Mechanising Applying Rules by Elimination, Proofs by Induction and Proofs by Cases in the Holbert Proof Assistant

Christopher James Perceval-Maxwell

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2022

Abstract

Holbert is a new, web-based, interactive proof assistant built on *higher-order logic* and *natural deduction*. Holbert's primary focus is to be an educational tool for students; it aims to replace popular proof assistants that are geared more towards advanced users, such as Coq and Isabelle, in the classroom. Holbert, previously, was only practical for proving basic logical deductions. In this project, we have made it possible for the user to apply rules by elimination and, as a result, write proofs by induction and proofs by cases in Holbert. We also wrote a new unification algorithm to make these proof types possible. In this report we discuss why Holbert is a beneficial alternative to such popular proof assistants, considerations of its limitations and areas for further work.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Christopher James Perceval-Maxwell*)

Acknowledgements

I would like to thank my supervisor, Liam O'Connor, for their continuous support and motivation throughout this project. Their help has been invaluable for the completion of this project.

I would also like to thank my friends and family for their support throughout the entirety of my undergraduate degree.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Holbert [O'Connor] is being developed with the primary focus for improving the teaching experience for theorem proving, specifically for its use in teaching the foundations of programming languages, and related courses. Although many popular proof assistants (PAs), such as Coq [INRIA, 2009] or Isabelle [Nipkow et al., 2002], are currently used in the classroom, teachers and students alike face many challenges utilising such tools.

Proof by induction is an extremely important component of any proof assistant as it is the base for lots of computational reasoning. One of is many uses is in mathematics for proving a theorem holds for all the natural numbers, $\mathbb{N}$ – this is discussed in more detail in Section 2.1.3. Without proof by induction, Holbert would not be a complete proof assistant.

1.1.1 Proof Assistants in Education

Narboux [2005] discusses the introduction of proof assistants into the classroom, specifically for teaching mathematics. Some key advantages listed by Narboux are the "flexibility", "rigour", "clarity" and "interactivity" a proof assistant can offer. These attributes provide the means to more effectively: teach how to incorporate axioms and theorems into a proof; determine the correctness of a proof that is not reliant on the teacher; determine how each step arrives at its conclusion or why it cannot unify a rule with a goal. Pierce [2009] advocates for the use of an automated proof assistant in the classroom. Pierce describes proof assistants as the "Ultimate TA" (Teaching Assistant) – a personal tool available to the student for immediate, correct feedback.

Coq is another popular proof assistants employed in the classroom. Pierce discusses trialling Coq to teach a course on introductory programming languages theory, based on their book *Software Foundations* [Pierce et al., 2017]. This course had a syllabus that included functional programming with higher-order logic (HOL) in Coq, inductive proofs on functional programs, programming with propositions

(i.e. functional programming as logic), operational semantics and typing. Pierce believes that this trial was a great success, where students enjoyed using Coq as it enabled them to be more critical of their work. Narboux [2005] cites Guilhot [2004] and their contributions to a Coq development environment used for teaching theorems in high school geometry in their paper discussing the future of teaching mathematics using proof assistants. Agda [Norell, 2009] is another proof assistant utilised in the classroom, notably by Wadler et al. [2020] for teaching the course TSPL (Types and Semantics for Programming Languages), based on their book PLFA (Programming Language Foundation in Agda), at Edinburgh University.

Xena was a project run by Buzzard [2017a], at Imperial London College, that used the L∃∀N theorem prover as a tool for undergraduate mathematics students to check their work. Buzzard is currently using L∃∀N to teach their course "Formalising Mathemetics" [Buzzard, 2022]. Thoma and Iannone [2021, 2019] conducted research into whether Buzzard's approach (making use of their L∃∀N libraries [Buzzard, 2017b]) improves the learning experience for proofs, with a self-selected group of students. In this focus group, seven students had experience using L∃∀N, and twenty-nine did not. Although this study did not see improvements in the number of correct proofs written by students, the qualitative analysis concluded with many other positive aspects. Such aspects have been highlighted as characteristics that may help play a role in students being able to write complete proofs in the future. Students proved to be more accurate with their use of mathematical language; their style of writing proofs leaned more towards an academic one, and their flow of logical judgements to form a proof also showed improvements. However, they also discuss that it may still be necessary to devote time to teaching the programming of this language and that L∃∀N's UI and documentation is not easy for first-time users.

Narboux [2005] also discusses why the current state of proof assistants is not beneficial for teaching; Narboux notes many features that he believes need to be modified to improve the teaching experience:

- It is common for proof assistants to automate intuitive steps of a proof. However, this is counter-intuitive for a teaching tool. The teacher may want to display, or have the student perform, all the steps of the proof.
- The error messages thrown when a rule cannot be applied to the current goal are unclear, especially for those with no experience in logic or theorem proving.
- The Graphical User Interface (GUI) and basic *pretty-printing* can be intimidating for novice users. Current proof assistants can be hard to navigate and, at times, it can be unclear where to locate certain information in a proof. Pretty-printing formats syntax trees into text. An example of this is "$x^2 \cdot \frac{y}{z}$" which could be translated from "`x^2*(y/z)`", or for example in Holbert, "`exists (x. _/\_ (P x) (Q x))`" to "$\exists$ (x. (P x) $\wedge$ (Qx))".

Kokke et al. [2020] discuss writing their course PLFA, arriving at the conclusion that Coq should not be used for teaching programming language foundations as

too much time was spent teaching "learning tactics" rather than the course content. Eastlund et al. [2007] talks about using the ACL2 proof assistant [Kaufmann et al., 2000] to teach first-year students computational logic topics. He highlights that students found that ACL2's feedback was "overwhelming".

Holbert has distinguished itself from popular proof assistants by fine-tuning areas to improve the user experience for teachers and students alike. Features that Holbert introduces that are absent from such tools are:

- Since Holbert is entirely browser-based, it is easy for all students to use as there is nothing to install and there are no device requirements other than an internet connection and a web browser.
- The worksheet stylisation of the editor is friendly for students unfamiliar with theorem provers as it emulates a hand-written style. Saving and importing the editor state is hassle-free and allows teachers to create assignments for students to fill in and return. Pretty-printing is employed to make the UI more readable as novice users will not be used to reading statements unstyled (such as for logical operators or quantifiers).
- Proofs can be displayed using *tree* notation, or *prose-style* notation. Both forms of notation emulate how proofs are commonly written on paper. This contrasts with PAs such as Isabelle that use a series of commands to apply individual rules. Also, in Isabelle, the user can only view the resulting sub-goals for a single rule – this makes it difficult to picture the proof as a whole, as can be seen in Figure 1.1. The tree notation comprises of writing the theorem we wish to prove, working from the bottom up applying rules, as seen in Figure 1.2a. The prose style is very similar but splits up each goal into its tasks, as seen in Figure 1.2b.

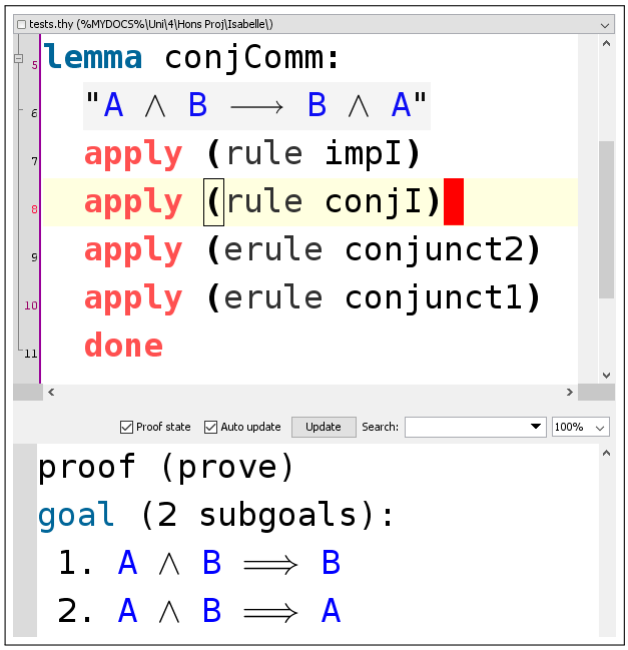
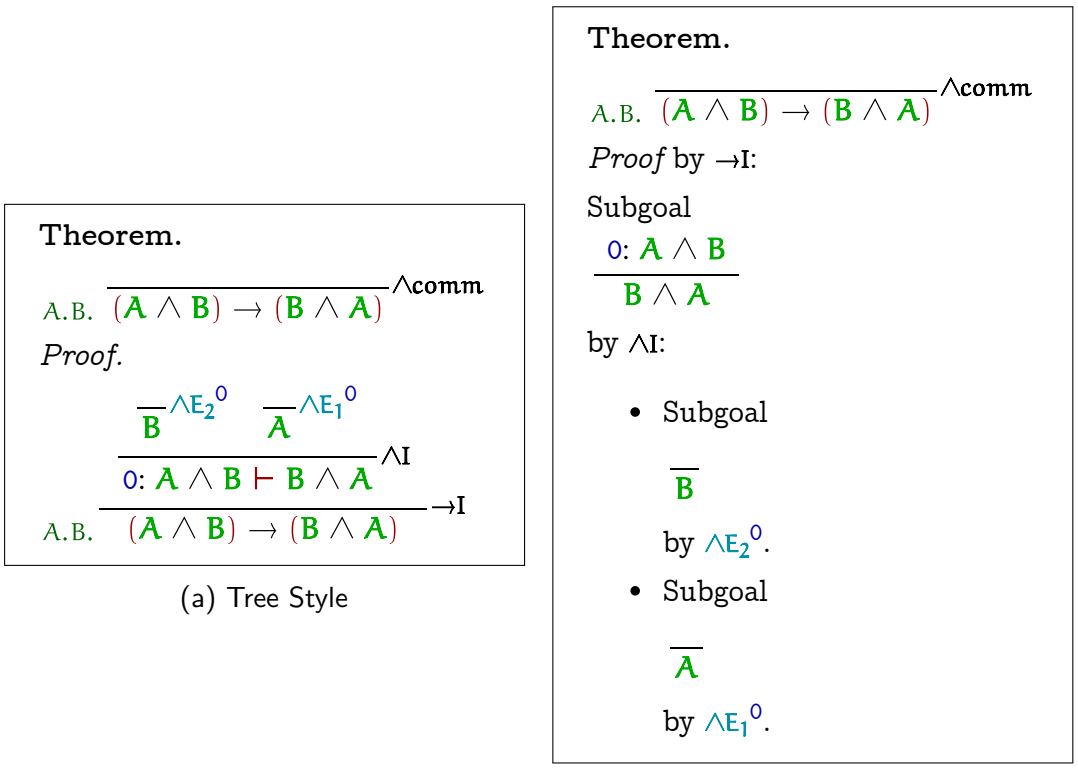


Figure 1.1: Conjunction Commutativity in Isabelle



(a) Tree Style

(b) Prose Style

Figure 1.2: Proof Notations in Holbert

1.2 Contributions

The four main contributions to Holbert that were undertaken in this project were:

1. Making it possible to apply rules by elimination to a goal.
2. Making it possible to complete a proof by induction[1].
3. Making it possible to complete a proof by cases[1].
4. Making it possible to define multiple rules in one section, and also remove them individually.

The ultimate goal for this project was making it possible to write proofs by induction – making it possible to apply rules by elimination was required for this. Proofs by cases was a by-product of the first two contributions. The last contribution was an extension of the primary goals and is the foundation for further work to be carried out on proofs by induction.

[1]Technically, it was possible to write these proofs by manual instantiation of many generated unification variables in Holbert. However, this task would require detailed understandings of the inner workings of the PA and would be beyond almost all intended users.

1.3 Report Structure

The rest of the report is structured as follows:

- Chapter 2, "Background", discusses the foundations of automated reasoning and the functionality of Holbert. We take a closer look into natural deduction and how to construct a proof, analyse, specifically, what a proof by induction is and how proofs are constructed in Holbert.
- Chapter 3, "Implementation", discusses the approach, challenges and implementation for the contributions made to Holbert.
- Chapter 4, "Evaluation", discusses what can now be completed in the current state of Holbert and how this compares to its functionality before the aforementioned contributions.
- Chapter 5, "Conclusion", reflects on the accomplishments of this project, its current limitations, the future work for proofs by induction, and some final remarks.

Chapter 2

Background

2.1 Automated Reasoning Techniques

This section will cover the foundations of automated reasoning and how to formulate a proof by induction in Holbert.

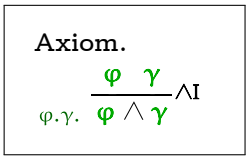
2.1.1 Natural Deduction

In Holbert, proofs are represented as trees of natural deduction derivations [Gentzen, 1935]. Natural deduction expresses logical reasoning with inference rules.

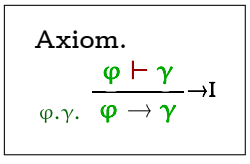
Inference rules can *introduce* or *eliminate* logical judgements, such as the logical connectives: negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\longrightarrow$) and bi-implication ($\longleftrightarrow$); and the quantifiers: universal ($\forall$) and existential ($\exists$). An introduction rule simply introduces a logical connective or quantifier in its conclusion, and, in contrast, an elimination rule describes how to deconstruct a compound proposition. Take for example the following three rules:

$$\underset{\varphi.\gamma.}{\dfrac{\varphi \qquad \gamma}{\varphi \wedge \gamma}}\ \wedge\mathrm{I} \qquad \underset{\varphi.\gamma.}{\dfrac{\varphi \wedge \gamma}{\varphi}}\ \wedge\mathrm{E}_1 \qquad \underset{\varphi.\gamma.}{\dfrac{\varphi \wedge \gamma}{\gamma}}\ \wedge\mathrm{E}_2 \tag{2.1}$$

"Conjunction Introduction" ($\wedge$I) concludes that the formula $\varphi \wedge \gamma$ holds provided φ holds and so too does γ; "Conjunction Elimination 1" ($\wedge\mathrm{E}_1$) concludes that φ holds provided the formula $\varphi \wedge \gamma$ holds; and, on the other hand, "Conjunction Elimination 2" ($\wedge\mathrm{E}_2$) concludes that γ holds provided that the formula $\varphi \wedge \gamma$ holds. Holbert's GUI follows the same notation, where we can define rules in "Axiom". Say we have:



(a) $\wedge$I Rule



(b) $\longrightarrow$ I Rule

Figure 2.1: Introduction Rules in Holbert

We define theorems in a similar fashion and an empty goal will be rendered below the definition:

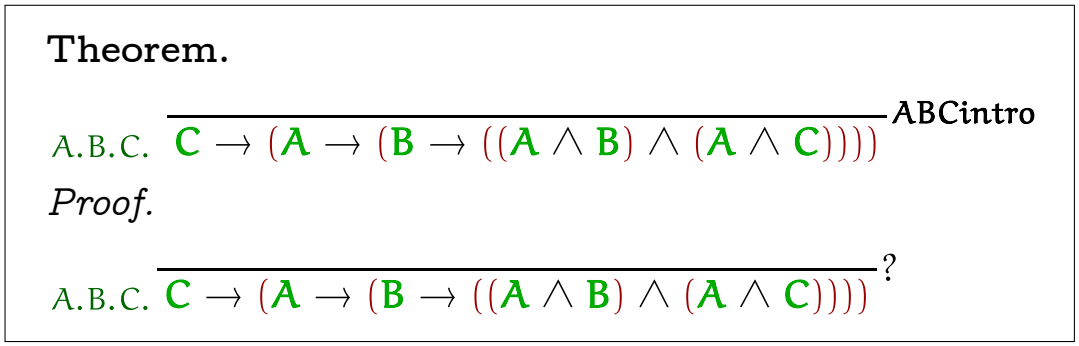


Figure 2.2: Proof Using Introduction Rules in Holbert

To write a proof we want to apply rules to our goal. To apply a rule by introduction, we attempt to *unify* the goal with the conclusion of the rule – this is demonstrated in Figure 2.3. To apply a rule by elimination, we first attempt to unify the assumptions of the goal and the first premise of the rule, then the substituted goal and conclusion – this is discussed in depth in Section 3.2.

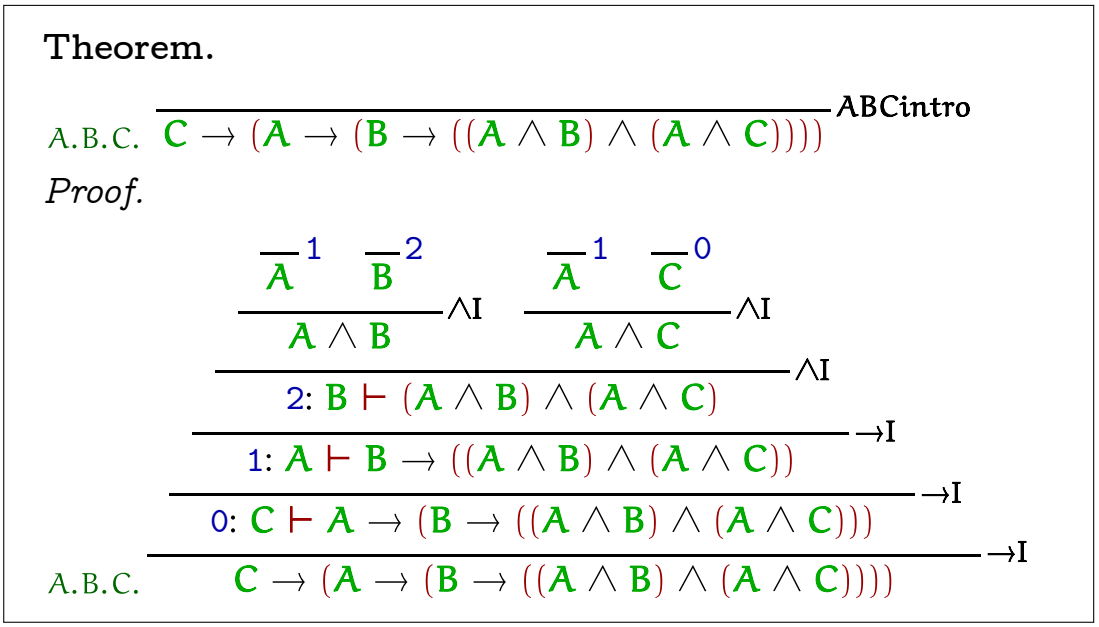


Figure 2.3: Proof Using Introduction Rules in Holbert

The unification problem is finding a substitution θ to metavariables t and u, such that

$$\theta t \equiv_{\alpha\beta} \theta u \tag{2.2}$$

Thus, if we can make t and u equivalent, we can continue our proof. The unification problem is the foundation of automated reasoning. Finding such a substitution θ over λ-terms is undecidable. However, Miller [1991] provides an $O(n)$ algorithm for

a decidable subset (the *pattern subset*) which covers enough cases to be considered suitable for Holbert. In Holbert's case, we use a Haskell [Marlow, 2010] port of the Nipkow unification implementation [Nipkow, 1993]. More on Haskell is discussed in Section 2.2.1.

2.1.2 Formulae Representation in Holbert

A formula in Holbert is represented as a Hereditary Harrop Formula (HHF) [Harrop, 1956]. Isabelle also uses HHF where formulae are represented as the following:

$$\mathsf{H} \equiv \left(\bigwedge \mathsf{x}^*.\ \mathsf{H}^* \Longrightarrow \mathsf{A} \right), \tag{2.3}$$

where $\bigwedge$ is a universal meta-quantifier, x is the bound variable, A is an atomic proposition and $*$ indicates zero or more. This notation is not representative of traditional natural deduction notation. Holbert's pretty-printing renders this as a slight generalisation of the notation of Gentzen [1935] – that reflects the traditional method of writing proofs – and is represented as the following, where H_n is a HHF:

$$\mathsf{x}^*. \ \frac{\mathsf{H}_0 \qquad \mathsf{H}_1 \qquad \ldots}{\mathsf{A}} \ , \tag{2.4}$$

An atomic proposition, A, is a term in untyped λ-calculus [Church, 1941]. Untyped systems lead to logical inconsistency, as noted by Girard [Coquand, 1986], so other theorem provers use typed λ-calculus [Church, 1940] to avoid this problem.

Unlike popular PAs, Holbert is untyped and is therefore unsound – being unsound, Holbert should not be used for verification. If Holbert used a type system, however, using it to aid a course about type systems would prove very difficult. If we wanted to use Holbert to teach types, being knowledgeable in types would first be required to operate Holbert. Since Holbert's focus is to be of assistance in the classroom, especially for those with no experience with theorem provers, it is acceptable to not worry about soundness.

In λ-calculus, terms can be represented using the de Bruijn indices technique [de Bruijn, 1972], Holbert uses, specifically, the "locally nameless" technique, where free variables are still represented with names and only the bound variables with indices [Charguéraud, 2012, McBride and McKinna, 2004]. Bound variables are represented as de Bruijn indices by replacing each variable, right to left, with an integer starting from zero within a node of the syntax tree. Take, for example, the following λ-expression:

$$(\lambda x.\ \lambda y.\ \lambda z.\ (w\ x\ y\ z)\ x)(\lambda z.\ z) \tag{2.5}$$

There are two ways this can be expressed:

$$(\lambda 2.\ \lambda 1.\ \lambda 0.\ (3\ 2\ 1\ 0)\ 1)(\lambda 0.\ 0) \text{ in standard de Bruijn notation.} \tag{2.6}$$

$$(\lambda 2.\ \lambda 1.\ \lambda 0.\ (w\ 2\ 1\ 0)\ 1)(\lambda 0.\ 0) \text{ in "locally nameless" notation.} \tag{2.7}$$

Notice that in Notation 2.6 we reference w with 3 – free variables are represented with unique integers not used when replacing the bound variables. In Notation 2.7, w remains named. In both notations, we see that the variable z is given two different indices since they are in different nodes of the syntax tree.

This technique is extremely useful for α-*equivalence* since it ensures equality between two terms. Take, for example, the following two λ-expressions:

$$\lambda x.\ x, \quad \lambda y.\ y \tag{2.8}$$

These two terms are equal since, after applying de Bruijn indexing, they both resolve to $\lambda 0.\ 0$.

2.1.3 Proofs by Induction

A simple use of proof by induction is in mathematics where we attempt to prove a goal holds for all natural numbers.

$$\frac{}{0\ \mathbf{Nat}}\ \textsc{basis} \qquad \frac{n\ \mathbf{Nat}}{n+1\ \mathbf{Nat}}\ {\scriptstyle n.}\ \textsc{succ} \tag{2.9}$$

Proof by induction first takes a set of introduction rules: our *base case* (or *basis*) and our *inductive steps* – these steps are recursive functions. Suppose that for a statement P that is any property of natural numbers, $P(0)$ is our basis. Our inductive hypothesis is that $P(n)$ holds, where n is a natural number. We then must prove it holds for the *successor function*, $P(n+1)$.

We can summarise these proof requirements in the form of an inductive principle, which is an elimination rule that effectively states that the set **Nat** is the smallest set satisfying the rules in Equation (2.9). Our principle is defined in the language of HOL as follows:

$$\textsc{p.x.}\ \frac{x\ \mathbf{Nat} \qquad P(0) \qquad {\scriptstyle n.}\ \dfrac{\begin{array}{c}P(n)\\ \vdots \\ P(n+1)\end{array}}{}}{P(x)} \tag{2.10}$$

We can also express this in the language of first-order logic (FOL) as follows:

$$P(0) \land \forall n.\ (P(n) \longrightarrow P(n+1)) \longrightarrow \forall n.\ P(n) \tag{2.11}$$

How we use the inductive set and principle to prove a goal for some P, for all n, is discussed in Section 3.3.

2.2 Programming and Web Technologies

Holbert is implemented in Haskell and rendered in-browser with the Miso application framework [Johnson, 2021].

2.2.1 Haskell

Proof assistants and theorem proving tools are traditionally developed in functional languages, or ones that support functional styles, such as Haskell, OCaml [Wright and Felleisen, 1994] (for Coq) or Scala [Odersky, 2006] (for Isabelle). Haskell is especially useful for handling terms in Holbert.

Holbert incorporates lenses [Pickering et al., 2017] to act on specific parts of the editor state (i.e. the axioms, theorems and text elements). The editor state is large, thus it would be very difficult to operate on these individual states without being able to traverse the components. Specifically, the `optics` library [Grenrus, 2021] is used to accomplish this.

We traverse lenses to access the editor state component we wish to modify. Lenses are composed of two functions: the "getter" and "setter" methods. These functions allow us to access the index we have traversed to and either read or write its value.

2.2.2 Miso

Miso is a framework that allows for purely functional GUIs. It employs a similar model to that of Elm [Czaplicki and Chong, 2013], a framework itself loosely based on functional reactive programming [Courtney and Elliott, 2001] – a purely functional way to specify GUIs. Many popular frameworks, such as React [Facebook, 2021], employ this model also.

Chapter 3

Implementation

3.1 The Approach

To write a proof by induction, we must apply the inductive principle by elimination to the assumption of the goal. Previously in Holbert, we could only apply rules by introduction. Thus, the first task was to implement a function that could apply rules by elimination. Once it was possible to do proof by induction, features to improve the user experience could be added. Such additions were making it possible to define multiple rules in one section and defining a new element for induction axioms, specifically.

3.2 Applying a Rule by Elimination

The existing implementation of applying a rule by introduction was used as a foundation to build applying a rule by elimination. Suppose we have a rule in the form:

$$\nu^*.\ \frac{\mathsf{P}_1 \qquad \mathsf{P}_2 \qquad ...}{\mathsf{C}}\ , \tag{3.1}$$

where ν^* are the bound variables of the rule, $\mathsf{P}_1, \mathsf{P}_2,\ ..., \mathsf{P}_n$ are our premises, and C is our conclusion. We also have a goal for a theorem in the form:

$$u^*.\ \overline{\mathsf{A},\ \Gamma \vdash\ \mathsf{G}}\ , \tag{3.2}$$

where u^* are the bound variables of the theorem, A is a particular assumption we wish to eliminate, Γ is all other assumptions, and G is our goal. The steps for applying a rule by introduction are:

1. Instantiate the variables of the rule, ν^*, with fresh unification variables.
2. Unify the goal, G, with our conclusion, C, to find the substitution θ.

3. Return our premises $P_2, \ldots, P_n$ as our new sub-goals, and the substitution, θ, is applied to the whole proof tree.

To apply a rule by elimination we must unify the assumptions first, then unify the substituted goal and conclusion. Thus, the steps required to apply a rule by elimination are:

1. Instantiate ν^*, with fresh unification variables.
2. Unify the assumption of the theorem, A, with our first premise, P_1, to find the substitution θ.
3. Unify the substituted goal, $\theta(G)$, with our substituted conclusion, $\theta(C)$, to find the substitution σ.
4. Return $P_2, \ldots, P_n$ as our new sub-goals, and the composition of the two substitutions, $\theta \circ \sigma$, is applied to the whole proof tree.

Of course, if neither of the unification steps can find a solution, then we cannot apply the rule. When all sub-goals have been unified with a rule, then the proof is complete.

3.2.1 Instantiating the Variables of an Elimination Rule

To instantiate the variables ν^* is to replace them with unification variables. We denote unification variables with a name leading with a question mark, $?x$, just as in Isabelle. Unification variables are global to the proof and therefore cannot be substituted for terms that contain bound local variables. We circumvent this problem by applying all those bound variables u^* to each $?x$. This process is simplified through the use of de Bruijn indexing, as those of u^* are just the integers from zero to the length of u^*. Suppose that we have the following rule:

$$\varphi.\gamma.\ \frac{\varphi \wedge \gamma}{\varphi}\ \wedge E_1 \tag{3.3}$$

We also have the goal for conjunction commutativity that has already had the introduction rule $\wedge I$ applied:

$$\text{Q.R.}\ \frac{\dfrac{}{0 : Q \wedge R \vdash R}\ ? \qquad \dfrac{}{0 : Q \wedge R \vdash Q}\ ?}{0 : Q \wedge R \vdash R \wedge Q}\ \wedge I \tag{3.4}$$

We want to apply the rule $\wedge E_1$ by elimination to the goal $0 : Q \wedge R \vdash Q$ (highlighted in blue). The variables ν^* are $[\gamma, \varphi]$, and the variables u^* are $[R, Q]$. Notice that these lists are reversed so as to obtain their de Bruijn indices easily. Thus, γ and R have an de Bruijn index of 0, and φ and Q have an index of 1. The unification variables are applied like so, where x and y are fresh obtained integers:

$$\frac{(?x\ R\ Q)\ \wedge\ (?y\ R\ Q)}{(?x\ R\ Q)}\ \wedge E_1 \tag{3.5}$$

3.2.2 Finding the Substitution θ

Once instantiation has been completed for each of those in ν^*, we begin unification. We attempt to unify A with a premise P_1, to obtain θ.

In our example, our premise P_1 is $(?x\ R\ Q)\ \wedge\ (?y\ R\ Q)$ and our assumption A is $Q\ \wedge\ R$. Thus, we obtain $\theta = \{(\lambda R.\ \lambda Q.\ Q)/?x,\ (\lambda R.\ \lambda Q.\ R)/?y\}$. After substitution and β-reduction, our rule will look like:

$$\frac{Q\ \wedge\ R}{Q}\ \wedge E_1 \tag{3.6}$$

3.2.3 Finding the Substitution σ

We only proceed to this step if we have successfully found a solution when unifying the assumption with the premise. To obtain σ, we attempt to unify $\theta(G)$ with $\theta(C)$.

In our example, our goal G is Q and our conclusion C is $(?x\ R\ Q)$. Therefore, $\theta(G)$ is still Q and $\theta(C)$ is also Q. Thus, we obtain an empty substitution $\sigma = \{\}$ since our substituted goal is equal to our substituted conclusion.

3.2.4 Composing the Substitutions

If we can obtain a substitution σ, we can compose our substitutions, $\theta \circ \sigma$, and apply this to the proof tree. This composition will update the unification variables throughout the proof tree and render the new sub-goals. If the list of goals is empty, then the proof is complete.

In our example, our proof tree after the composition of our substitutions is:

$$\text{Q.R.}\ \frac{\dfrac{}{0 : Q\ \wedge\ R\ \vdash\ R}\ ?\qquad \dfrac{\dfrac{}{0 : Q\ \wedge\ R\ \vdash\ Q\ \wedge\ R}\ 0}{0 : Q\ \wedge\ R\ \vdash\ Q}\ \wedge E_1}{0 : Q\ \wedge\ R\ \vdash\ R\ \wedge\ Q}\ \wedge I \tag{3.7}$$

In Holbert's notation, when applying a rule by elimination, we superscript the assumption we use on the applied rule. This notation is explained in Section 3.5 where we walk through how to write proofs in Holbert.

$$\text{Q.R.}\ \frac{\dfrac{}{0 : Q\ \wedge\ R\ \vdash\ R}\ ?\qquad \dfrac{}{0 : Q\ \wedge\ R\ \vdash\ Q}\ \wedge E_1{}^{0}}{0 : Q\ \wedge\ R\ \vdash\ R\ \wedge\ Q}\ \wedge I \tag{3.8}$$

3.3 Proof by Induction

Although applying a rule by elimination was successful on a simple proof, when we attempted to apply an inductive principle by elimination it was not; unification would choose a valid, but unhelpful, solution. For example, in Figure 3.1, we want to prove that the successor of an even number is odd. When applying the inductive principle Even-Induct by elimination to the first sub-goal, the chosen solution (S n) Odd is trivial and does not let us continue with our proof. This section will discuss this issue in greater detail and our solution, thus making it possible to write a proof by induction.

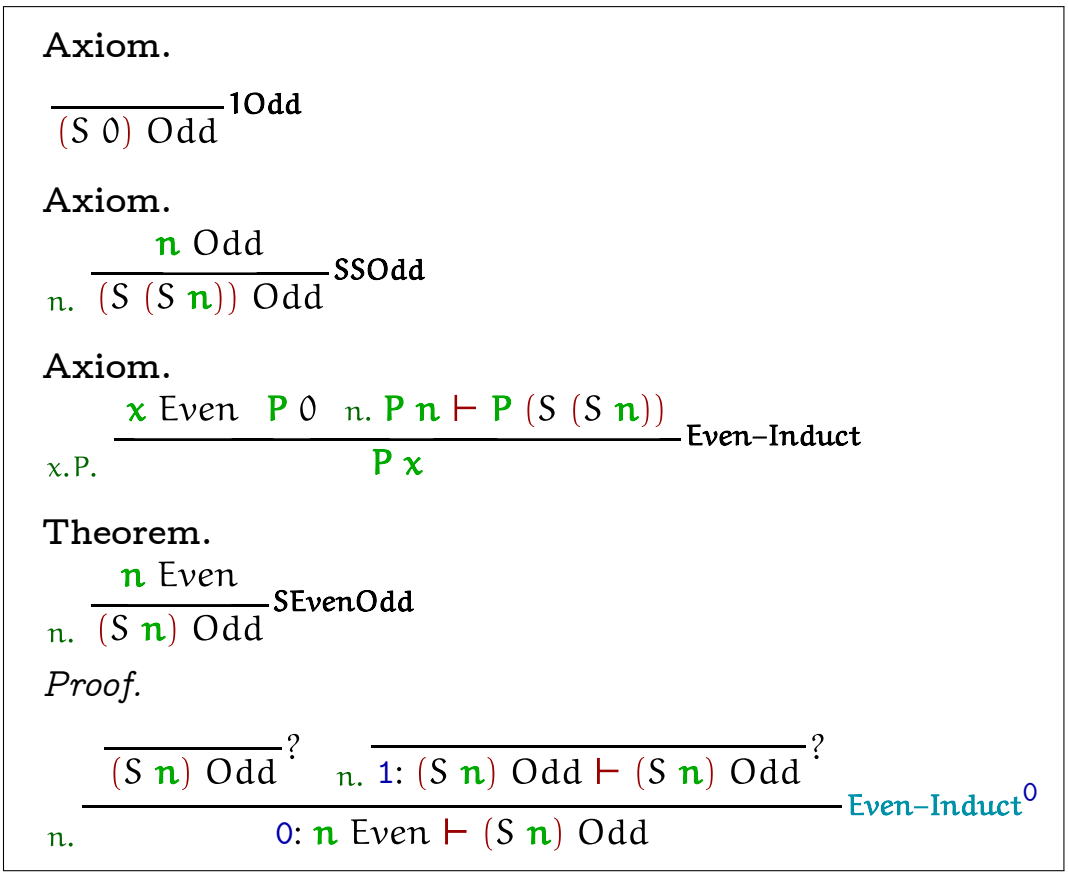


Figure 3.1: Unification Choosing an Unhelpful Solution in Holbert

3.3.1 The Issue with Unification

The issue faced when applying the inductive principle by elimination was that the unification algorithm selected the first possible valid solution rather than the most useful one. The unification function was originally only used in the implementation of applying a rule by introduction and therefore did not encounter this issue.

Suppose we wanted to show that Q x holds for all natural numbers x, where Q is

fixed. We have our inductive principle:

$$\dfrac{\mathfrak{n}\ \mathbf{Nat} \qquad \mathsf{P}\ 0 \qquad \overset{\mathsf{P}\ \mathsf{k}}{\overset{\vdots}{\overline{\mathsf{P}\ (\mathsf{k}+1)}}}\ \mathsf{k}.}{\mathsf{P}\ \mathfrak{n}}\ \mathsf{P}.\mathfrak{n}. \tag{3.9}$$

As mentioned, applying by elimination will instantiate all the variables in the rule, in this case P and $\mathfrak{n}$, with fresh unification variables applied to all variables in the scope of the current goal - in our case here, just $\mathfrak{x}$:

$$\dfrac{(?\mathfrak{b}\ \mathfrak{x})\ \mathbf{Nat} \qquad ?\mathfrak{a}\ \mathfrak{x}\ 0 \qquad \overset{?\mathfrak{a}\ \mathfrak{x}\ \mathsf{k}}{\overset{\vdots}{\overline{?\mathfrak{a}\ \mathfrak{x}\ (\mathsf{k}+1)}}}\ \mathsf{k}.}{?\mathfrak{a}\ \mathfrak{x}\ (?\mathfrak{b}\ \mathfrak{x})} \tag{3.10}$$

Since we are applying our rule by elimination, we first try unifying $(?\mathfrak{b}\ \mathfrak{x})\ \mathbf{Nat}$ with $\mathfrak{x}\ \mathbf{Nat}$. This unification gives us the solution $?\mathfrak{b} = (\lambda\mathfrak{x}.\ \mathfrak{x})$. Applying this substitution to our rule and β-reducing, we get:

$$\dfrac{\mathfrak{x}\ \mathbf{Nat} \qquad ?\mathfrak{a}\ \mathfrak{x}\ 0 \qquad \overset{?\mathfrak{a}\ \mathfrak{x}\ \mathsf{k}}{\overset{\vdots}{\overline{?\mathfrak{a}\ \mathfrak{x}\ (\mathsf{k}+1)}}}\ \mathsf{k}.}{?\mathfrak{a}\ \mathfrak{x}\ \mathfrak{x}} \tag{3.11}$$

The assumption $\mathfrak{x}\ \mathbf{Nat}$ can be disposed of since it is discharged by unification:

$$\dfrac{?\mathfrak{a}\ \mathfrak{x}\ 0 \qquad \overset{?\mathfrak{a}\ \mathfrak{x}\ \mathsf{k}}{\overset{\vdots}{\overline{?\mathfrak{a}\ \mathfrak{x}\ (\mathsf{k}+1)}}}\ \mathsf{k}.}{?\mathfrak{a}\ \mathfrak{x}\ \mathfrak{x}} \tag{3.12}$$

We now try unifying $?\mathfrak{a}\ \mathfrak{x}\ \mathfrak{x}$ with our goal $\mathsf{Q}\ \mathfrak{x}$. This unification problem is *outside the pattern subset*, meaning it does not have a unique solution. Therefore, both $?\mathfrak{a} := (\lambda\mathfrak{a}.\ \lambda\mathfrak{b}.\ \mathsf{Q}\ \mathfrak{a})$ and $?\mathfrak{a} := (\lambda\mathfrak{a}.\ \lambda\mathfrak{b}.\ \mathsf{Q}\ \mathfrak{b})$ would be valid solutions. Choosing the first solution gives us:

$$\dfrac{\mathsf{Q}\ \mathfrak{x} \qquad \overset{\mathsf{Q}\ \mathfrak{x}}{\overset{\vdots}{\overline{\mathsf{Q}\ \mathfrak{x}}}}\ \mathsf{k}.}{\mathsf{Q}\ \mathfrak{x}} \tag{3.13}$$

This rule is trivial and very unhelpful. This solution is what Holbert previously decided to choose, as was seen in Figure 3.1. However, if we apply the second

solution, we get:

$$\dfrac{\mathsf{Q}\ 0 \qquad \overset{\mathsf{Q}\ \mathsf{k}}{\overset{\vdots}{\mathsf{k}.\ \mathsf{Q}\ (\mathsf{k}+1)}}}{\mathsf{Q}\ \mathsf{x}} \tag{3.14}$$

Much better – this is actually what we want! We can now apply an inductive step to show $\mathsf{Q}(\mathfrak{n})$ for some $\mathfrak{n} \in \mathbb{N}$.

3.3.2 The Solution for Fixing Unification

As mentioned previously, the issue with the unification algorithm was that it was choosing the first solution rather than the second. To solve this, we needed to nudge it towards the second solution, i.e. the most helpful solution. The approach taken was to figure out a way to identify such cases and **not apply all variables in scope** to the unification variable – $?\mathfrak{a}$ in the case of the Formulation 3.10. If we did not apply x to $?\mathfrak{a}$ at the beginning but still applied it to $?\mathfrak{b}$, then we would have:

$$\dfrac{(?\mathfrak{b}\ \mathsf{x})\ \mathbf{Nat} \qquad ?\mathfrak{a}\ 0 \qquad \overset{?\mathfrak{a}\ \mathsf{k}}{\overset{\vdots}{\mathsf{k}.\ ?\mathfrak{a}\ (\mathsf{k}+1)}}}{?\mathfrak{a}\ ?1\ \mathsf{x}} \tag{3.15}$$

After applying the assumption unification, we have:

$$\dfrac{\mathsf{x}\ \mathbf{Nat} \qquad ?\mathfrak{a}\ 0 \qquad \overset{?\mathfrak{a}\ \mathsf{k}}{\overset{\vdots}{\mathsf{k}.\ ?0\ (\mathsf{k}+1)}}}{?\mathfrak{a}\ \mathsf{x}} \tag{3.16}$$

From this, we arrive at the unification problem $?\mathfrak{a}\ \mathsf{x} \sim (\lambda \mathsf{x}.\ \mathsf{Q}\ \mathsf{x})$. This problem is easily solved with the substitution $?\mathfrak{a}\ \mathsf{x} := \mathsf{Q}$ by pattern unification.

To accomplish this, our implementation had to be modified to, first, only instantiate the variables mentioned in the first premise of the rule, P_1, present in ν^*, and unify with A before instantiating the rest of the variables in ν^*. We can now unify, and obtain θ, without having to instantiate all the variables of the rule. This means we can use θ to guide how to instantiate the remaining variables in ν^*. The modified steps that we listed in Section 3.2 for applying a rule by elimination are now:

1. Only instantiate those of ν^* that are mentioned in P_1 with fresh unification variables that are applied to all variables in scope of the goal $\mathfrak{u}^*$.
2. Unify A with P_1 to find θ.

3. Instantiate the remaining variables in ν^* with fresh unification variables that are applied to certain variables in scope of the goal $\mathfrak{u}^*$, guided by θ. Details on this are given below.
4. Unify $\theta(G)$ with $\theta(C)$ to find σ.
5. Return $P_2, \ldots, P_n$ as our new sub-goals, and $\theta \circ \sigma$ is applied to the whole proof tree.

Again, if unification cannot find a solution, then the rule cannot be applied.

3.3.3 Instantiating Variables Only Mentioned in a Premise

To instantiate only the mentioned variables, we must first obtain the de Bruijn indices for the variables of ν^*. We then filter out the variables that do not occur in the premise P_1. We then instantiate on this new list. On completion, just as before, we attempt to obtain θ.

3.3.4 Instantiating The Remaining Variables

Once θ has been obtained, we instantiate with the variables we excluded from ν^* with fresh unification variables. Unlike previously, however, these unification variables are not applied to all of $\mathfrak{u}^*$, but only those variables that do not occur in the in the substituted P_1, $\theta(P_1)$. Note that this would produce the exact desired proof tree, Equation (3.15), we sketched above. We can now attempt to obtain σ, and if successful apply $\theta \circ \sigma$ to the proof tree, just as before.

3.4 Multiple Axioms GUI Element

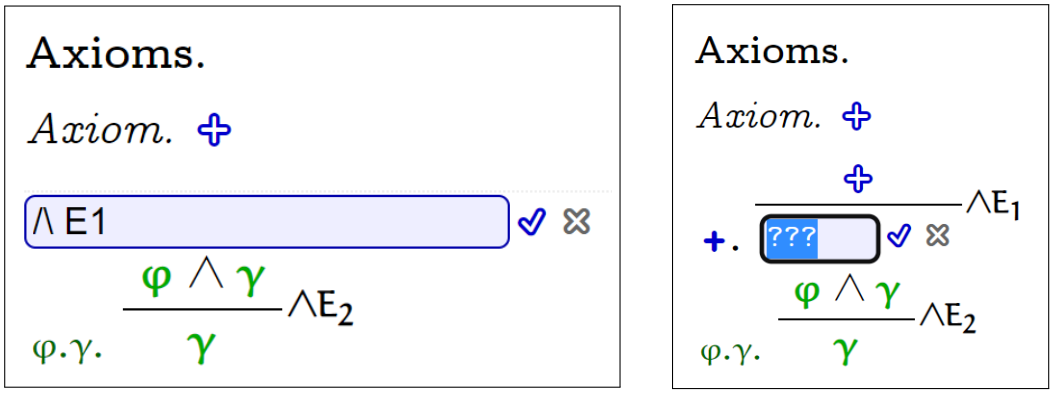
After successfully implementing proof by induction, we moved onto adding functionality that allows users to define multiple rules in a single "Axiom" element. We wanted this functionality so the user could define the inductive set in a streamlined fashion. This list of rules will be used as the foundation for further work on automating the generation of the inductive principle – this is discussed further in Section 5.2.1.

3.4.1 Adding a New Rule in Same Element

Creating the functionality of allowing the user to add multiple rules in one element involved modifying the behaviour of adding an axiom from the ground up. Originally, when an axiom element was created, the user was prompted to enter a name for the rule. After the name was submitted, a single rule was rendered in the axiom element.

For an axiom element to contain multiple rules, we had to initialise the element with a list containing one rule, to which we could later append rules to the front. A 'plus' icon button was added that indicates that clicking the button appends a

new rule inside the element – there is also a tool-tip that prompts the user with what action the button carries out.



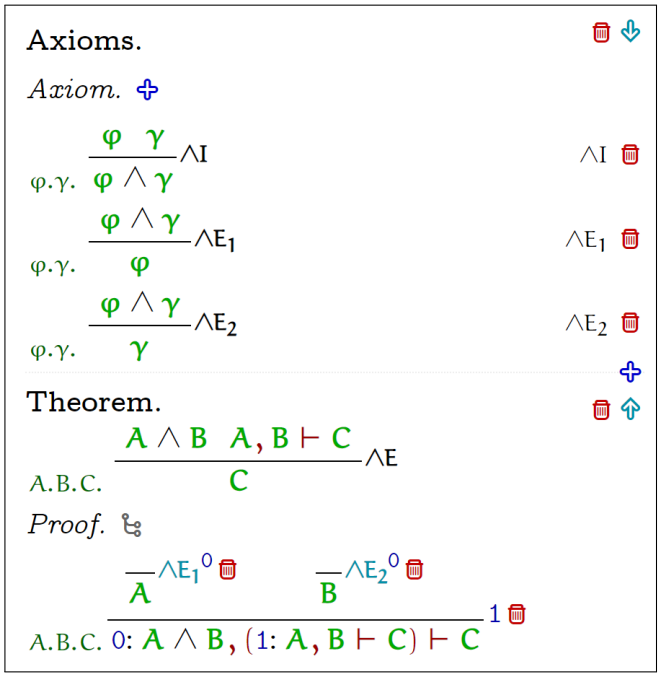
(a) Adding a New Rule

(b) New Rule Added

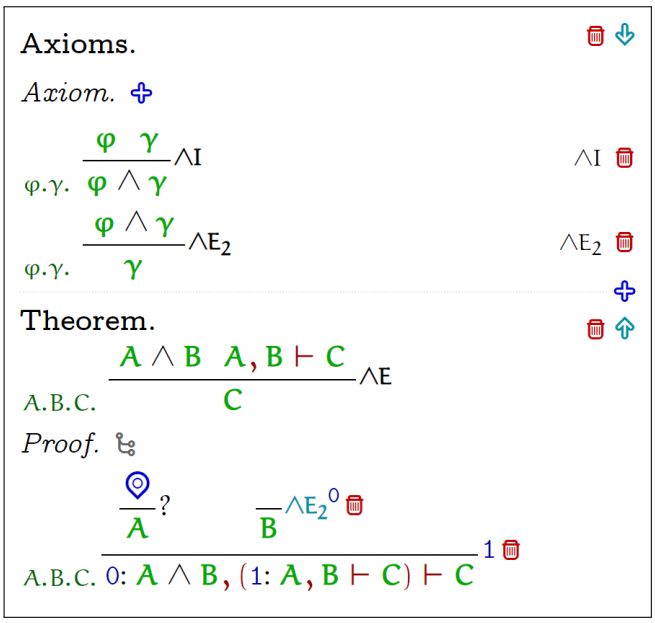
Figure 3.2: Adding Multiple Rules in an Element in Holbert

3.4.2 Removing a Rule From an Element With Multiple Rules

Two actions occur when a rule is removed from the editor: the rule is invalidated and it is graphically removed. When the user hits the 'bin' icon button, we isolate this rule in the list and perform these two actions. To invalidate the rule, we reset all proofs in the document to a state before said rule has been applied. We also make the rule name valid to be used again, as no more than one rule can have the same name. Once the rule is invalidated, we return a concatenation of all the list items before and after said rule – this will update the GUI with the new list of rules.



(a) Proof With Rules in a Multi-Axiom Element



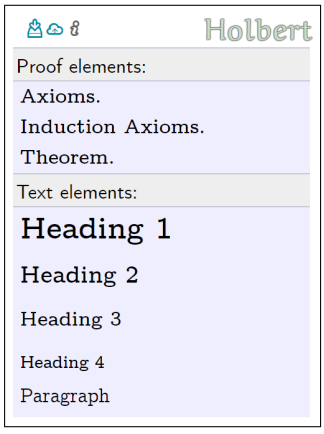
(b) $\wedge E_1$ Removed and Proof Updated

Figure 3.3: Removing a Rule from a Multi-Axiom Element in Holbert

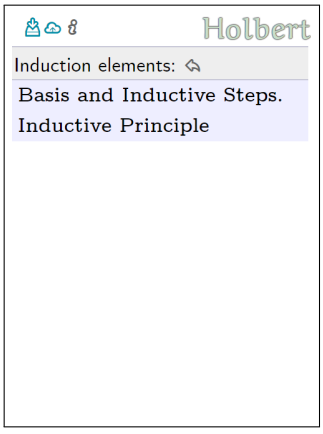
3.4.3 Induction Axioms Elements

Since Holbert is designed to cater for those with no experience using proof assistants, an element that clearly labels that it is used for induction will help users easily navigate the document.

There are two induction axiom elements, one for defining the basis and inductive step(s) and another for defining the inductive principle. Both of these elements have the option to define multiple rules in the element. The side menu offers an 'Induction Axioms' item that displays a sub-menu containing these two options, and a 'back arrow' icon button to return to the parent menu.

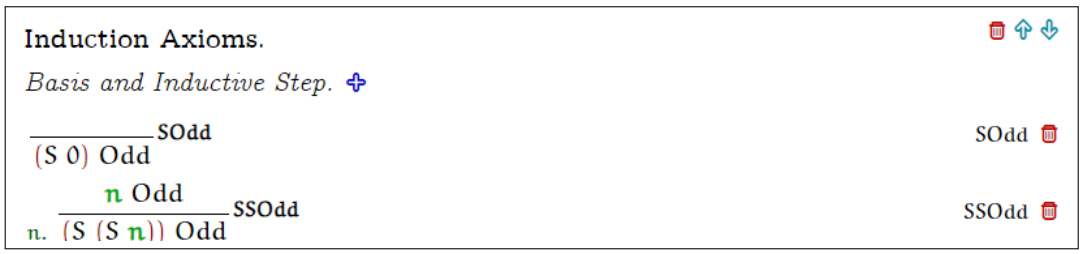


(a) Elements Menu

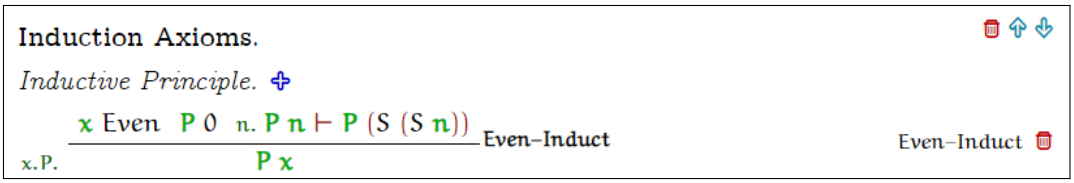


(b) Inductions Axiom Sub-Element

Figure 3.4: Elements Side-Menu in Holbert



(a) Basis and Inductive Steps



(b) Inductive Principle

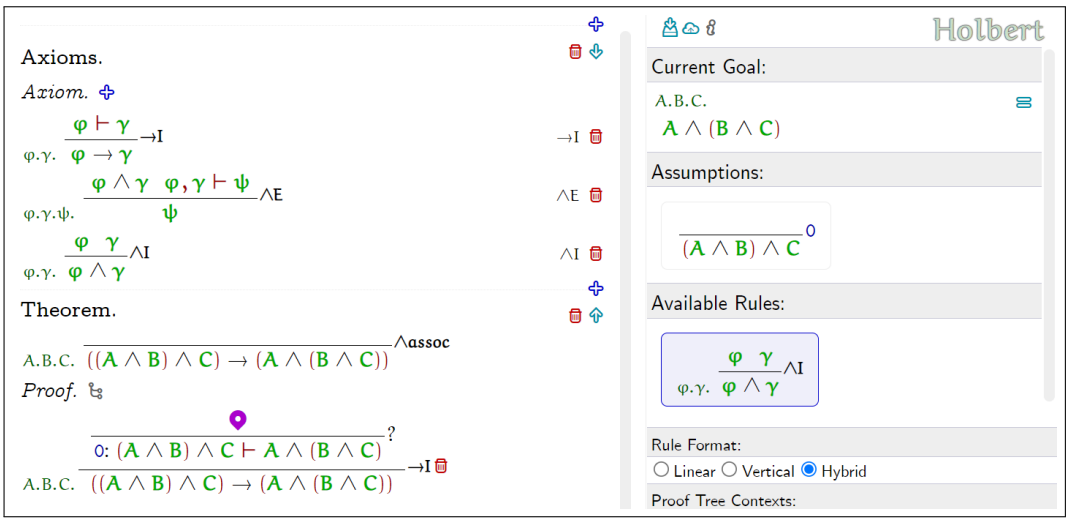
Figure 3.5: Induction Axioms Element in Holbert

3.5 Writing Proofs in Holbert

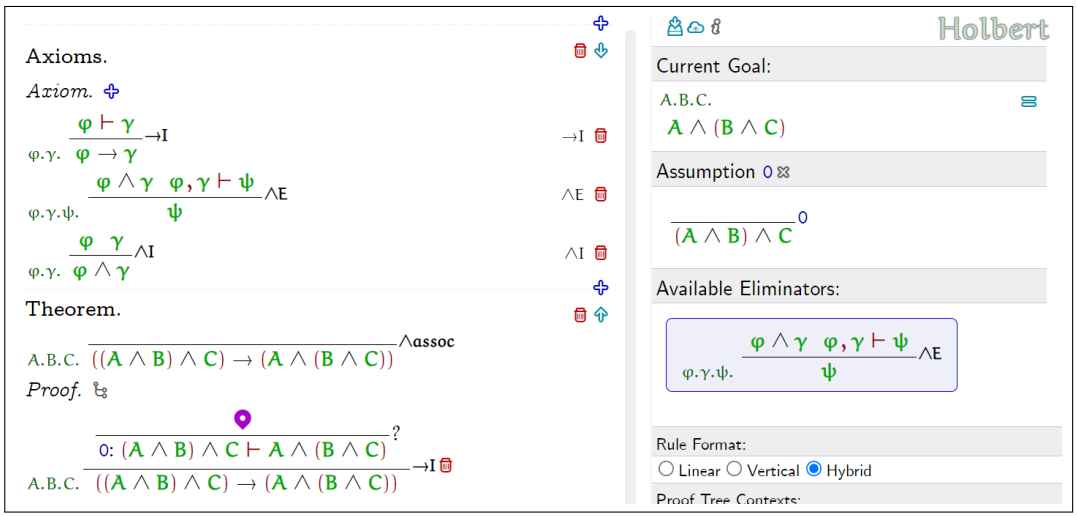
When the user selects the 'pin' icon on the path of the proof tree, we attempt to apply every rule in the document by introduction. In the side menu the user can choose to select an assumption (for applying by elimination) or a rule that can be applied successfully (i.e. unification found a solution). If an assumption is selected, we attempt to apply every rule in the document by elimination. Again, only rules that can be successfully applied are displayed for use. The process of first choosing the assumption to eliminate required a new, unconventional, piece of notation. Say we use assumption zero (0) to apply the rule $\wedge E_1$, then we superscript the assumption like so: $\wedge E_1{}^0$. This rule is also coloured turquoise in the proof tree.

3.5.1 Applying a Rule by Elimination

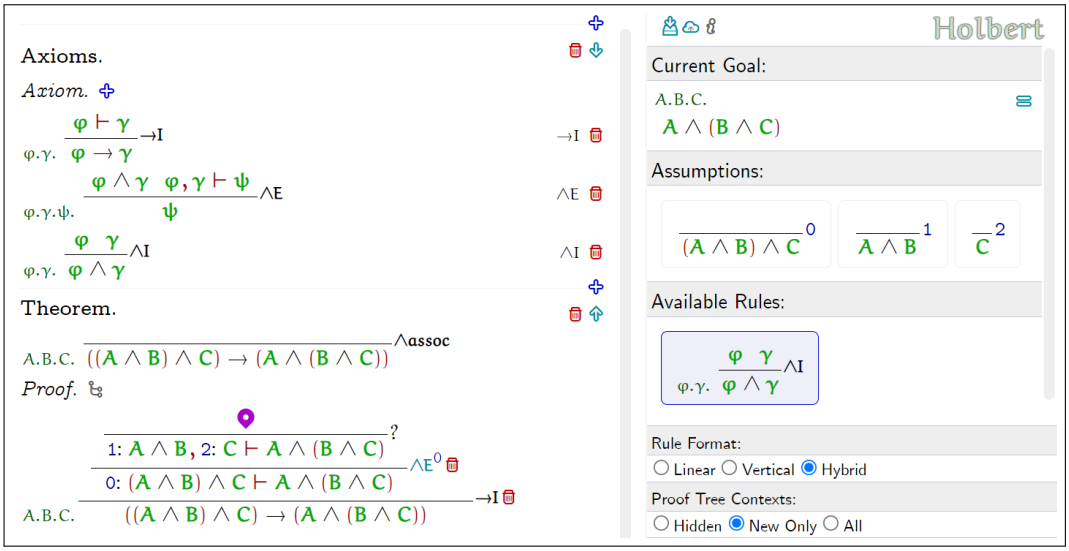
Implementing the steps outlined in Section 3.2 allows us to successfully apply rules by elimination in simple proofs. In Figure 3.6 we see this in action in Holbert:



(a) Path Selected Using the 'Pin'



(b) Assumption 0 Selected



(c) Rule Applied by Elimination

Figure 3.6: Applied $\wedge E_1$ By Elimination in Holbert

3.5.2 Writing a Proof by Induction

The steps outlined in Section 3.3.2 successfully chooses the helpful solution, and thus a proof by induction can be completed. Figure 3.7 is the same theorem as we saw in Figure 3.1 with the proof by induction completed using the inductive set we defined. Our composition of substitutions, $\theta \circ \sigma$, gives us the first sub-goal of (S 0) Odd, and second of (S (S (S n))). For the first sub-goal, we can now apply our inductive step SOdd. For the second sub-goal, we have obtained a new assumption, (S n) Odd, and we can now apply our inductive step SSOdd and said assumption to complete the proof.

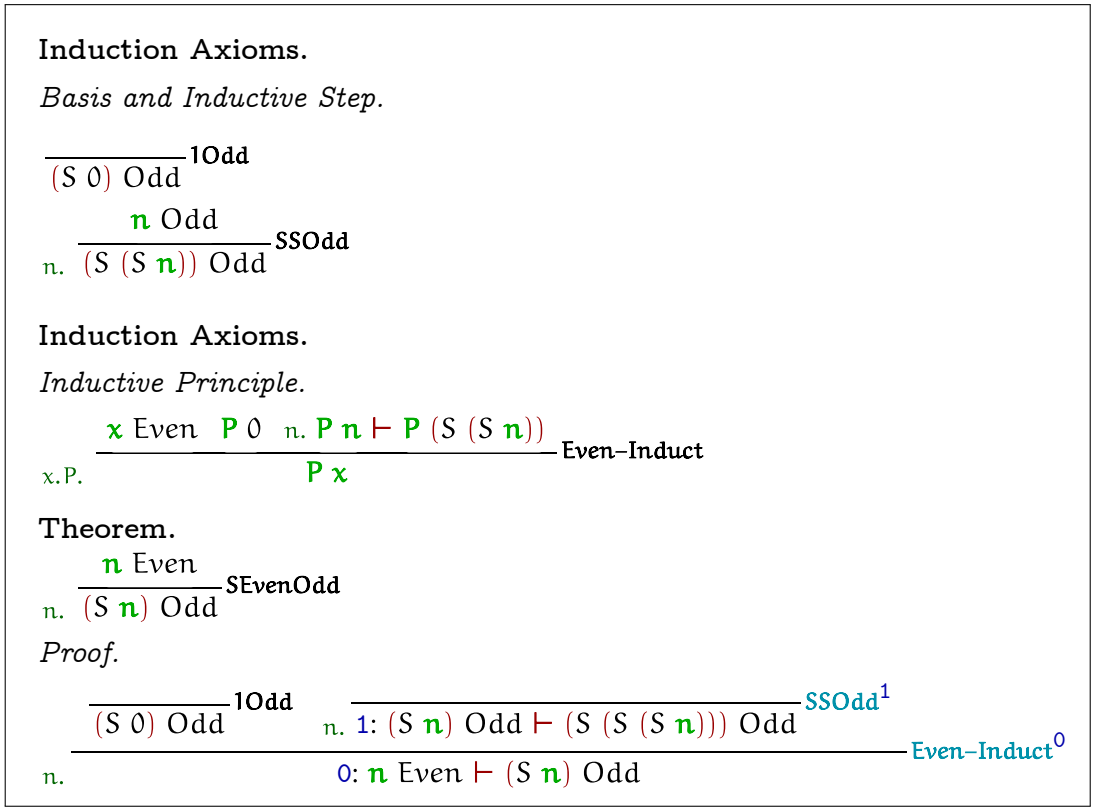


Figure 3.7: Successful Proof By Induction in Holbert

Chapter 4

Evaluation

In this chapter, we will evaluate the functionality of Holbert. Previously, Holbert was only useful for proving basic logical deductions; since the implementation of applying rules by elimination, users are now capable of writing proofs by induction and by cases.

4.1 Applying Rules by Elimination

Before we look at applying rules by elimination in action, let's first define the rules we'll be using. Our introduction rules are:

$$\underset{\varphi.\gamma.}{\frac{\varphi \qquad \gamma}{\varphi \wedge \gamma}}\ \wedge\mathrm{I} \qquad \underset{\varphi.\gamma.}{\frac{\varphi \vdash \gamma}{\varphi \longrightarrow \gamma}}\ \longrightarrow\mathrm{I} \qquad \underset{\varphi.}{\frac{\varphi \vdash \bot}{\neg\varphi}}\ \neg\mathrm{I} \tag{4.1}$$

Our elimination rules are:

$$\underset{\varphi.\gamma.}{\frac{\varphi \wedge \gamma}{\varphi}}\ \wedge\mathsf{E}_1 \qquad \underset{\varphi.\gamma.}{\frac{\varphi \wedge \gamma}{\gamma}}\ \wedge\mathsf{E}_2 \qquad \underset{\varphi.\gamma.}{\frac{\neg\varphi \qquad \varphi}{\gamma}}\ \neg\mathsf{E}$$

$$\underset{\varphi.\gamma.\psi.}{\frac{\varphi \vee \gamma \qquad \varphi \vdash \psi \qquad \gamma \vdash \psi}{\psi}}\ \vee\mathsf{E}$$

$$\underset{\varphi.\gamma.}{\frac{\exists u.\ \varphi(u) \qquad \nu.\ (\varphi(\nu) \vdash \gamma)}{\gamma}}\ \exists\mathsf{E} \qquad \underset{\varphi.\gamma.}{\frac{\exists u.\ \varphi(u) \qquad (\nu.\ \varphi(\nu)) \vdash \gamma}{\gamma}}\ \forall\mathsf{E} \tag{4.2}$$

Figure 4.1 is a simple proof for a unified conjunction elimination rule. In this proof, we apply the rules $\wedge E_1$ and $\wedge E_2$ using the assumption $A \wedge B$.

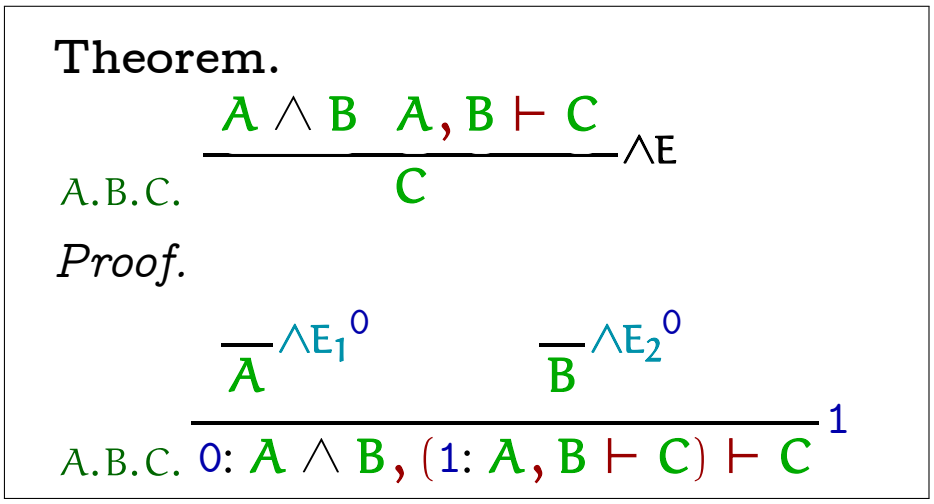


Figure 4.1: Conjunction Elimination 1 and 2 in Holbert

In Figure 4.2 we apply the rule $\wedge E$ using the assumption $(S \vee R) \wedge (\neg S)$, $\vee E$ using the assumption $(S \vee R)$ and $\neg E$ using the assumption $\neg S$.

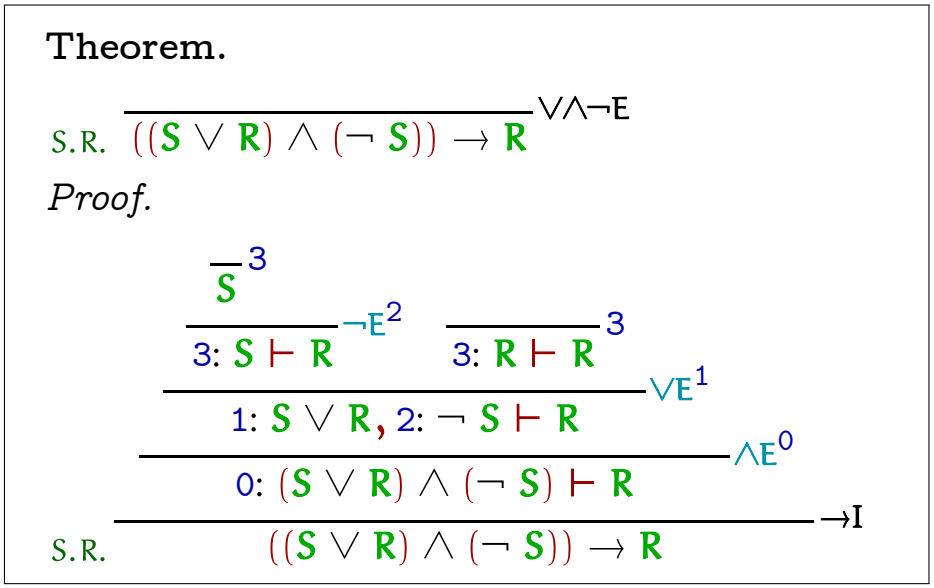


Figure 4.2: Conjunction, Disjunction and Negated Elimination in Holbert

In Figure 4.3 we see the quantifier rules being employed. The rule ∃E is applied using the assumption that ∃x. ¬P(x), and ∀E is applied using the assumption that ∀x. P(x).

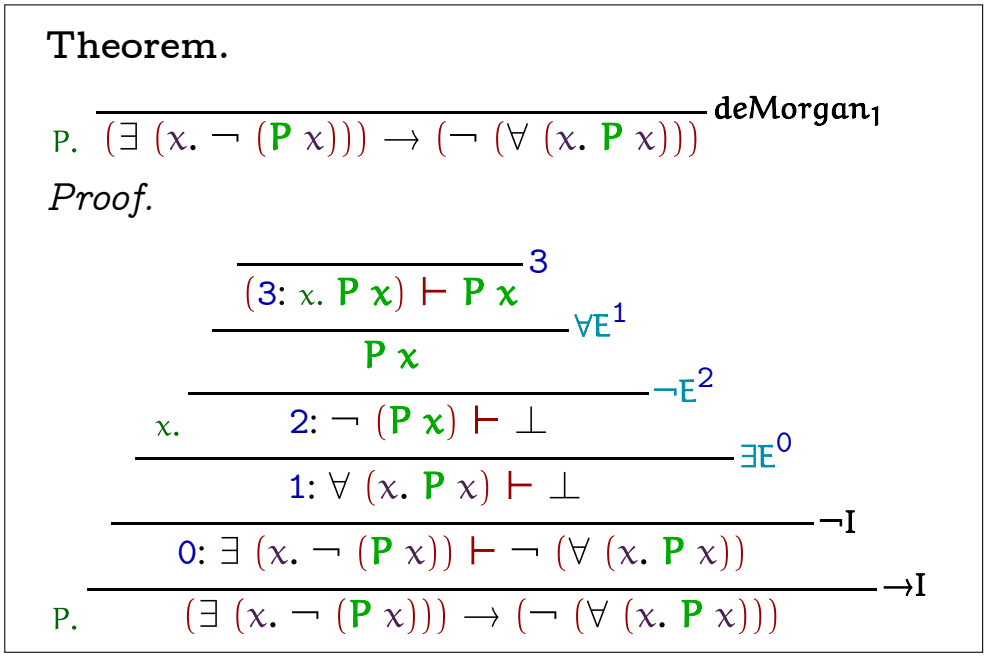


Figure 4.3: Quantifier Elimination in Holbert

In our final example, we see that we can also apply introduction rules by elimination. Although possible, this functionality is probably not all that useful. We see in Figure 4.4 that we first select the assumption A and then apply ∧I by elimination, and then select the assumption B ⊢ A and apply ∧I by elimination again.

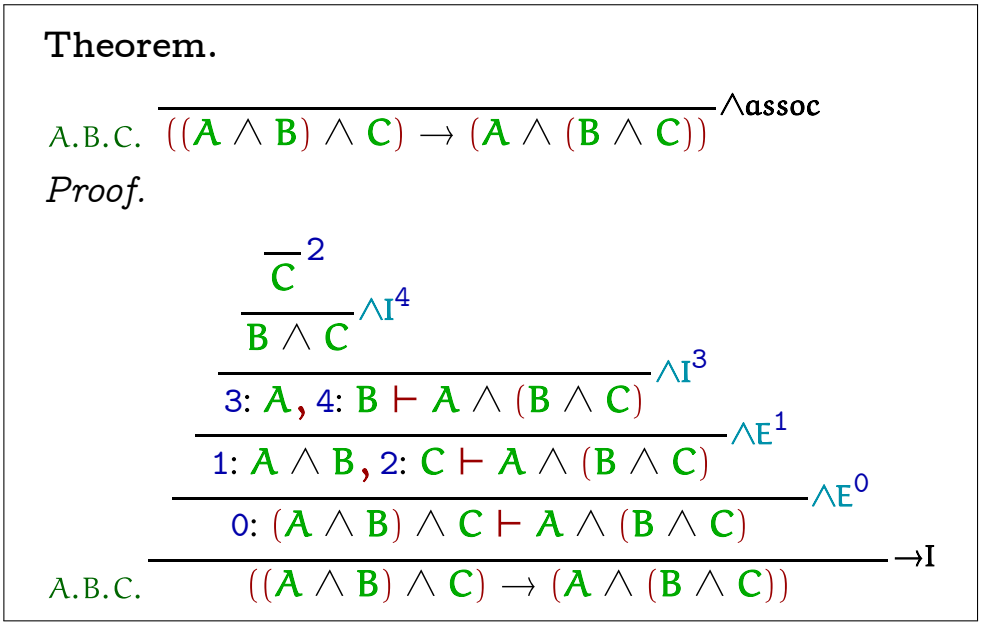


Figure 4.4: Introduction Rules Applied By Elimination

4.2 Writing Proofs by Induction

Now that users can apply rules by elimination, and after fixing unification, they can write a proof by induction. First the user must define their basis, inductive steps and inductive principle. Again, for the target audience of Holbert, manually writing the inductive principle is a challenging step. Automating the generation of this rule is a consideration for future work and is discussed in greater detail in Section 5.2.

In Figure 4.5 we have a proof for the successor of an odd number, $\mathtt{n}$, is an even number. We showed a similar proof in Figure 3.7 but the other way around.

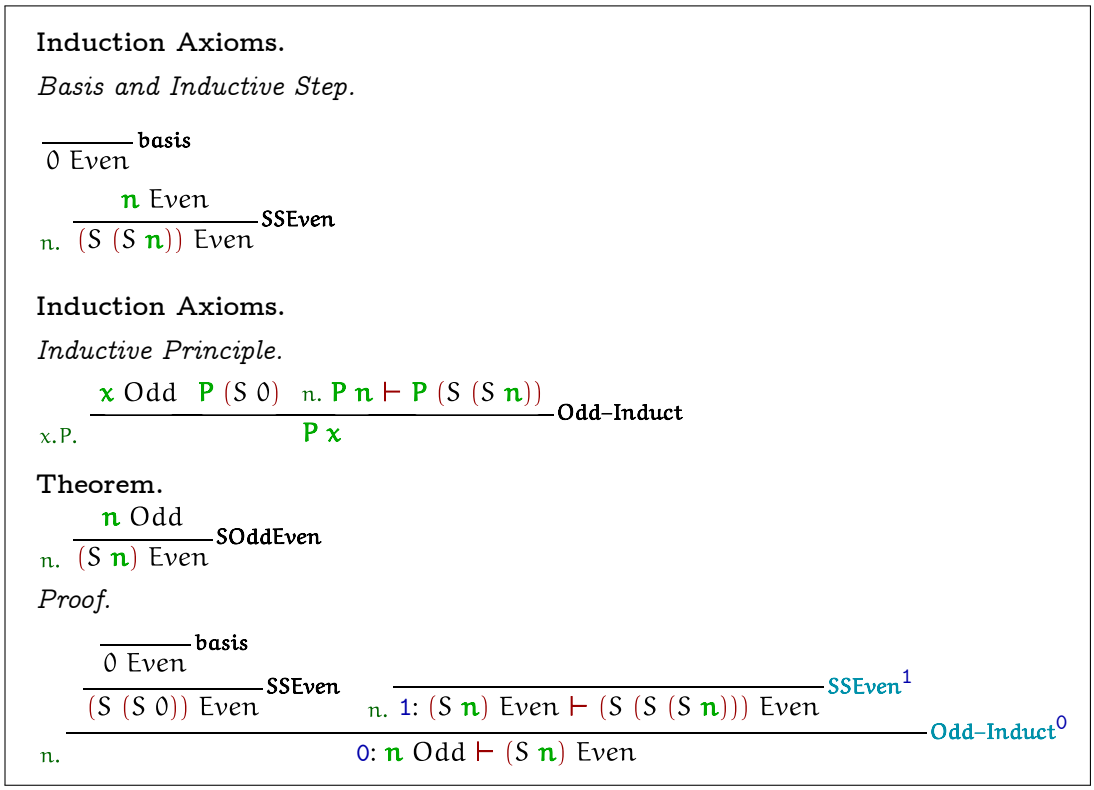


Figure 4.5: Proof By Induction in Holbert – Successor of Odd is Even

4.2.1 Simultaneous Proofs by Induction

We can also perform *simultaneous proofs by induction*. Simultaneous proofs by induction are where we have mutually inductive definitions for odd and even numbers. Take for example Figure 4.6, where we are proving that an odd or even number, $\mathtt{n}$ is a natural number. Notice that our "Odd and Even Axioms", SOdd and SEven, define $\mathtt{n}$ in terms of the successor of each other. Our induction principles now have a slightly different form to handle this behaviour; our premises use P and Q to handle the odd to even direction and vice versa.

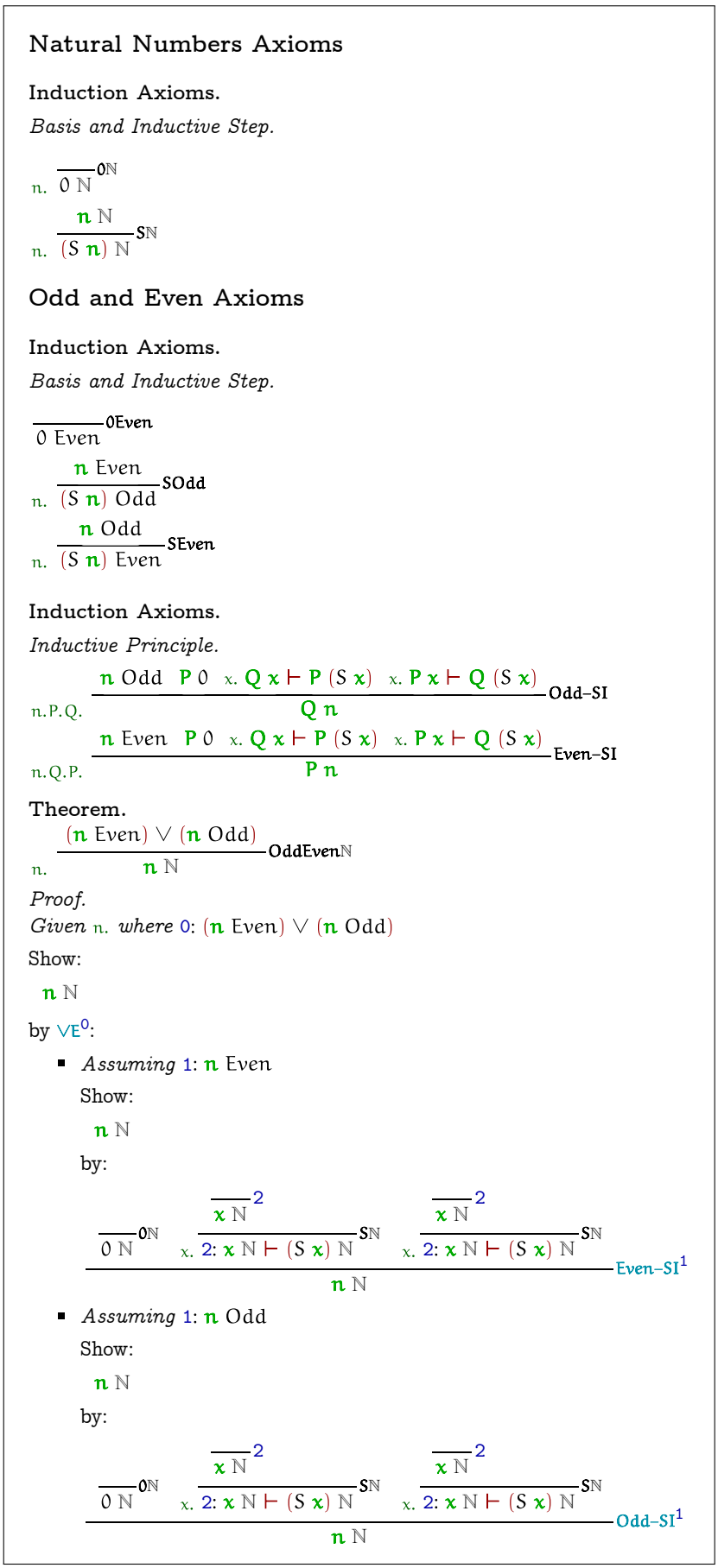


Figure 4.6: Simultaneous Proofs By Induction in Holbert

4.3 Writing Proofs by Cases

As a by-product of implementing applying rules by elimination, and fixing unification, it is also possible to write proofs by cases. Figure 4.7 is the proof for eliminating double negation. This proof also uses equality and rewriting, implemented by Amjad [2022]. We set up introduction rules to define booleans, with the elimination rule b-Case. We also define negation using equalities. We then apply b-Case by elimination and rewrite the goals using the negation definition (rewriting is coloured purple in the proof tree):

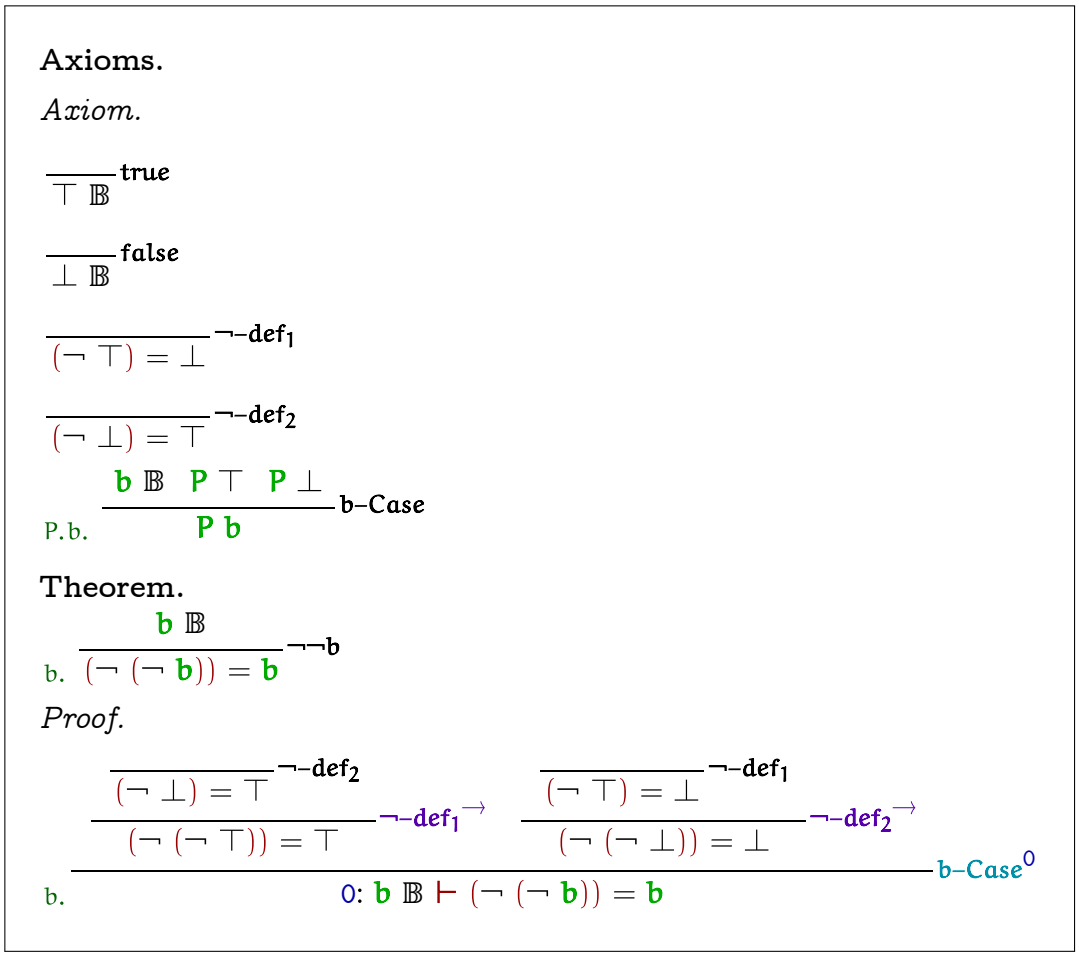


Figure 4.7: Proof by Cases - Double Negation Elimination

4.4 Heuristic Evaluation

A heuristic evaluation was also conducted to ensure that the process of writing a proof by induction was usable. A heuristic evaluation consists of a small group of evaluators that will examine a system and then determine if it meets the standards of a list of recognised usability principles. Nielsen [1994] curated a list of usability principles that are commonly used for UI heuristic evaluation. However, due to time constraints, the evaluation was only conducted by one person, the author, which is a limitation to consider and is discussed further in Section 5.2.

Provided below is an evaluation of using Holbert to write a proof by induction following the aforementioned usability principles:

1. **Visibility of system status:** This principle states that the UI should always provides suitable feedback to the user on the system's status. Generally, this principle is not violated as the user is always shown the current goals of the proof. When a 'pin' icon is selected, the side menu displays the current goal, all available assumptions and available rules that can be applied. When the path of the proof tree is complete, the 'pin' icon is removed. However, when no rules can be applied, the side menu is empty. The user could interpret this as that it is still loading or that there are no rules that can be applied – a message to signal this status could be added.

2. **Match between system and the real world:** This principle states that the UI should feel familiar with what the user already knows. This principle is not violated as the proof trees are rendered similarly to that of writing them on paper. Holbert's prettyprinting ensures the readability of statements, and similar options to control the editor are grouped in the side menu.

3. **User control and freedom:** This principle states that the user should have the freedom to take a step back if they do not wish to proceed with the chosen action. This principle is generally not violated since there are many ways to undo an action. When the user applies the wrong assumption, they can undo this action and re-view the available assumptions and rules, or if they have applied the wrong rule they can hit the 'bin' icon button to delete it. Also, if the user no longer wishes to insert an induction axiom element, there is a back button to take them back to all the menu of all elements that can be inserted. An action that could be implemented, however, is the ability to undo deleting an axiom. Since deleting an axiom will reset any proofs that use it, if the user deletes the wrong axiom they will have to rewrite it and the proof.

4. **Consistency and standards:** This principle states that users should not have to guess what certain actions do or if multiple actions do the same thing. Currently, this rule could be considered violated since there is no difference between the "Axioms" element and the "Induction Axioms" element – both can be used in a proof by induction. However, for the context of this paper, this is satisfactory but note that this can be improved upon, as detailed in

Section 5.2. However, generally, it is clear what actions do and how they are distinguished from each other when attempting to write a proof.

5. **Error prevention:** This principle states that the user should not be allowed to perform erroneous actions. In general, this principle is not violated since only the rules that can be applied will be displayed in the side menu.

6. **Recognition rather than recall:** This principle states that the user should not have to remember information across different tasks. This principle is not violated since the proof tree displays all paths of the sub-goals, and the user can use the side menu to jump to definitions of theorems or axioms present on the page.

7. **Flexibility and efficiency of use:** This principle states that the UI should offer advanced functionality that expert users can access, hidden from novice users. This principle is not violated since, at the bottom of the side menu, there are radio buttons and check-boxes to show and hide extra information inside the proof tree. These options are comparatively smaller than the other actions in the menu, and thus are not as obvious to novice users.

8. **Aesthetic and minimalist design:** This principle states that the UI should be concise and not display more information than necessary. This principle is not violated since only the assumptions and rules applicable to the selected path of the proof tree are displayed in the side menu.

9. **Help users recognise, diagnose, and recover from errors:** This principle states that users should receive meaningful error messages. Since only the rules that can unify with the goal are displayed, an error cannot occur. Thus, we cannot say we have or have not violated this principle.

10. **Help and documentation:** This principle states that the user should be provided with information in how to use the system and troubleshoot errors. Generally, this rule is not met. However, worksheets can be created to explain the functionality of writing a proof by induction in Holbert. For example, the default worksheet loaded by the website could contain such information.

Chapter 5

Conclusion

5.1 Accomplishments

The ultimate goal of this project was to make it possible to write proofs by induction in the Holbert proof assistant. In doing so, the accomplishments of this project were:

1. Making it possible to apply rules by elimination to a goal.
2. A new unification method.
3. Making it possible to write both single and simultaneous proofs by induction[1].
4. Making it possible to write proofs by cases[1].
5. A modified axiom element that supports defining multiple rules and being able to delete them individually.

This report also includes extensive research on the current state of proof assistants in the classroom and why Holbert is a beneficial alternative to popular tools, such as Haskell and Coq.

5.2 Current Limitations and Future Work

The primary limitation of this project is that the user must still manually write their inductive principle. This process will ultimately be a challenging step for the users of Holbert since the target audience is those who have little to no experience with theorem provers. Automating the generation of the inductive principle would likely be the first area for further work.

Another area for further work would be conducting a heuristic evaluation as a team and a usability test study. Since the scope of this project was to make it

[1]Although it was technically possible to write these proofs before, it is no longer an onerous task since substitution of unification variables is performed.

possible to write proofs by induction, carrying out studies to improve the HCI (Human-Computer Interaction) was not a goal. However, we believe this would be essential work in the future since Holbert's ultimate goal is to be used in the classroom; we need to cater for the target audience.

5.2.1 Automating the Generation of the Inductive Principle

Implementing the functionality for defining multiple rules in a single axiom element, as described in Section 3.4, was the first step in the process of generating the inductive principle. The idea is that the user can insert an "Inductive Principle" element and then select a "Basis and Inductive Steps" element from which to generate an elimination rule [Dybjer, 1994, Paulson, 2000].

5.2.2 Heuristic Evaluation

Due to time constraints, it was not possible to gather a team to conduct a heuristic evaluation. It is recommended that a team of five or six evaluators is optimal for catching at least 75% of principle violations [Nielson, 1994]. Although conducting the evaluation as one person was not optimal, it sufficed for this project. We were critical of the completed work and had the oppertunity to justify our design choices.

5.2.3 Usability Testing

Usability testing should be conducted on the process of writing a proof by induction. This study will ensure that further work done to proof by induction captures the needs of the target users, thus making the transition to using it in the classroom successful. A mixed-method lab study would be appropriate, where several HCI methods are combined to gather a greater understanding of the topic [Preece et al., 2019]. The benefit of using a mixed-method study is that what one method, as a stand-alone, may miss, another will catch. These studies do, however, take much longer to plan and run. Such a study must define what makes the system "usable". A suitable definition for proofs by inducting being usable would be that the user can confidently set up the induction axioms and write a proof by induction using them.

Although the target users for Holbert are those with little or no experience in theorem provers, to test the functionality of the proof by induction, study participants would likely need a background in mathematics proofs or be familiar with theorem provers. Participants would be asked to interact with Holbert under the observation of someone with experience of teaching, or tutoring, classes related to the foundation of programming languages or automated reasoning. Holbert would be set up with a worksheet that introduces how to write axioms and theorems and some sample problems for them to solve. After a set amount of time using Holbert, they would be asked to fill out a survey on their experience using it; questions asked in the survey would aim to gather information on how easy the user felt it was to accomplish certain tasks and if anything in the system

felt unclear on how to operate. The results from these studies would highlight complications that may arise using it in the classroom.

5.3 Final Remarks

Holbert can now successfully apply rules by elimination and proof by inductions can be written. Modifying the unification method was an unexpected problem to tackle during implementation but proved to be a very interesting challenge. Extended work that was done to allow the user to define multiple rules in an axiom element has laid a solid foundation for further work to be carried out on the proof by induction to improve the user experience.

Now that it is possible to apply rules by elimination and write proofs by induction in Holbert, and in combination with the extensive work done by my colleague Rayhana Amjad on equality and rewriting, Holbert is in a promising position to be brought to the classroom soon. I have thoroughly enjoyed working on Holbert this year and look forward to following its progress in development and introduction to the classroom.

Bibliography

Liam O'Connor. Holbert. URL `http://liamoc.net/holbert/`.

INRIA. *The Coq Proof Assistant Reference Manual*. 2009.

Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3-540-43376-7.

Julien Narboux. Toward the use of a proof assistant to teach mathematics. *International Conference on Technology in Mathematics Teaching*, 07 2005.

Benjamin C. Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. In *International Conference on Functional Programming*, page 121–122, Edinburgh, Scotland, 2009. Association for Computing Machinery. ISBN 9781605583327. doi: 10.1145/1596550.1596552.

Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2017. URL `http://www.cis.upenn.edu/~bcpierce/sf`. Version 6.1.

Frédérique Guilhot. Formalisation en coq d'un cours de géométrie pour le lycée. 01 2004.

Ulf Norell. Dependently typed programming in Agda. In *Types in Language Design and Implementation*, pages 1–2. ACM, 2009. ISBN 978-1-60558-420-1. doi: 10.1145/1481861.1481862.

Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda*. Electronic textbook, 2020. URL `http://plfa.inf.ed.ac.uk/20.07/`.

Kevin Buzzard. What is the Xena project?, 2017a. URL `https://xenaproject.wordpress.com/what-is-the-xena-project/`. Accessed Febuary 2021.

Kevin Buzzard. Formalising Mathematics, 2022. URL `https://github.com/ImperialCollegeLondon/formalising-mathematics-2022`. Accessed Febuary 2021.

Athina Thoma and Paola Iannone. Learning about proof with the theorem prover

lean: the abundant numbers task. *International Journal of Research in Undergraduate Mathematics Education*, 07 2021. doi: 10.1007/s40753-021-00140-1.

Athina Thoma and Paola Iannone. Learning proof with lean, 04 2019.

Kevin Buzzard. xena, 2017b. URL `https://github.com/kbuzzard/xena`. Accessed Febuary 2021.

Wen Kokke, Jeremy G. Siek, and Philip Wadler. Programming language foundations in Agda. *Science of Computer Programming*, 194:102440, 2020. ISSN 0167-6423. doi: 10.1016/j.scico.2020.102440.

Carl Eastlund, Dale Vaillancourt, and Matthias Felleisen. ACL2 for freshmen: First experiences. In *ACL2 Workshop*, 01 2007.

Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, USA, 2000. ISBN 0792377443.

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

Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 253–281, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

Simon Marlow, editor. *Haskell 2010 Language Report*. 2010.

Tobias Nipkow. Functional unification of higher-order patterns. In *Logic in Computer Science*, pages 64–74, 1993. doi: 10.1109/LICS.1993.287599.

Ronald Harrop. On disjunctions and existential statements in intuitionistic systems of logic. *Mathematische Annalen*, 132(4):347–361, 1956.

Alonzo Church. The calculi of lambda-conversion. *Annals of Mathematics studies*, 6(4), 1941. doi: 10.2307/2267126. Lithoprinted in Journal of Symbolic Logic 6(4), 171-172, Cambridge University Press.

Thierry Coquand. An analysis of Girard's paradox. In *Symposium on Logic in Computer Science*, pages 227–236. IEEE Computer Society Press, 1986.

Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940. ISSN 00224812.

N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 75(5):381–392, 1972.

Arthur Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, 49:1–46, 10 2012. doi: 10.1007/s10817-011-9225-2.

Conor McBride and James McKinna. Functional Pearl: I Am Not a Number–I Am a Free Variable. In *Haskell Workshop*, page 1–9, Snowbird, Utah, USA,

2004. Association for Computing Machinery. ISBN 1581138504. doi: 10.1145/1017472.1017477.

David Johnson. The Miso framework, 2021. URL `https://github.com/dmjio/miso`. Accessed October 2021.

Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information & Computation*, 115(1):38–94, 1994.

Martin Odersky. An overview of the scala programming language. *École Polytechnique Fédérale de Lausanne (EPFL)*, 2, 2006.

Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *The Art, Science and Engineering of Programming*, 1(7), 2017.

Oleg Grenrus. Optics: Optics as an abstract interface, 2021. URL `https://hackage.haskell.org/package/optics-0.4`. Accessed October 2021.

Evan Czaplicki and Stephen Chong. Asynchronous functional reactive programming for GUIs. In *Programming Language Design and Implementation*, pages 411–422, New York, NY, USA, June 2013. ACM Press.

Antony Courtney and Conal Elliott. Genuinely functional user interfaces. In *Haskell Workshop*, September 2001.

Facebook. React: A JavaScript library for building user interfaces, 2021. URL `https://reactjs.org/`. Accessed October 2021.

Rayhana Amjad. Equational Reasoning in the Holbert Proof Assistant. The University of Edinburgh, 2022.

Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, page 152–158, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916506. doi: 10.1145/191666.191729. URL `https://doi.org/10.1145/191666.191729`.

Peter Dybjer. Inductive families. *Form. Asp. Comput.*, 6(4):440–465, jul 1994. ISSN 0934-5043. doi: 10.1007/BF01211308. URL `https://doi.org/10.1007/BF01211308`.

Lawrence C. Paulson. *A Fixedpoint Approach to (Co)Inductive and (Co)Datatype Definitions*, page 187–211. MIT Press, Cambridge, MA, USA, 2000. ISBN 0262161885.

Jakob Nielson. How to conduct a heuristic evaluation, 1994. URL `https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/`. Accessed March 2022.

Jennifer Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 5 edition, 2019. ISBN 978-1-119-54725-9.