Advanced Proof Viewing in PROOFTOOL

Tomer Libal
Microsoft Research – Inria Joint Center,
École Polytechnique.
shaolin@logic.at

Martin Riener*
Institute of Computer Languages,
Vienna University of Technology.
riener@logic.at

Mikheil Rukhaia†
Institute of Applied Mathematics,
Tbilisi State University.
mrukhaia@logic.at

Sequent calculus is widely used for formalizing proofs. However, due to the proliferation of data, understanding the proofs of even simple mathematical arguments soon becomes impossible. Graphical user interfaces help in this matter, but since they normally utilize Gentzen's original notation, some of the problems persist. In this paper, we introduce a number of criteria for proof visualization which we have found out to be crucial for analyzing proofs. We then evaluate recent developments in tree visualization with regard to these criteria and propose the Sunburst Tree layout as a complement to the traditional tree structure. This layout constructs inferences as concentric circle arcs around the root inference, allowing the user to focus on the proof's structural content. Finally, we describe its integration into PROOFTOOL and explain how it interacts with the Gentzen layout.

1 Introduction

The need for visualizing data precedes the invention of computers. Even so, the large data processed by computers made this need more explicit and initiated much research in data visualization and particularly in tree visualization. For example, the traditional disk usage analyzers were all implemented as trees, with directories and files represented by nodes and edges denoting the containment relation. In the last decades, the increase in disk space and the increase in number of files that followed, prompted the design of new tree visualization methods which will be more space efficient. One of the first methods was TreeMap [25] which divides a box into several smaller boxes representing the subtrees. Other algorithms made the nodes implicit by drawing fractals [18, 22], added a third dimension [19, 13, 17] or used hyperbolic and other radial approaches to better group subtrees [16, 32, 12, 27]. Treevis.net [24], a visual bibliography of tree viewers, now contains more than 270 different algorithms.

GAPT[1] is a framework providing data-structures, algorithms and user interfaces for analyzing and transforming formal proofs. The framework is very general and implements the basic data structures for simply-typed lambda calculus, for sequent and resolution proofs as well as expansion proofs. Various theorem provers have already been integrated into this framework [7]. In parallel, we have developed a Graphical User Interface called PROOFTOOL [8] which can be used both as a pure visualization tool (with the features like zooming, scrolling, searching, etc.) and as a proof manipulator (allowing to call GAPT's proof transformations such as cut-elimination, regularization, skolemization, etc.). We are continuously extending and improving the system and one such extension was presented in [11].

Sequent calculus proofs are often depicted as trees and in fact, the tree representation was used from the very beginning. Gentzen's representation for sequent calculus proofs can be seen as a variant of an algorithm by Donald Knuth [15]. The child nodes are horizontally aligned in the distance of the width

*Supported by the Vienna PhD School of Informatics.
†Supported by the project No. PG/6/4-102/13 of the Shota Rustaveli National Science Fundation.
[1]General Architecture for Proof Theory, http://www.logic.at/gapt

Submitted to:
UITP2014

of their respective subtrees with their parent node being aligned centrally between them. The vertical alignment is determined by the distance from the root.

However, although this presentation seems natural, it is not well suited for large proof as their structure is no longer visible. Even tracing the ancestors of a formula is cumbersome, since the distance between parent inferences can be very large.

Despite the abundant research done in the field of tree visualization and the fact that proofs are normally represented as trees, little was done so far in integrating these advancements into tools for proof visualization. In fact, the first viewer which was integrated into PROOFTOOL was a traditional tree viewer. Proof General [1], which also manages the proof visualization for provers such as Coq [14], supports the traditional tree view as well. Other systems like *LΩUI* [26] and Theorema [31] provide a structural overview in form of a DAG and a tree, respectively. However, their main focus lies on human-readable proofs where the formula level is directly contained in the text. One of the few graphical user interfaces which deviates is IDV [29]. It renders DAG proofs in the TSTP format [28] using the spring layout [4]. This layout turned out to be insufficient for our needs as is discussed in [6]. The reason that many advancements in tree visualization are only slowly reaching the proof theory community may primarily lie with the fact that only few of the provers care about a visual presentation of the generated object, if they generate it at all. Nevertheless, proof visualization is a crucial tool for analyzing large proofs like the ones we encounter in our work. Therefore, we find it important to search for and integrate efficient tree viewers.

In this paper we propose some criteria for visualizing sequent calculus proofs and use them to analyze the existing layouts. We argue that Sunburst Trees [27] are the most adequate layout and develop a new viewer for PROOFTOOL, the graphical user interface of the GAPT framework, which is based on them. The viewer allows the displaying of the structure of the whole proof at once, to easily identify similar subproofs, to zoom in to relevant parts and to see the relevant inference details. At the same time it is connected to the classical Gentzen layout, which allows the user to focus on a small number of inferences or to be able to see an aspect of the proof which is better displayed using the traditional view. We believe that the visualization of proofs using a structural viewer will be useful for tools other than PROOFTOOL. In this paper we therefore present the benefits of using such a viewer and demonstrate its integration within PROOFTOOL.

The paper is organized as follows: Section 2 introduces the requirements we find most important in proof visualization. In Section 3 we explain our choice of the Sunburst Tree. Section 4 is devoted to the integration of our Sunburst viewer into PROOFTOOL and to a comparison between the two views. Section 5 gives a description of the implementation details and we conclude the paper in Section 6.

2 Criteria for Visualizing Sequent Calculus Proofs

When using the traditional tree layouts, wide node labels often stretch the width of the tree and deform its structure. One reason for that is that the context formulas of an inference need to be repeated along many branches. Moreover, it can also happen that the main or auxiliary formulas become overly wide themselves. Therefore, it is helpful to completely separate the tree structure from the information about the inference itself. Since sequent calculus and the inference viewer work well on the sequent level, we concentrate on the requirements for the structural layout.

In this section, we compile a small set of requirements which were identified as critical for our proof analysis. We have tried comparing our conclusions with other works concerning the aesthetics of mathematical proofs. Surprisingly, they often stay only on an abstract level. Hardy [10] names *unexpectedness*,

inevitability and *economy* as aesthetic properties. The first refers to an element of surprise when a conclusion is reached, which has similarities to narratives [5]. The second stands for a detailed, convincing deduction whereas the third means restricting a proof to the minimal steps in order to prove the theorem. Some works deepen these concepts and introduce case studies [21], but we know of no work which details the relation between graphical notations and the aesthetics of a proof. Therefore, the requirements given here are the result of the authors' own involvement with the formalization of mathematical proofs.

1. *Displaying of large proofs* is one of the most important factors. Proofs containing thousands of inference steps can become very hard to read. We would like to find the most efficient tree representation. For example, despite the fact that proofs are traditionally denoted as trees, the edges between the nodes play a very small role and are not space efficient. Another aim is to be able to represent the full proof on a single screen in a comprehensive way. This is not only useful for exporting purposes, but for tracking changes after the application of proof transformations, such as substitution or skolemization.
2. *Distinguishing between different kinds of rules* is important as some rules, like instantiation rules, give information about the content of the proof while others, like contractions, give information about the shapes of proofs. Different coloring of rules is one way to distinguish between them.
3. Without *easy navigation*, one would not be able to follow the logical progress of the proof. The sub-proof relation should always be obvious and easy to navigate.
4. In many cases, *formula ancestor information* is important in order to relate a sequent with the atomic formulas by which it is implied.
5. Proofs have many uses and one would sometimes like to *focus on different aspects of the proof*. Proof complexity, different instantiations, cuts complexity and contractions may all be important for a prospective viewer of a proof.
6. The ability to relate *shape of proofs* and sub-proofs to their content might also be an important factor since it might allow us to detect redundancies and similarities of content.

3 Choosing the proper tree visualization

One of the most comprehensive bibliographies for research on tree visualization is Treevis.net [24] which contains over 270 different algorithms. Consequently, it is a challenge to pick an adequate algorithm out of the numerous ones which have been published. However, Treevis.net also provides a categorization of the techniques in terms of the criteria of dimensionality (2D, 3D or hybrid), representation (implicit, explicit or hybrid) and alignment (axis-parallel, radial or free). In this section we will first explain our choices with regard to the categories mentioned. We then identify the algorithm satisfying our category requirements and show that it also meets our visualization criteria.

We decided to focus on two-dimensional representations, since there is no general additional structure which could be mapped to the third dimension. Although both explicit and implicit representations of edges would suit our purposes, the later allows us to expect a more compact layout. Consequently, we would like to focus on this case. This is additionally motivated by the fact that sequent calculus proofs often contain a high amount of unary rules, where the edge is then redundant. From the algorithms meeting these requirements, we now excluded those which do not meet criterion 6 from Section 2. The remaining options were unexpectedly low in number. A reason for that is that the large class of layouts based on TreeMap divides a box into equal sub-parts for each subtree. The problem there is

that in a series of identical subproofs connected by binary inferences, each subproof has half the size of the preceding one, making it nigh impossible to recognize their similarity. Fractal layouts have similar problems, whereas grid embeddings [33, 23] do not reflect the similarity of subtrees.

What remained were explicit axis-parallel layouts and implicit radial layouts. The first class consists of improvements on the classical Tidy Tree algorithm [30], whereas the later centers around representing the tree from the root outwards.

Of special appeal to our applications was the Sunburst Tree [27]. It is particularly efficient for *displaying large proofs* due to its radial shape and the fact that it eliminates all edges. One can easily *distinguish different kinds of rules* by setting different colorings. The user can group the rules by their function in the proof and thus separate the proof into parts with logical, equational or quantifier inferences. Together with the branching structure, (sub-)proofs are already distinguishable from each other without referring to the formula level. *Navigation* is similarly simple. A single click into a subproof shrinks the original proof to half its size. At the same time the selected subproof is projected onto the circle around the full proof, giving it sufficient space to see detailed inferences. This kind of stretching can impede the identification of similar structures within the proof. Nonetheless, the combination with the full view allows a comparison on the same level. In order to *focus on different aspects of proofs*, a customization of two parameters is possible. By changing both the coloring scheme as well as the inference width ratio, one can single out instantiations denoted by weak quantifiers, different subproof complexities and other aspects. Another strong point of the Sunburst viewer is its ability to relate content to *proof shape*. This is again achieved by inference coloring and width ratio and by its efficient presentation of a whole proof on a single screen. Finally, a radial layout is also helpful in that it can always be drawn into a square, leaving room on the screen for the inference information.

There is one requirement where the Sunburst viewer falls far behind the traditional tree viewers and this is with keeping *formula ancestor information*. The relationship between a formula and its ancestor, while easily displayed in the traditional viewer, cannot be represented in Sunburst. This raises another important requirement for a useful proof visualization tool, its ability to support different viewers and the switching between them.

4 Integration in PROOFTOOL

We have integrated the Sunburst view as an option accessible from the menu. Choosing this option loads the Sunburst view into a separate dialog window. In addition to the proof, which is displayed on the left side, we display also an inference panel. The information in this panel contains details about the inference: its type, its primary and auxiliary formulas, and quantifier instantiation information, if applicable.

In the remainder of this section, we will describe the new interface in terms of the conditions given in Section 2. To emphasize what is written, we have inserted snapshots of views of actual proofs. The appendix contains information about the names of each proof and of how to load it using our system.

Displaying of large proofs. The traditional Gentzen layout contains abundant and redundant white spaces, not only due to its use edges, but also because it has to create extra horizontal spaces between premises of binary inferences. Therefore, proofs with many binary inferences, even if the formulas are hidden, are too wide to fit on the screen. An example of this can be seen in Figure 1.

Projecting the sequent calculus proof to a circle allows roughly four times more space to render the inferences of a certain level[2]. Since the formulas are hidden, an inference is an easily clickable

[2]Let us assume the Gentzen proof to be an isosceles triangle with base length w_1, and height d. If we further assume the

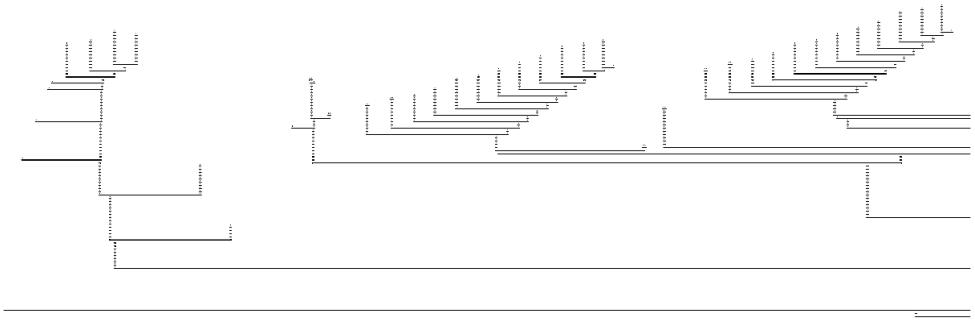

Figure 1: A small part of the Gentzen view of a proof with more than 2000 nodes.

section of the disc, covering the whole area below its parents. Also, hovering over an inference with the mouse cursor triggers a darkening of its bounds. This is particularly helpful when tracing a formula throughout a proof, as one can then easily identify branching without the need to zoom in.

As an example, Figure 2 shows a proof with more than 2000 inferences together with a zoomed in subproof. Comparing the two figures shows that even if we hide all inference information in the Gentzen view, we will still not be able to fit this proof on the screen. In contrast, the Sunburst view allows us to identify the main parts which constitute the proof: the top and bottom side have the same shape, for they contain the same reasoning structure on different terms. Only a small proof, which gives rise to a case distinction, is situated on the left hand side. Just by hovering over or selecting the cut-formulas (colored green), the user can identify the three parts of the proof. The right hand side of Figure 2 shows a zoom into one of the proof instances just mentioned.

Distinguishing between different kinds of rules. This is easily achieved in Sunburst view by coloring the inference depending on the rule type. In order to obtain the highest contrast, we assigned the colors of the rainbow (see Figures 2 and 3) to groups of rules according to Table 1. The relative size

Cut	green	Unary Logical Rule	orange	Strong Quantifier Rule	red
Structural Rule	gray	Binary Logical Rule	yellow	Weak Quantifier Rule	blue
Axiom	gray	Equational Rule	violet	anything else	magenta

Table 1: Rules coloring schemes.

of subtrees to each other is adjustable by defining a so called weight. At the moment, the weight of a subtree is just the number of inferences. Depending on the application, one can imagine metrics which prioritize specific rules or even specific inferences.

Easy navigation. Navigation is very easy in Sunburst, as can be seen in Figure 3. A single click on a subproof shrinks the whole proof to half of its size while displaying the subproof on its outer ring. Navigating backward can be done using the nested original proof. There are also keyboard shortcuts (Ctrl + arrow keys) available in the Sunburst view in order to help users navigate inside the proof. The up key moves selection to the child of a unary inference; the left and right keys

window is maximized, we can estimate the ratio $w_1 : d = 16 : 10$. Then the height d is also the radius of the Sunburst tree giving it circumference $w_2 = 2d\pi$. The ratio of the two lengths is then $w_1 : w_2 = \frac{20\pi}{16} \approx 4$.

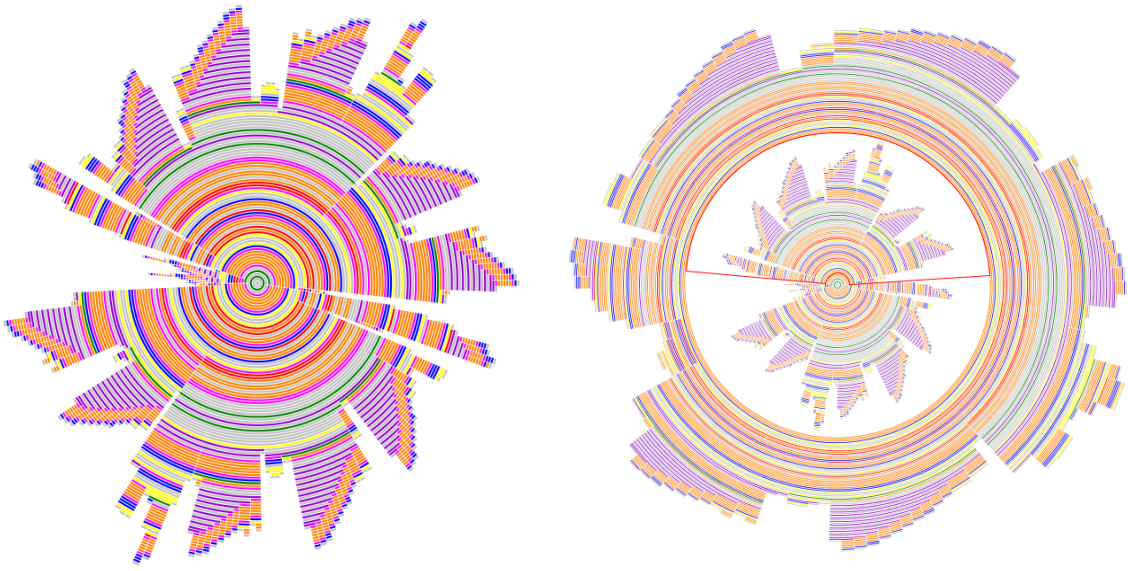

Figure 2: Sunburst view of a large proof in full view (left) and zoomed in (right).

select their respective premises of a binary inference and the down key selects the parent of any currently selected inference.

One drawback of this form of navigation is that zooming into a subproof distorts its shape, affecting the user ability to understand the structure of the proof. Zooming into a subproof distorts all inferences in the same way. We therefore believe that this has only a minor affect on understanding the proof structure, since a human user can easily compensate for this fixed distortion.

The Gentzen layout and the Sunburst view are tightly integrated to make better use of their respective strengths. When one navigates using the keyboard inside the proof in Sunburst view, the Gentzen viewer scrolls to the end-sequent of the selected node and highlights it as shown in Figure 4. There is also a "Show node in LK view" context menu available in the Sunburst view which allows to scroll to the chosen node in the Gentzen view.

Formula ancestor information. The sunburst view window is divided into two parts as shown in Figure 4. The first part shows the structure of the proof, while the second part gives additional information about the selected node. This includes the inference name, its auxiliary and principal formulas, and the substitution used, if any. But still, this information is not enough to see the ancestor relationship as well as it is possible in the Gentzen layout. The two views complement each other in this aspect as is illustrated in Figure 5.

Focus on different aspects of the proof. In order to display different aspects of the same proof, one can take advantage of the possibility to customize the colors and width ratio of inferences in Sunburst. We would like to have a set of such pre-defined customizations which will emphasize different aspects, such as sub-proof and cut complexities, variable instantiations and specific rules and inferences. We plan to implement this feature in the near future.

Shape of proofs. In some situations where the formula level is obscured, it is helpful to concentrate on the structure of the proof. In the following we describe two phenomenons we encountered.

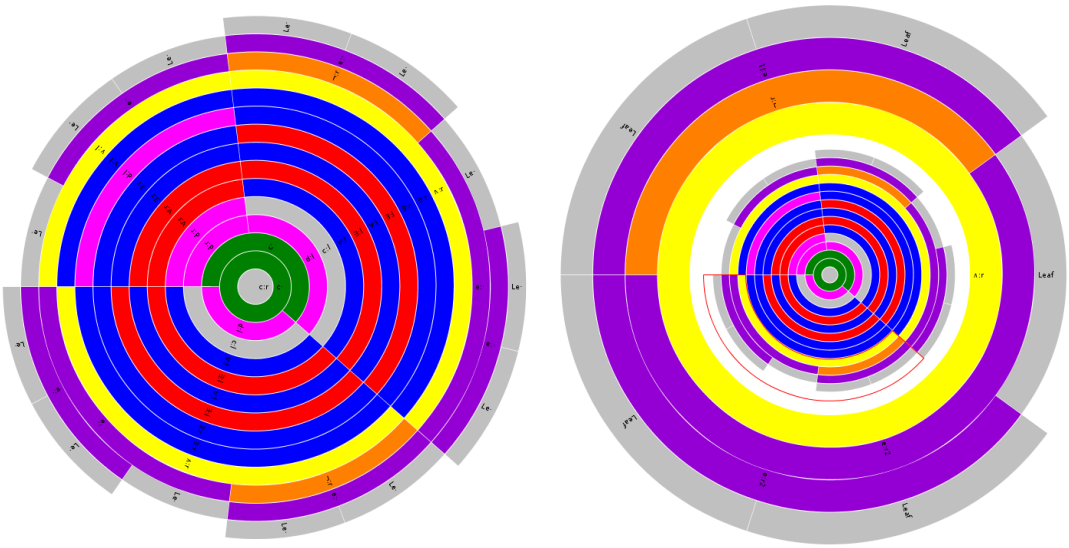

Figure 3: A combinatory proof (left) with a zoom into one of its subproofs (right).

Proof transformations often keep the structure intact. Some proof transformations such as elimination of definition rules and skolemization strongly change the proof on a formula level, but only slightly modify the structural layout. Nonetheless, locating an inference in the Gentzen layout with the find function of PROOFTOOL becomes virtually impossible, since the subterms allowing a unique identification of a formula often have changed. The Sunburst view allows to use the proof structure to find the inference. For example, it is not always clear how a skolem term ends up in a weak quantifier inference, since the term might be carried over from a different part of the proof. In the Sunburst view, we can navigate to this inference and use both views for further investigation.

Understanding proof arguments. In the process of formalizing a proof, one might not recognize all the possibilities where the proof can be generalized. In the Sunburst view, structural similarities are easier to spot and can then be checked whether a generalization is indeed possible.

As an example, we can look at subsequent instances of a formalization of Fürstenberg's proof of the infinity of primes [3]. Here the schematic nature of the proof was already taken into account during formalization, but now the induction argument becomes clearly visible (see Figure 6).

5 Implementation

The GAPT framework is implemented in the programming language Scala [20]. PROOFTOOL makes heavy use of Scala's Swing wrapper library. Details about PROOFTOOL and how it displays formulas, sequents and proofs can be found in [8]. In this section we concentrate on the integration of the Sunburst view in PROOFTOOL.

Our implementation is based on TREEVIZ, an open source library[3] written in Java. Since PROOFTOOL,

[3] Visualization of Large Tree Structures, `http://www.randelshofer.ch/treeviz/`

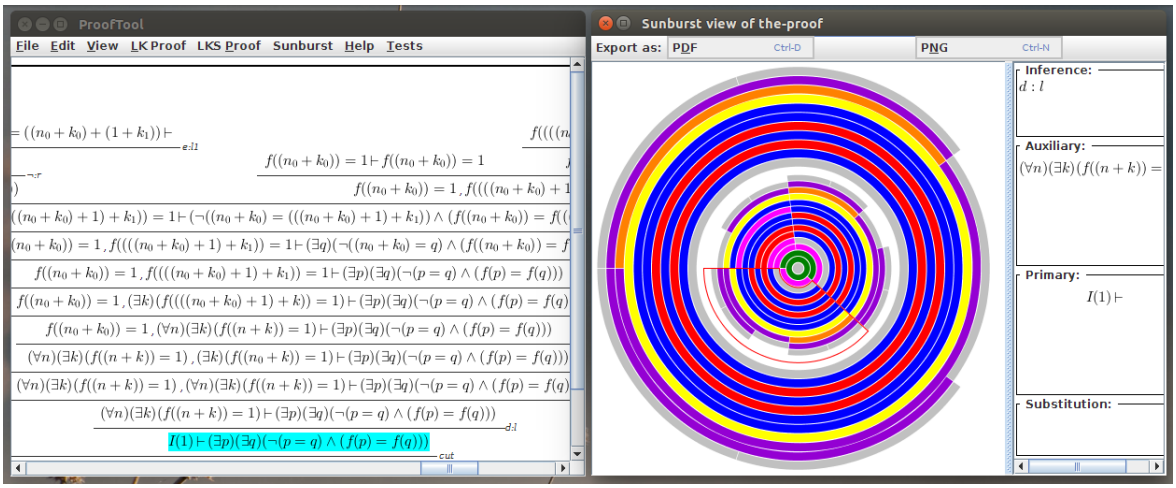

Figure 4: Synchronizing the two views.

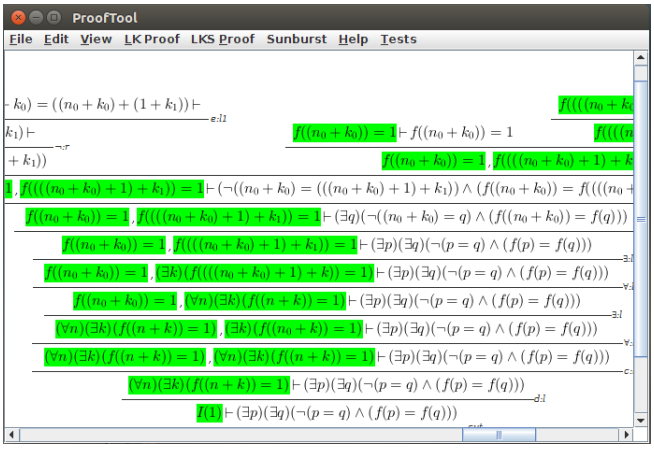

Figure 5: The cut-formula ancestors marked green.

and GAPT in general, are implemented in Scala, we have several wrappers around the classes from the TREEVIZ library. They also add functionality in the form of events which expose a newly selected node.

We created a SUNBURSTTREEDIALOG that displays the structural information as a Sunburst tree as well as the information about the selected inference. The full architecture of SUNBURSTTREEDIALOG is shown in Figure 7

SUNBURSTTREEDIALOG consists of a SPLITPANE, containing our Sunburst wrapper REACTIVE-SUNBURSTVIEW on the left (upper) side. The right (lower) side contains the inference viewer called DRAWSINGLESEQUENTINFERENCE. The orientation of the split pane is detected on run-time, depending on the dialog window size, allocating maximal space to the Sunburst view of the proof. The user can change this by either resizing the window or moving the delimiter of the pane.

DRAWSINGLESEQUENTINFERENCE extends a SCROLLPANE and uses BOXLAYOUT to display information about the selected inference, such as the inference name, auxiliary and principal formulas, and in cases of weak quantifier rules, it displays the substitution as well. In order to fit formulas best on the screen when the main window aligns objects horizontally, the DRAWSINGLESEQUENTINFERENCE

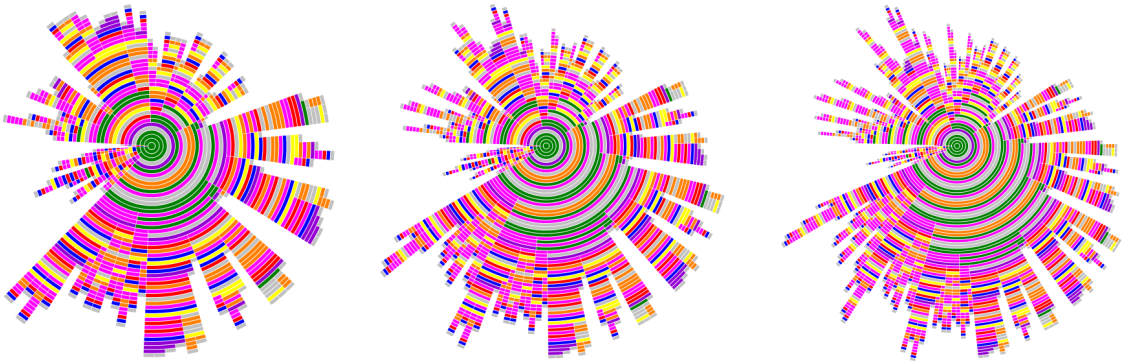

Figure 6: Instances 1, 2 and 3 of the formalization of Fürstenberg's proof of the infinity of primes.

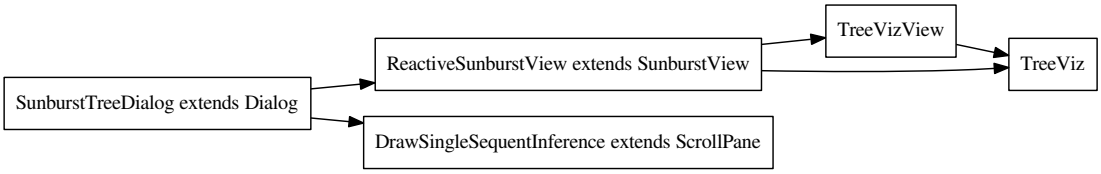

Figure 7: Architecture of SUNBURSTTREEDIALOG

layout is changing the orientation from vertical to horizontal and vice versa, depending on the size of the auxiliary and principal formulas. This means that shorter formulas are aligned side-by-side and longer ones on top of each other.

6 Conclusion and Future Work

In this paper we have explained the issues that standard tree viewers have when faced with large proofs. We have then identified various criteria for a suitable tree visualization and analyzed the available algorithms with respect to them. Our results show that Sunburst Trees seem to be the most adequate structural layout for viewing sequent calculus proofs. The global structure can be better seen than in standard layouts, which makes large proofs readable. We found identifying inferences, navigation, and tracing derivations superior to the Gentzen layout. At the same time, formula or context intensive tasks such as identifying the ancestor relationship are better left to the latter. The integration of the Sunburst viewer alongside the Gentzen viewer in PROOFTOOL demonstrate how well these two complementary layouts interact with each other.

Some improvements are still of interest to us. Foremost, multiple Sunburst trees can be represented by a forest structure. We plan to take advantage of this in two ways. First, larger proofs usually consist of several subproofs solving partial problems. In other words, they can be represented as a forest with links [9] to their subproofs. This division is often explicitly contained in the proof input language[4], but the proof object itself usually does not carry on this information. If all proof transformations are adjusted to carry on the link structure, the result can be divided into a set of proofs, making the meta-structure of

[4]The proof languages hlk, shlk and llk defined in the context of GAPT can be seen as examples for this.

the proof visible. Moreover, it is also possible to display DAGs by converting them into forests. This would enable the viewer to display resolution refutations. The high reuse of clauses in a refutation might fill the forest with many tiny trees, but this is open to experimentation.

A practical improvement is the addition of viewing profiles. By setting different color schemes and inference width ratios for each profile, we can customize the viewer to better display different aspects of proofs, like subproof complexity and instantiations.

The last two planned improvements are on the level of formulas. First, we might increase the readability of large formulas by replacing them with new symbols. In addition, we would like to improve the search facilities in PROOFTOOL. Right now, searching for a specific formula in the Gentzen view mark all occurrences of the formula. We plan to add a similar facility to the Sunburst view. One idea is to put the search results into a list in a new window, thus allowing the user to browse through the search results and jump to the right inference.

References

[1] David Aspinall, Christoph Lüth & Daniel Winterstein (2007): *A Framework for Interactive Proof*. In: *Proceedings of the 14th Symposium on Towards Mechanized Mathematical Assistants: 6th International Conference*, Springer-Verlag, Berlin, Heidelberg, pp. 161–175.

[2] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter & Hendrik Spohr (2005): *Cut-Elimination: Experiments with CERES*. In Franz Baader & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) 2004*, *Lecture Notes in Computer Science* 3452, Springer, pp. 481–495.

[3] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter & Hendrik Spohr (2008): *CERES: An analysis of Fürstenberg's proof of the infinity of primes*. *Theor. Comput. Sci.* 403(2-3), pp. 160–175. Available at `http://dx.doi.org/10.1016/j.tcs.2008.02.043`.

[4] Giuseppe Di Battista, Peter Eades, Roberto Tamassia & Ioannis G Tollis (1994): *Algorithms for drawing graphs: an annotated bibliography*. *Computational Geometry* 4(5), pp. 235–282.

[5] AlanJ. Cain (2010): *Deus ex Machina and the Aesthetics of Proof*. *The Mathematical Intelligencer* 32(3), pp. 7–11, doi:10.1007/s00283-010-9141-z. Available at `http://dx.doi.org/10.1007/s00283-010-9141-z`.

[6] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Martin Riener, Mikheil Rukhaia, Daniel Weller & Bruno Woltzenlogel Paleo (2013): *PROOFTOOL: a GUI for the GAPT Framework*. In Cezary Kaliszyk & Christoph Lüth, editors: *UITP*, *EPTCS* 118, pp. 1–14. Available at `http://dx.doi.org/10.4204/EPTCS.118.1`.

[7] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Martin Riener, Mikheil Rukhaia, Daniel Weller & Bruno Woltzenlogel-Paleo (2012): *System Feature Description: Importing Refutations into the GAPT Framework*. In David Pichardie & Tjark Weber, editors: *Second International Workshop on Proof Exchange for Theorem Proving (PxTP 2012)*, *CEUR Workshop Proceedings* 878, pp. 51–57.

[8] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Martin Riener, Mikheil Rukhaia, Daniel Weller & Bruno Woltzenlogel-Paleo (2013): *ProofTool: a GUI for the GAPT Framework*. In Cezary Kaliszyk & Christoph Lüth, editors: *Proceedings 10th International Workshop On User Interfaces for Theorem Provers (UITP 2012)*, *Electronic Proceedings in Theoretical Computer Science* 118, pp. 1–14.

[9] Cvetan Dunchev, Alexander Leitsch, Mikheil Rukhaia & Daniel Weller (2013): *CERES for First-Order Schemata*. *CoRR* abs/1303.4257. Available at `http://dblp.uni-trier.de/db/journals/corr/corr1303.html#abs-1303-4257`.

[10] Godfrey Harold Hardy (1940): *A mathematician's apology*. Cambridge University Press.

[11] Stefan Hetzl, Tomer Libal, Martin Riener & Mikheil Rukhaia (2013): *Understanding Resolution Proofs through Herbrand's Theorem*. In Didier Galmiche & Dominique Larchey-Wendling, editors: *Automated*

Reasoning with Analytic Tableaux and Related Methods (Tableaux 2013), Lecture Notes in Computer Science 8123, pp. 157–171.

[12] Yozo Hida, John O. Lamping & Ramana B. Rao (2005): *Tree visualization system and method based upon a compressed half-plane model of hyperbolic geometry*. Available at `http://www.freepatentsonline.com/6901555.html`.

[13] Mao Lin Huang, Quang Vinh Nguyen, Wei Lai & Xiaodi Huang (2007): *Three-Dimensional EncCon Tree*. In Ebad Banissi, Muhammad Sarfraz & Natasha Dejdumrong, editors: *CGIV'07: Proceedings of the Computer Graphics, Imaging and Visualisation*, IEEE Computer Society, pp. 429–433, doi:10.1109/CGIV.2007.82.

[14] Gérard Huet, Gilles Kahn & Christine Paulin-Mohring (1997): *The Coq Proof Assistant : A Tutorial : Version 6.1*. Rapport de recherche RT-0204, INRIA. Projet COQ.

[15] D.E. Knuth (1971): *Optimum binary search trees*. *Acta Informatica* 1(1), pp. 14–25, doi:10.1007/BF00264289. Available at `http://dx.doi.org/10.1007/BF00264289`.

[16] John Lamping, Ramana Rao & Peter Pirolli (1995): *A focus+context technique based on hyperbolic geometry for visualizing large hierarchies*. In Irvin R. Katz, Robert Mack, Linn Marks, Mary Beth Rosson & Jakob Nielsen, editors: *CHI'95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press/Addison-Wesley Publishing Co., pp. 401–408, doi:10.1145/223904.223956.

[17] Lars Linsen & Sabine Behrendt (2011): *Linked Treemap: A 3D Treemap-nodelink layout for visualizing hierarchical structures*. *Computational Statistics* 26(4), pp. 679–697, doi:10.1007/s00180-011-0272-2.

[18] John Meier & Clifford A. Reiter (1996): *Fractal representations of Cayley graphs*. *Computers and Graphics* 20(1), pp. 163–170, doi:10.1016/0097-8493(95)00101-8.

[19] Tamara Munzner (1997): *H3: laying out large directed graphs in 3D hyperbolic space*. In John Dill & Nahum D. Gershon, editors: *InfoVis'97: Proceedings of the IEEE Symposium on Information Visualization*, IEEE Computer Society, pp. 2–10, doi:10.1109/INFVIS.1997.636718.

[20] Martin Odersky, Lex Spoon & Bill Venners (2010): *Programming in Scala: A Comprehensive Step-by-step Guide*, 2nd edition. Artima, Inc.

[21] Grace D Paterson (2013): *The Aesthetics of Mathematical Proofs*. Master's thesis.

[22] James Rosindell & Luke J. Harmon (2012): *OneZoom: A Fractal Explorer for the Tree of Life*. *PLoS Biology* 10(10), p. e1001406, doi:10.1371/journal.pbio.1001406.

[23] Adrian Rusu & Confesor Santiago (2007): *A Practical Algorithm for Planar Straight-line Grid Drawings of General Trees with Linear Area and Arbitrary Aspect Ratio*. In Ebad Banissi, Remo Aslak Burkhard, Georges Grinstein, Urska Cvek, Marjan Trutschl, Liz Stuart, Theodor G. Wyeld, Gennady Andrienko, Jason Dykes, Mikael Jern, Dennis Groth & Anna Ursyn, editors: *IV'07: Proceedings of the International Conference on Information Visualisation*, IEEE Computer Society, pp. 743–750, doi:10.1109/IV.2007.14.

[24] H. Schulz (2011): *Treevis.net: A Tree Visualization Reference*. *Computer Graphics and Applications, IEEE* 31(6), pp. 11–15, doi:10.1109/MCG.2011.103.

[25] Ben Shneiderman (1991): *Tree visualization with Tree-maps: A 2-d space-filling approach*. *ACM Transactions on Graphics* 11, pp. 92–99.

[26] Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis & Volker Sorge (1998): *A Distributed Graphical User Interface for the Interactive Proof System*. In: *Proceedings of the International Workshop "User Interfaces for Theorem Provers 1998 (UITP'98)*, Eindhoven, Netherlands, pp. 130–138. Available at `http://christoph-benzmueller.de/papers/W1.pdf`.

[27] John Stasko & Eugene Zhang (2000): *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations*. In Jock D. Mackinlay, Steven F. Roth & Daniel A. Keim, editors: *InfoVis'00: Proceedings of the IEEE Symposium on Information Visualization*, IEEE Computer Society, pp. 57–65, doi:10.1109/INFVIS.2000.885091.

[28] G. Sutcliffe (2009): *The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0*. Journal of Automated Reasoning 43(4), pp. 337–362.

[29] Steven Trac, Yury Puzis & Geoff Sutcliffe (2007): *An Interactive Derivation Viewer*. Electronic Notes in Theoretical Computer Science 174(2), pp. 109 – 123, doi:http://dx.doi.org/10.1016/j.entcs.2006.09.025. Available at `http://www.sciencedirect.com/science/article/pii/S1571066107001739`. Proceedings of the 7th Workshop on User Interfaces for Theorem Provers (UITP 2006).

[30] Charles Wetherell & Alfred Shannon (1979): *Tidy Drawings of Trees*. IEEE Transactions on Software Engineering SE-5(5), pp. 514–520, doi:10.1109/TSE.1979.234212.

[31] W. Windsteiger (2012): *Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System*. In Cezary Kaliszyk & Christoph Lueth, editors: *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, Bremen, Germany, July 11th 2012*, Electronic Proceedings in Theoretical Computer Science 118, Open Publishing Association, pp. 72–82. Available at `http://arxiv.org/abs/1307.1945v1`. Doi 10.4204/EPTCS.118.5.

[32] Jing Yang, Matthew O. Ward & Elke A. Rundensteiner (2002): *InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures*. In Pak Chung Wong & Keith Andrews, editors: *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, IEEE Computer Society, pp. 77–84, doi:10.1109/INFVIS.2002.1173151.

[33] Hee Yong Youn & Adit D. Singh (1988): *Near optimal embedding of binary tree architecture in VLSI*. In: *ICDCS'88: Proceedings of the International Conference on Distributed Computing Systems*, IEEE Computer Society, pp. 86–93, doi:10.1109/DCS.1988.12503.

Appendix A: How to use the Tool

All the PROOFTOOL snapshots shown in this paper are taken using actual proofs. In this appendix we will describe how to install the tool and display the proofs.

The GAPT framework provides two kinds of interfaces, a commands based shell prompt and the graphical PROOFTOOL. The shell prompt is based on the Scala shell, so its usage is more comfortable if one needs to do inline programming apart from predefined functions. To visualize generated objects it calls the PROOFTOOL, which allows for both the presentation and manipulation of proofs and supports a subset of the functionality of GAPT.

The two executables, bundled together with an examples directory, can be downloaded at `http://www.logic.at/gapt/downloads/gapt-1.8.zip`. The examples directory contains, among other things, also the proofs used in this paper.

In this paper we used different versions of two proofs. The first proof, called the Tape proof, proves that if there is an infinite tape filled-in with two symbols, then there are at least two cells with the same symbol [2]. The second proof is the formalization of Fürstenberg's proof of the infinity of primes [3].

In Figures 1 and 2 a higher-order version of the Tape proof is shown. In order to load it from the shell prompt, first run the shell by executing `./cli.sh` and then load the proof using the following commands:

```
scala> val p = loadLLK("./examples/tape/ntape.llk")
scala> val elp = regularize(eliminateDefinitions(p, "TAPEPROOF"))._1
scala> val selp = lkTolksk(elp)
scala> PT.display("TAPEPROOF",selp)
```

Figure 1 is then obtained by changing the font size in the View menu to minimum and hiding all the formulas (from the Edit menu). Figure 2 is obtained by calling "Sunburst view" form the Sunburst menu and selecting the inferences either by directly clicking on them or by navigating to it via the keyboard.

The rest of the proofs can be loaded directly from PROOFTOOL. To load the PROOFTOOL directly, execute `./gui.sh`.

The first-order Tape proof is shown in Figures 3, 4 and 5. From the File menu, please open the `./examples/tape/tape.xml.gz` file. Open the Sunburst view and place the two windows side-by-side in order to see the connection between them. Start navigating inside the Sunburst view from the keyboard and the other window will change its view as shown in Figure 4. Call the Edit>Mark Cut-Ancestors menu item to get the effect shown in Figure 5.

The Fürstenberg's proof is formalized as a proof schema [9] and instances for 1, 2 and 3 primes are extracted into separate files. Using again the File menu, the three proofs can be loaded using the following files:

```
./examples/prime/ceres_xml/prime1-1.xml.gz
./examples/prime/ceres_xml/prime1-2.xml.gz
./examples/prime/ceres_xml/prime1-3.xml.gz
```

and then calling the "Sunburst view" in order to get the views shown in Figure 6.