Treehehe: An interactive visualization of proof trees

Chelsea Battell

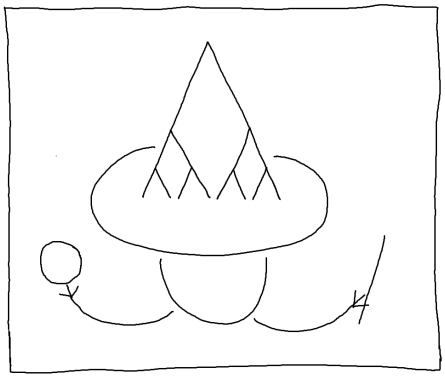

Fig. 1: Experience the magic of interactive proof trees

Abstract—Proofs are commonly illustrated as trees to make the structure of the argument salient. A proof tree can be tedious to record and as a static object it does not realize its full potential as a route to comprehension of the proof it represents. There is literature on how to use visualization to support mathematics education and implementations of visual tools to walk through real analysis proofs, but none of this related work addresses the use of interactive proof trees or visualization for the proof theory community. To address this gap, we implement a tool for interacting with visual representations of proof trees to provide insight into the structure of the proof and its founding logic. The proof visualization tool allows either open exploration of a proof or a directed walk-through, revealing supplementary information to serve as a form of discourse as the nodes are visited. This work provides a tool for gaining understanding of the structure of proofs, insight into the processes used in constructing such proofs, and also serves as a starting point for visualizations of proof trees in more complicated logics.

Index Terms—information visualization, visual analytics, proof visualization, proof tree, proof theory, mathematics education

1 Introduction

Proofs are naturally visual artifacts and an essential part of comprehension and advancement in logic. They are a requirement for confidence in the logician's intellectual explorations and constructions. A proof is a logical argument providing evidence of the truth of some statement. It connects assumptions and a goal through the application of rules that preserve truth. These arguments have a tree structure with the goal as the root and the assumptions as leaves. We call a drawing of a proof as a tree a *proof tree*. This work presents a tool to allow a user to explore proof trees and to gain insight in to specific proofs and the logics they are built from.

Proof trees are a common visual representation used to illustrate both the structure and details of a logical argument. There are many advantages to be found using proof trees: they are useful for learning about a logic through experimentation with writing proofs, they reveal structure in the proof that a linear proof presentation may obscure, they allow cognitive offloading when working through a challenging argument, and they can serve as a form of documentation.

There are critical limitations in the use of proof trees in certain mediums. Even a straightforward, "small" proof tree can easily escape the bounds of a sheet of paper. Creating a digital version eliminates bounds on the size of the tree.

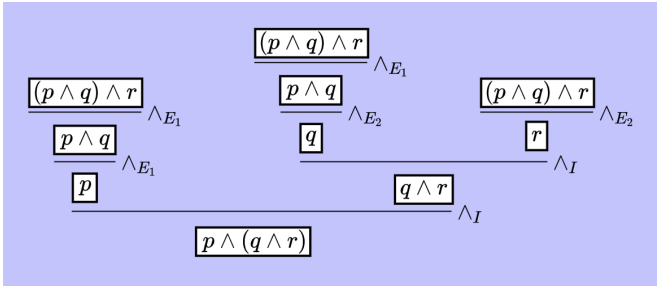

Fig. 2: A proof tree showing that the symbol $\wedge$ is associative in the logic used

A proof tree is a static object which requires some level of expertise to be parsed by the reader. Treehehe adds interaction to proof trees to help the user understand and explore them. In [4], Duval criticizes proof trees and mathematics visualizations by arguing that they don't aid in operational or discursive comprehension. In this case he is speaking of static graphics and figures used when learning mathematics. To overcome this limitation of static proof trees, the user is able to interact with the proof in two distinct ways.

The first is in the form of a proof walk-through that supports operational comprehension by guiding the user through the steps of the proof. At every stage in the walk-through, there is a node that is focused. The second form of interaction allows more free-form exploration of the proof tree. The user can click on any node to focus on it. Colour is used to bring attention to the currently focused node and to distinguish between previously seen inferences and future inferences. Past nodes are coloured to be less visible. Another panel on the page has

• *Chelsea Battell is with the School of Electrical Engineering and Computer Science at the University of Ottawa. E-mail: hello@chelsea.lol. December 2018.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

information on how the focused node is derived to provide discourse.

In [1], Alcock and Wilkinson present a tool for walking through analysis proofs. The proofs are written as they typically would be in analysis lecture notes, with a combination of text explanations and mathematical notation. Users are able to step through a proof and graphical elements are used to bring the reader's attention to the current point in the proof, as well as link to dependencies elsewhere in the proof. Tools like this do not appear to exist for visualizing proof trees. We hope to provide the logic and proof theory community with a resource with benefits similar to what e-proofs provide in analysis lectures. We also see Treehehe as having utility in addition to increased comprehension, such as sparking further insight on logical systems and being a resource for experimentation and documentation. We also hope to add more features to further this goal. This is discussed in Section 9.

Before further discussion of the visualization tool, we will take a step back to review the logic concepts needed to understand how to use it in Section 2. Next in Section 3 we look at the original design, followed by the current design in Section 4, focusing on the tool description and expected user tasks. Following an initial discussion of the tool and how to use it, we discuss the technologies used to implement it in Section 5. InfoVis results motivating the design of the tool will be explored in Section 6. Finally, in the remaining sections we discuss limitations and deeper topics related to the tool, future work, and finish with a conclusion reviewing what is presented here. We hope the reader will be left with an understanding of the motivation for and use of Treehehe and know how InfoVis techniques implemented here help users to learn about specific proofs and proof systems.

2 Logic Background

Recall that the deliverable of Treehehe, the tool described in this report, is an interactive visualization of proof trees. Before we can discuss the tool in detail, it is necessary to first review the application area. We will present the basics of logic, inference systems and proof trees from these systems. We will aim to discuss these preliminary concepts generally, while carrying through the specific example of natural deduction, first proposed by Gentzen [6], as our logic. Our presentation of the natural deduction system is motivated by [8, 1.2 Natural Deduction] and the lecture notes of Frank Pfenning [10].

A *formula* is an expression that represents a true or false statement. We can write a grammar for the formulas of a given logic. For example, in natural deduction formulas are described by the following grammar:

$$F ::= \top \mid \bot \mid F \wedge F \mid F \vee F \mid F \supset F$$

Note that below we allow parentheses to be used to denote precedence.

The symbols $\wedge$, $\vee$, and $\supset$ are called *connectives*. Think of a connective as notation for an adhesive that connects formulas. The connectives in the grammar for natural deduction formulas are all binary connectives because they all build a new formula from two smaller ones. The symbols $\top$, $\bot$, and the connectives are the *logical constants* of this language of expressions.

The grammar tells us how to build formulas, but not what they mean. For example, given formulas p, q, and r, we can build the expression $((p \wedge q) \vee r) \supset ((p \vee r) \wedge (q \vee r))$ from the grammar above, but without any understanding of the semantics of the logical constants this is simply a string of symbols without meaning. We don't know if the expression is true or false. The semantics of formulas can be encoded in a set of inference rules for the logic.

$$\text{conditions}\ \frac{\mathit{Premise}_1 \quad \ldots \quad \mathit{Premise}_n}{\mathit{Conclusion}}\ \text{rule name}$$

Fig. 3: Structure of inference rules

An *inference rule* is a structure that tells us how to derive a formula. The notation for an inference rule is as shown in Figure 3. It has the premises of the rule written beside each other, with consistent space in between, above a horizontal line which is above the conclusion of the rule. The name of the rule is written to the right of the horizontal line and any side conditions that must hold are written to the left of the line. There may be some slight variation in laying out inference rules in other writings, but here we will follow the conventions described above. The meaning of this inference rule is "if we either assume or derive $\mathit{Premise}_1$ to $\mathit{Premise}_n$, then we can derive *Conclusion*".

$$\frac{P_1 \quad P_2}{P_1 \wedge P_2}\ \wedge_I \qquad \frac{P_1 \wedge P_2}{P_i}\ \wedge_{E_i}$$

$$\frac{P_1}{P_1 \vee P_2}\ \vee_{I_i} \qquad \frac{\begin{array}{ccc} & \overline{P_1}^{\,u} & \overline{P_2}^{\,w} \\ & \vdots & \vdots \\ P_1 \vee P_2 & P_3 & P_3 \end{array}}{P_3}\ \vee_{E^{u,w}}$$

$$\frac{\begin{array}{c} \overline{P_1}^{\,u} \\ \vdots \\ P_2 \end{array}}{P_1 \supset P_2}\ \supset_{I^u} \qquad \frac{P_1 \supset P_2 \quad P_1}{P_2}\ \supset_E$$

$$\frac{}{\top}\ \top_I \qquad \frac{\bot}{P}\ \bot_E$$

Fig. 4: Inference rules for natural deduction

Inference rules tell us how to build a formula or take it apart. These rules give meaning to the logical constants used in them. For example, see Figure 4 for the rules of natural deduction. Rules with names containing "I" are called introduction rules because a logical constant is introduced in the conclusion of the rule. Rules with names containing "E" are called elimination rules because a logical constant that occurs in one of the premises is eliminated and is not present in the conclusion.

We will explain the rules $\wedge_I$ and $\wedge_{E_i}$ since they are used in an example proof later in this paper, but we do not describe the full set in Figure 4 here. For a more complete explanation of the rules, we direct the reader to the lecture notes by Pfenning [10], the textbook Logic in Computer Science [8] by Huth and Ryan, or to the Wikipedia entry on natural deduction. By considering the collection of natural deduction rules one can deduce that the symbols $\top$, $\bot$, $\wedge$, $\vee$, and $\supset$ mean "true", "false", "and", "or", and "implies", respectively.

We understand the rule $\wedge_I$ to mean if we can assume or derive P_1 and P_2, then we can use this rule to derive the formula $P_1 \wedge P_2$, introducing a formula with the symbol $\wedge$. The rule $\wedge_{E_i}$ tells us that if we can assume or derive the formula $P_1 \wedge P_2$, then we can derive each of P_1 and P_2 on their own by applying this rule, eliminating a formula with the symbol $\wedge$. This is consistent of our natural understanding of the word "and", which the logical constant $\wedge$ represents. Later we will see how to use these rules to prove that $\wedge$ is commutative; that is, expressions $p \wedge q$ and $q \wedge p$ have the same truth-value.

Suppose we wish to prove that from the n assumptions $A_1, \ldots, A_n$, we can derive the conclusion C. This goal can be written as $A_1, \ldots, A_n \vdash C$, called a *sequent*. So a sequent is an object containing a set of formulas representing a set of assumptions and a formula representing a conclusion, notated as just shown.

In our application, the root of a proof tree is the conclusion to be derived and the leaves are assumptions. We understand a proof tree with leaves $A_1, \ldots, A_k$ and root C built with correct application of rules and consistent substitutions for rule variables to be a proof of the sequent $A_1, \ldots, A_k \vdash C$. For example, in Figure 5, we see a tree with the unique leaf formula $p \wedge q$ and the conclusion $q \wedge p$. If the inferences are all valid, then this is a proof of $p \wedge q \vdash q \wedge p$.

$$\dfrac{\dfrac{p \wedge q}{q}\wedge_{E_2} \quad \dfrac{p \wedge q}{p}\wedge_{E_1}}{q \wedge p}\wedge_I$$

Fig. 5: Proof that $\wedge$ is commutative

We wish to check that the proof of $p \wedge q \vdash q \wedge p$ in Figure 5 is valid. To check this proof, we will begin at the root. The formula at the root of the proof tree, our conclusion, is $q \wedge p$. The only rule that could have been used to derive this formula is $\wedge_I$ applied to premises q and p (in that order). The first premise can be derived by the rule $\wedge_{E_2}$ applied to the premise $p \wedge q$. We stop this line of reasoning here and allow $p \wedge q$ to remain an assumption. Back to the first inference considered, we had a second premise p. The formula p can be derived by the rule $\wedge_{E_1}$ applied to $p \wedge q$. We are already seeing $p \wedge q$ as an assumption, so we can halt our proof search/check here. We now see that the proof tree in Figure 5 is a valid proof tree providing evidence of $p \wedge q \vdash q \wedge p$.

Notice that in working through the example proof above, we began with the conclusion of the proof, rather than working from assumptions to conclusion. We favour the former backward reasoning style, but one could also read the proof in the forward direction, beginning with assumptions. This choice will be discussed further in Section 8.

3 Design

The main content to display in this tool is a proof tree, so we will first discuss the display of proof trees. For this task we follow the conventions for drawing proof trees in the math and logic community. Namely, uses of inference rules similar to those in Figure 4 are chained together to build a tree with the proof conclusion at the root and assumptions at the leaves. More detail on the construction of proof trees can be found in Section 2.

We wish to be able to direct attention of the user to particular nodes in the tree. This is done using colour to highlight a node for focus and lightly bring attention to the children of the node, since they will follow in a proof traversal. Compare the Treehehe version of the proof that $\wedge$ is commutative in Figure 6 with the standard proof tree showing the same fact in Figure 5.

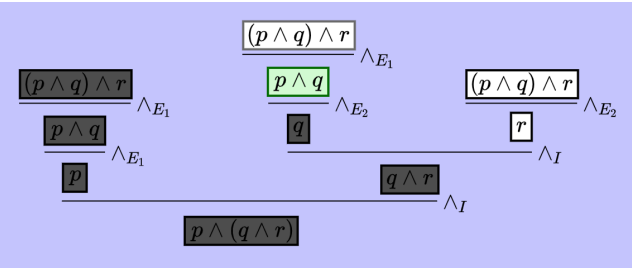

Fig. 6: Treehehe version of proof of associativity of $\wedge$ with node $p \wedge q$ focused

Important goals for the design of the project were to make it comfortable for users to explore a proof without necessarily being told what to do. To provide some form of discourse, supplementary information explaining how a selected node was derived and connecting it to the inference system was important.

It was clear that the following regions needed to be available on the page: title, example selection, tree display, selection information, and rules. See Figure 7 for a sketch of the initial page design. We also wanted to have controls to toggle between explore and walk-through modes and possibly alternative views for the tree, but we did not end up implementing alternative tree views. The distinction between walk-through and explore modes is now more in alternative ways of using the system, and we do not force this to be done by changing the mode of the entire application. Thus neither toggle control is included in the final version.

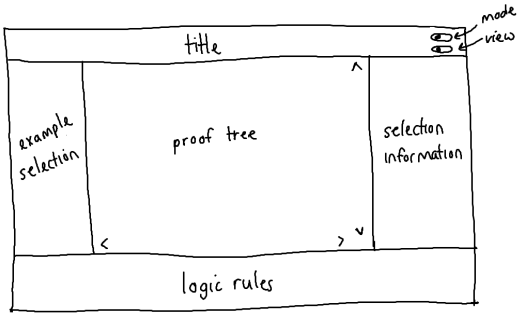

Fig. 7: Original Treehehe design

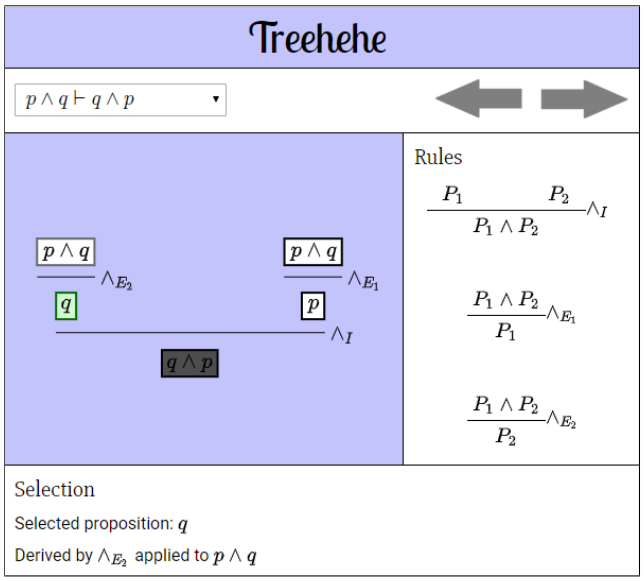

Fig. 8: Screenshot of Treehehe with green node containing q focused

Controls that had not been considered in the initial version include navigation arrows for the walk-through and a drop-down list for the example selection. By containing the list of examples in a drop-down menu, the examples display no longer required a large amount of vertical space. It also became clear that the information on the selected node is usually a few long lines with some large spans of math content. From these insights the page display was redesigned. The content that was displayed did not change, but the location of much of it did. The current version of the design is described in detail in the next section.

4 Tool Description and User Tasks

Treehehe is implemented as a webpage. A screenshot of the page can be seen in Figure 8. The major elements at the top of the page include a panel for the title and any future menu and below this a controls panel for example selection and stepping through a proof.

First the user selects an example from the drop-down list in the controls panel. We have chosen examples from [8], an introductory logic textbook, to ensure a breadth of examples accessible to beginner audiences, although more complicated examples from varied logics could easily be added to the system. This is explained in Section 8. Example selection updates the main display area of the page, the tree display panel, with a proof tree corresponding to the selected example.

The selected example proof tree represents a proof within a specific logic, so once an example is selected the rules panel to the right of the tree display panel is populated with the rules of that logic. These rules can be reviewed on their own or referenced as they are used while working on understanding a proof.

Once a tree is displayed, the user has the option of open exploration of the proof, called *explore mode*, or a directed walk-through, called *walk-through mode*. In explore mode, any node containing a formula can be selected for focus. Once this is done, nodes in the tree are coloured to make the selection most salient, while also highlighting the children of the current node and making nodes occurring earlier in the proof traversal less visible. At the same time, information related to the focused node is displayed in the selection panel.

Recall that our reading of the example proof in the previous section applied reasoning in the backward direction from goal to assumptions. With this in mind, the focused node selection information includes the formula contained in the selected node, the rule used to derive this formula, and the instantiated premise(s) of that rule. For example, in Figure 8, the selected formula is the formula q, which is derived by applying the rule $\wedge_{E_2}$ to the premise $p \wedge q$. So the rule $\wedge_{E_2}$ is used with P_1 and P_2 instantiated as p and q, respectively.

In walk-through mode, the information displayed for the focused node and colouring of nodes in the tree is the same as in explore mode, but the method for focusing nodes is different. In a walk-through, the arrows in the controls panel of the page are used to navigate forward and backward through the steps of the proof, again with a backward reasoning reading. The nodes are visited following a preorder depth-first traversal. This is a reading of the proof that seems intuitive to the author, but could easily be changed to other traversals if they seem a better fit. We consider this further in Section 8.

5 Implementation

This project is implemented as a webpage, so the main languages used are the standard web development languages: HTML, CSS, and JavaScript. Trees are implemented as JSON objects. Although the design does not prioritize small-screen devices, Flexbox has been used to help make the page responsive. By implementing the tool as a webpage, it is more easily accessible for a wide audience. The lack of type safety in JavaScript may make it a poor choice for a project where correctness is of vital importance, but it is a good choice for relatively quick development of a visualization project. The page is hosted using GitHub pages. As of this writing, it can be found at http://chelsea.lol/treehehe. This tool has been tested on Google Chrome v70.0.3538.110.

Proof trees are laid out following an efficient generalization of the Tilford-Reingold algorithm, described in more detail in Section 6. Rather than implementing this algorithm directly, we use D3 trees. A D3 tree accepts an object that contains at least fields for name and children, recursively nested with each child containing the same fields, and computes the positions of the nodes according to the tree layout algorithm.

A proof tree is drawn quite differently from standard tree drawings, and this causes some challenges in implementation. Rather than being represented as a point, a node in a proof tree contains a formula so nodes have varied widths. Also, we don't draw links or edges between connected nodes. Instead, we separate a node from its children with a horizontal line with the rule name to the right of the line and any side-conditions to the left, following the convention for drawing inference rules presented in Section 2. Note that the rule name and side condition are data connected to the node below the inference line.

Before discussing the differences in the tree data, we will look at how proof trees are similar to standard tree objects. Proof trees have nodes, and there is a relationship that can be defined between the nodes. The relations representing the invisible tree edges are "is derived from" and "is used to derive". There is a unique root, and a unique path from the root to any leaf. We can see that a proof tree is in fact a tree, but in this specific application we attach some extra data to each node.

Every node has a field for a formula. Recall this is some expression that is either true or false. Each logic will have different expression grammars and rules for how to construct these expressions. Non-leaf nodes all have a field for the name of the rule that was used to derive the contained formula. Leaf nodes that represent hypothetical assumptions (for example, P_1 in the premise of the rule $\supset_I$ in Figure 4) use the rule name field for the hypothesis label. Leaf nodes representing unlabeled assumptions of the proof have no value in the field for rule name. Note that our abstraction of the tree object favours the backward reasoning discussed in the proof check in Section 2, because at each node, this rule name field tells us "how did we get here?" rather than "how do we keep going?", and directs the user from a node to its children.

The standard audience for this work will typically have seen math content and proof trees laid out as when using LaTeX languages. Here we try to respect the traditions from which this work is motivated. The math content seen in this tool is written in LaTeX and embedded in HTML using MathJax. Since we want the tree to be interactive, we don't write the full tree in LaTeX, but only the node content.

A few challenges arise in our handling of the layout of math content. Since the mathematical expressions are initially written in LaTeX, they are almost always much larger than the final typeset expression. For example, the expression \$(p \wedge q) \vee r\$ requires more width than its typeset version $(p \wedge q) \vee r$. This means that D3 allocates more space to these nodes than they need, which is fine, but this has implications for drawing other tree content.

To draw the inference line between a node and its children and draw the rule name and any side conditions, we need to know the location of the bottom left of the first child and the bottom right of the last child after typesetting. So the content of the tree must be drawn in a very particular order. First, all nodes are positioned according to the content of their "formula" field, then once MathJax has finished this first typesetting, we can position the inference line and rule name. To position any side condition or text to the left of the inference line, we need to know the typeset width of the side condition text so we know how far to shift it to the left. To make this possible, we add the side condition to the page after the initial node positioning so that it will be typeset by MathJax, but we don't display it yet. Then once MathJax has run again, we can position the side condition properly.

6 InfoVis Elements

Data in this visualization project are trees, so an important first consideration is determining a visual formalism for drawing trees that is consistent with conventions for this task. Herman et al. [7] discuss a number of techniques for visualizing trees. Several space saving representations are presented, such as H-tree layout, radial view, and balloon view. In Treehehe, much of the utility of a proof tree comes from being able to read a flow of inferences, seeing the leaves as axioms or assumptions and the root as the conclusion of the proof. Each node contains a propositional formula, so to be readable all nodes need to have the same orientation.

The space-saving layouts could be useful if it is clear which node is the root and some other form of iconography could be used for constructing formulas. For the current project, we will use the Reingold and Tilford layout for trees, which could be described as visually rooted, since we have a convention that tells us which node we see as the root. We want to allow for inferences with more than two premises, so we need the version of the Reingold and Tilford algorithm extended by Walker to general trees [9], then made more efficient to run in linear time by [3].

We will invert the standard tree constructed with this algorithm so that the root is at centre bottom as is conventional for drawing proof trees. The bottom center location of the root is a consistent position where we can find the goal of the proof, which is also the formula on the right side of a sequent, as described in section 2. Another benefit of the Reingold and Tilford algorithm for this application is the vertical alignment of nodes at the same level (distance from the root) in a proof. This permits a final difference between proof trees and general trees without making the relations between nodes unclear: we don't draw and links or edges between nodes. Instead, a horizontal line is drawn between a node and its children.

Focus+context techniques as discussed in [7] are naturally applied in Treehehe. When we focus on a node, we maintain the context of the node in the tree as we can see its relation to nearby nodes. We emphasize the context by adjusting the colour of the children and parent of the focused node.

In [11, Chapter 3] we learn many best practices from Ware for using colour in a visualization. In Treehehe, colour is used to highlight a selected node in a proof tree to bring the focus of the user to the current inference of interest. We also use colour to categorize nodes. Colour is preattentively processed, so as described in [11, Chapter 11] the user is able to have attention on spatially disconnected elements without each element being the current fixation. We use this property of attention to allow the user to attend to and distinguish between groups of nodes, such as previously visited nodes, future nodes, and the current focus.

Figueiras presents a taxonomy of interaction consisting of eleven categories of techniques [5]. The categories of interaction used in this project are *select* and *overview/explore*.

A user may want to select data to learn more about it or track how it changes in response to other interaction. In Treehehe, a node can be selected to see more detail on its role in the proof tree. Its related parent and child inferences can be highlighted, the rules used displayed, and the substitutions used in applying the rules explicitly stated.

Techniques in the overview/explore category first display an overview of the data, then allow exploration through zooming, filtering, and the display of details on demand. Large proof trees may sometimes escape the bounds of a monitor, so it may not be possible to have a full proof tree visible. Thus a proper "overview" is not guaranteed in all cases, but the full tree is available in the application, possibly requiring panning. The user can then explore the proof through the guided walk-through, seeing more details as the proof progresses and nodes are focused.

Yi et al. observe that sensemaking is one path to insight [13]. Sensemaking, in the context of interactive visualizations, is an intentional process in which a person continually reframes their understanding of a concept. The authors also propose four processes for gaining insight: provide overview, adjust, detect pattern, and match mental model.

Provide overview means a user can gain a higher-level understanding of a data set. In Treehehe, this is realized in the viewing of a complete proof tree.

Through adjust, a user is able to explore a data set. This can come from a variety of interaction techniques, such as adjustments to the level of abstraction, or selecting a range of values. In Treehehe, the user can step through a proof using navigation arrows, causing updates to the selected node and supplementary proof guidance related to that node.

When new structure is observed in data and possibly new discoveries made, the user has been able to detect a pattern. In Treehehe, examples of how this is realized include observing repeated arguments in a proof and being able to detect loops.

Match mental model means one has a bridge between the data and their mental model of it. Having an external visual version of the mental model allows for cognitive offloading. In this project, we will see the value of having a visualization of a proof to interact with rather than trying to hold the entire picture in one's head. It is then possible to allocate more mental resources to reasoning and gaining insight through the other processes.

7 Evaluation

Here we do a preliminary informal evaluation based on the author's experience in using Treehehe.

On initial page load, the first example from the example selection menu is displayed. It may be better if initially there was no example to be displayed so that there is less information on the screen on load.

It would be useful for the user to have more guidance on how to start using the tool. Being initially presented with some instructions on how to get started is necessary. If the user is unfamiliar with proof trees, a pointer to a basic logic tutorial will be helpful.

It is possible that a user without any logic background could experiment by clicking on the nodes, reading the selection information, and see patterns between the concrete proof tree with the rules in the rules panel. This is an ideal use of the tool. We want users to learn about proofs through exploration.

If a user clicks on a non-root node during exploration, then all nodes that would have been visited earlier in the programmed traversal are coloured to be less visible. This might be confusing, since different traversals are possible and the user will not have experienced the full traversal up to this point. The multiple sudden changes may be distracting.

The proof walk-through is initiated by pressing the "forward" arrow in the top right. This highlights the root node. As discussed earlier, the currently programmed traversal is depth-first preorder. We are starting at the root of the tree which is the conclusion of the proof, and we reason in a backwards direction. In this situation, it is useful to have previously visited nodes made harder to read, so that the user is focused primarily on the current node, but can also see ahead in the walk-through. At the end of the walk-through, the forward button can be pressed, but the walkthrough does not go beyond having the last node in the traversal highlighted. This could be improved by having a more clear indication that traversal is done. The forward button could be unavailable, or the final node is coloured as the previously visited nodes.

It would be pleasant if there was a way to "escape" from having a node focused, so that there are no longer any nodes with special focus.

Overall, it is easy to explore proof trees and see supplementary information. The display is generally pleasant. There are many small changes that could substantially increase how useful this tool is in practice.

In the future we hope to perform a more detailed evaluation following the cognitive dimensions of Blackwell and Green [2].

8 Discussion

There are many decisions made in implementing and describing this project that the reader may wish to see motivated or explained further. We attempt to address these here. These extra discussions may be of interest to the curious reader but can be freely skipped.

In this paper we have used natural deduction as the example system when presenting the logic background and describing how to use the tool. In the current version of Treehehe, most examples are natural deduction proofs. This logic was chosen because it is usually the first inference system presented in an introductory logic course, so it made sense as the system to use to make this paper readable to as wide an audience as reasonable. It also helped keep the presentation of background information manageable. A critical observation to make here is that there is nothing in the system that requires a user to commit to a single logic.

Since the tree data is a JSON object, the tree content can contain any string and thus be from any logic. An implication of this is that there is no check for correctness of the proof being displayed. In its current version, Treehehe is only for visualization and not verification.

In the version of Treehehe released with this paper, there is one example that is not a natural deduction proof. This is to illustrate the points just made. This example is a proof of $\Sigma; P \vdash \exists M, \mathit{length}\ (1 :: 2 :: 3 :: [])\ M$, a formalization of a Prolog query. Without being too pedantic, this query asks if there is an M such that M is the length of the list $(1 :: 2 :: 3 :: [])$. The expression query above is the root node of the proof tree. In this example in the webpage we can see how this query is proven in Prolog and how the value of M is computed to be 3.

A reader more familiar with, and favouring, the Martin-Löf approach to natural deduction may be wondering why we have not made an explicit distinction between formulas and judgments as in [10]. What we mean here is if we want to derive that formula p is true, the node in the proof tree should say this explicitly, i.e., the node would contain the judgment *p true*. The current presentation has preferred the more condensed presentation where whenever we see a formula p in a proof, we assume that this means p is true. Aside from a smaller use of space, another benefit is we avoid the discussion of judgments vs formulas, making the presentation easier to digest for less experienced users and readers. As mentioned above, the experienced user could build a proof tree in this system where the judgment is clearly distinguished from the formula.

In Treehehe, we have used a depth-first preorder traversal. A traversal had to be chosen to implement the proof walk-through and this choice of traversal felt intuitive to the author. If a user wishes to use a different traversal or it is found that another traversal would improve cognition in learning about proofs and logic, the change could be made fairly simply in the system. A two-way iterator was implemented to step forward or backward through the proof. This iterator accepts a callback representing a function to traverse a tree. This could either be updated only in the code, or a control could be added to the page so the user could select from a few traversal options.

Related to the traversal discussion, when we walked through an example in Section 2, we began with the root node. This backward reasoning is useful in automated reasoning, because we know we are starting with the root node and there is typically a finite set of rules that could have been used to infer that node (often only one rule). Starting with assumptions or leaves of the tree might feel more natural, because we start with what we know and work towards a goal. The issue with forward reasoning when constructing a tree is we don't know ahead of time what shape the tree will take and there is nothing in the shape of axioms or assumptions to guide our construction. So in building a proof without already knowing it, there is often a more algorithmic approach to the tree construction with backward reasoning than with forward. With all this said, in some logics (including natural deduction) backward reasoning alone is often not sufficiently intuitive, and in this case allowing the user to select different traversals would be useful.

9 Future Work

There are many additions that could be made to this work to implement more lessons from the information visualization community and also to make the tool more useful for a range of proof-related tasks. We will first look at the updates in the InfoVis area.

Since large proof trees can escape the bounds of a digital display, navigation is another topic explored by Herman et al. [7] that is relevant to this project. The user will want to be able to scroll around the graph while exploring. The technique of incremental exploration and navigation can be used to display only a desired subtree of the proof; nodes will have an icon to either exclusively display or to hide the subtree that it is the root of.

Weber and Mejia-Ramos [12] and Alcock and Wilkinson [1] have argued that too much detail visible in a proof obscures high-level structural information which is important for proof comprehension. To address this concern, a future version will have two views for the proof display: detail view and structure view. Detail view will show a proof tree, possibly with subtrees hidden as discussed above, and with full formulas, rule names, and rule side conditions as presented in the rest of this paper. Structure view will show only the shape of the tree and which rules are used.

The interaction technique abstract/elaborate presented by Yi et al. [14] and Figueiras [5] is used to adjust the level of abstraction of the data. By including a separate structure view option as previously described, this tool will have an abstract view displaying the structure and an elaborate view displaying full proof detail.

The final InfoVis updates we propose involve the rules panel on the page. When selecting a node, there is supplementary information displayed below the tree, including the name of the rule used to derive the focused node. We would like to highlight the rule used to derive the focused node so that it is easier for the user to move their fixation to the rule being used and connect the children of the focused node with the premises of the rule. In the other direction, we would also like the user to be able to select a rule from the rules panel and have uses of this rule in the proof tree be highlighted.

Other updates would add more uses for the tool that are extraneous to information visualization. Once a proof is built, the user may want to present it in a LaTeX document. It would be a small change to export the LaTeX code for the tree from the JSON object representing the tree. Another helpful update is to add "help" information in the form of easily readable instructions on how to use the tool and a tutorial on natural deduction. A final modification would allow the user to also "write" the proofs using the tool.

In the current version of Treehehe, there are a number of examples the user can choose to explore, but the tool would be far more useful if there was a way to add custom proofs. The best way for this to be done is to extend the interface so that new nodes can be added. Initial plans for this idea would add an icon to each node to add siblings, and another icon to each leaf node to make that node a result of an inference (i.e. add an inference line above), then children could be added above that node. The content of a new node would be entered in a text box and written in LaTeX.

Once the project is updated to the point where a user can enter their own proof, it should also be a more pleasant experience than trying to write proof trees on paper. When writing a tree by hand on paper, you are more committed to the spatial position of the nodes you write; in Treehehe, you will be able to easily modify the elements of the tree.

10 Conclusion

In this paper we have seen how information visualization techniques can be used to make a tool for visualizing proof trees. Students studying logic in university will see proof trees during their studies. Treehehe should help them gain confidence in using proof trees to construct logical arguments. Advanced users who write proofs in other formalisms, such as the Calculus of Constructions, or write logic program queries, may want to be able to be able to build a visual version of their proofs.

We presented the logical foundations necessary to understand how to use the Treehehe. Following this, we saw how to use the tool, looking at each major area of the webpage, and reviewed the technologies used implementing the tool. Since this is an information visualization project, the InfoVis elements that helped guide the design are described. Finally, we attempted to address other concerns and questions that the reader might have and future modifications to the project.

It is our hope that this tool will be a useful resource for learning about formal reasoning. For more experienced users, we hope it will aid in the understanding of larger proofs in the proof-writing process and for documenting reasoning.

References

[1] L. Alcock and N. Wilkinson. e-proofs: Design of a resource to support proof comprehension in mathematics. *Education Designer*, 1(3), 2011.

[2] A. F. Blackwell and T. R. Green. Notational systems — the cognitive dimensions of notations framework. *HCI Models, Theories, and Frameworkd Toward a Multidisciplinary Science*, pp. 103–134, 2003.

[3] C. Buchheim, M. Jünger, and S. Leipert. Improving walker's algorithm to run in linear time. In *Graph Drawing: 10th International Symposium*. Irvine, CA, USA, August 2002.

[4] R. Duval. Representation, vision and visualization: Cognitive functions in mathematical thinking. basic issues for learning. In *Proceedings of the Annual Meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education*. Cuernavaca, Morelos, Mexico, October 1999.

[5] A. Figueiras. Towards the understanding of interaction in information visualization. In *2015 19th International Conference on Information Visualization*, July 2015.

[6] G. Gentzen and M. E. Szabo. Investigations into logical deduction. In *Studies in Logic and the Foundations of Mathematics*, vol. 55, chap. The Collected Papers of Gerhard Gentzen, pp. 68–131. Elsevier, 1969. English translation of Untersuchungen uber das logische Schliessen, Mathematische Zeitschrift (1935) 176-210, 405431.

[7] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000.

[8] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, second ed., 2004. Chapter 1.

[9] J. Q. W. II. A node-positioning algorithm for general trees. *Software — Practice & Experience*, 20(7):685–705, July 1990.

[10] F. Pfenning. Lecture notes on natural deduction. Constructive Logic course notes (http://www.cs.cmu.edu/ fp/courses/15317-f17/lectures/02-natded.pdf), August 2009.

[11] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufman, third ed., 2013.

[12] K. Weber and J. P. Mejia-Ramos. Mathematics majors' beliefs about proof reading. *International Journal of Mathematical Education in Science and Technology*, 45(1):89–103, 2014.

[13] J. S. Yi, Y. ah Kang, J. Stasko, and J. A. Jacko. Understanding and characterizing insights: How do people gain insights using information visualization. In *Proceedings of the 2008 Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*. ACM, April 2008.

[14] J. S. Yi, Y. ah Kang, J. T. Stasko, and J. A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), November/December 2007.