made about pedagogy were good and reasonable.

[1] `http://www.cse.chalmers.se/~coquand`

Coquand's comments on our assumptions (bold text) about pedagogy and the questions asked (Q&A) after he had tested Conan are presented with corresponding answer below:

It is pedagogically sound to provide students with the option of choosing the amount of help they want: Yes, it sounds good.

It is pedagogically sound for an educational aiding tool to represent the learning material in a fashion similar to the representation in the learning material: Yes, it is good for the students as they are not confused by different views on proofs.

It is pedagogically sound for an educational aiding tool to include no more material than is required by the corresponding field: Yes, that sounds correct.

Q1: Do you think that the editor provides the necessary functionality of a proof editor?
A: From the quick look it looks okay.

Q2: On a scale from 1 to 5, how would you rate the editor's ability to edit proofs? Please elaborate.
A: Very difficult to say based on such a short demo but 4, maybe 5.

Q3: Are the logical conventions used in the program suitable?
A: Yes, it seems that you've followed the conventions in the book quite closely which is good.

Q4: Do you think that the editor could be beneficial to your course Logic in Computer Science?
A: Difficult to say, it has to be tested. If the verification of proofs is correct then it could be useful.

Q5: Do you think that Conan is a satisfactory educational aiding tool for students learning natural deduction in first-order logic? Please elaborate on your answer.
A: Impossible to say.

Q6: Do you have any suggestions for what you think would improve the editor?
A: Not at the moment. I will have to try it before making such a comment.

Appendix E

Non-exhaustive List of Proof Assistants in Lexical Order

This appendix lists, in lexical order, the proof assistants encountered when writing this thesis.

Table E.1: Proof Assistants encountered throughout the duration of this thesis.

Ωmega
ACL2
Agda
Albatross
ALF
Alfa
Apros
Aristotle
Athena Software
Automath
B method
Bertie and Twootie
Bertrand
blobLogic
Blogic
Carnegie Mellon University Proof Tutor
CLAM theorem proving system
Constructions
Coq
Deductive Tableau
Deriver (SoftOption)
Edinburgh Logical Framework
Educational Theorem Proving System
Expression Evaluator
F*

HOL light
HOL
HyperProof
ICLE
IMPS
IPE
Isabelle
Isar
Ivy
Jape
LCF
Lean
LEGO
Logic Theorist
Logica
Logitext Logitext
MacLogic
Matita
Metamath
MINLOG
Mizar
NDProofs
NuPRL
Otter
Panda
Pandora
PhoX
ProofPower
proofWeb
PVS
Symlog
Tarski's World
The LogicWorks
Theorem Proving System
Theorema
Twelf
Typelab
uProve
Venn
XBarnacle proof tool
Yarrow
Yoda

Appendix F

Development of Conan

Conan was written in Java 8, and here is a rough estimate of the effort required to code Conan.

Table F.1: Time spent coding Conan.

Part	Hours spent	Part	Hours spent
Back-end	150	Front-end	450
General	40	General	434
Parser	26	Undo/redo	16
Verification	70		
Generation	7		
Export to LaTeX	7		

600 hours total

Table F.2: Lines of code in Conan.

Part	Lines of code	Part	Lines of code
Back-end	4636	Front-end	3915
General	1044	General	2995
Parser	1328	Undo/redo	300
Verification and Generation	2164	FXML-file	381
Export to LaTeX	100	Stylesheets	239

8551 lines of code total