ProofTool: GUI for the GAPT Framework[*]

Cvetan Dunchev Alexander Leitsch Tomer Libal Martin Riener
Mikheil Rukhaia Daniel Weller[†]
Bruno Woltzenlogel-Paleo

{cdunchev,leitsch,shaolin,riener,mrukhaia,weller,bruno}@logic.at

Institute of Computer Languages (E185)
[†]Institute of Discrete Mathematics and Geometry (E104)
Vienna University of Technology

Abstract

This paper describes the implementation, as well as the features, of the graphical user interface, more specifically defined as a proof viewer, for the General Architecture for Proof Theory (GAPT) framework. It contains methods to render classical and schematic sequent calculus proofs as well as resolution proofs and other tree-like structures in a flexible way. Additional emphasis is put on the schematic proof input format which should be as user-friendly as possible for the end-user.

1 Introduction

GAPT[1] (General Architecture for Proof Theory) is a framework that aims at providing data-structures, algorithms and user interfaces for analyzing and transforming formal proofs. GAPT was conceived to replace and expand the scope of the CERES system[2] beyond the original focus on cut-elimination by resolution for first-order logic [BL00]. Through a more flexible implementation, which is based on the basic data structures for simply-typed lambda calculus, for sequent and resolution proofs, and was written in the hybrid functional object-oriented language Scala [OSV10], GAPT has already allowed the implementation of the generalization of the CERES method (cut-elimination by resolution) to proofs in higher-order logic [HLW11] and to schematic proofs [DLRW12]. Furthermore, methods for structuring and compressing proofs, such as cut-introduction [HLW12] and Herbrand Sequent Extraction [HLWWP08] are being implemented.

The GAPT system has a *command line interface* (CLI) that allows a user to access the capabilities of the system, e.g. to create and manipulate proofs. But such an interface is not suited for the visualization of proofs; when viewing proofs, which are large trees or DAGs, more sophisticated user interaction (scrolling, viewing/hiding of information), which cannot provided by a CLI, is required. Hence a proof viewer called ProofTool, was implemented to allow a more sophisticated visualization.

While it is possible to call a ProofTool frame from the CLI to visualize a particular piece of data, the aim is to extend ProofTool to become a stand-alone *graphical user interface* for GAPT. For the time being, ProofTool gives access to most of GAPT's features. It is constantly being extended towards becoming a fully fledged GUI for the GAPT system.

The rest of the paper is organized as follows: Section 2 describes the GAPT system in more detail and gives some basic understanding of the terms used in the paper. In Section 3 a language for the input of schematic proofs is presented and in Sections 4 and 5 ProofTool is described in detail. The final section discusses some future implementations and improvements.

[*]Partially supported by the project I383 of the Austrian Science Fund.
[1]GAPT homepage: `http://code.google.com/p/gapt/`
[2]CERES homepage: `http://www.logic.at/ceres/`

2 Preliminaries

In this section some basic terms which will be used through this paper are presented. We start with the CERES system and a comparison of ProofTool with its old prototype (both programs are called "prooftool", but to avoid confusion the old one is written with a verbatim font).

2.1 The CERES System

GAPT was conceived to replace and expand the scope of the CERES system, which consists of the programs `ceres`, `hlk` and `prooftool`. The CERES system uses Prover9 [McC10] to refute clause sets (which is a vital step of the CERES method).

CERES is a cut-elimination method by resolution. Its basic data structures are: *struct*, *characteristic clause set* and *projection set*. The struct is a clause term obtained from a proof with cuts and the characteristic clause set is a clause set obtained by evaluation of a struct. Projections are cut-free parts of a proof obtained by skipping all inferences going into cuts. In this paper we refer to these objects as *the data-structures of the CERES method*.

The system works in the following way: First the user enters a formal proof into the CERES system using the Handy LK[3] language. Then the program `hlk` compiles this formalization to an XML file which is used as input for the `ceres` program. In the next step `ceres` is used to perform the following steps:

- skolemize the proof
- compute the characteristic clause set
- call an external first-order resolution prover and compute a resolution refutation of the characteristic clause set
- compute the proof projections and generate the output (original proof, Atomic Cut Normal Form (ACNF), characteristic clause set, Herbrand sequent, etc.)

The output of `ceres` is again its input XML format. Now the program `prooftool` is used to visualize the formal proofs obtained by `ceres`. In the following we will call the XML format used during these steps as *ceres xml* format.

ProofTool has several advantages over the old prototype. First, it can display general tree-like structures such as the struct with varying graphical outputs. In contrast to that `prooftool` can only display sequent calculus proofs, so for instance a list of sequents is shown as a proof with unary inferences which is quite counter-intuitive.

Second, ProofTool is flexible enough to handle proof schemata as objects. This is necessary since the output of the CERES method is again a proof, which means, if a proof schema is transformed not only an instance should be displayed but the schema itself.

Third, it supports more input and output formats than the old prototype. A full comparison is given in Figure 1.

2.2 Proof Schemata

Formula schemata were introduced and investigated in [ACP09, ACP11]. A subclass called *regular schemata* was identified and shown decidable, and a tableau calculus STAB was defined and implemented [ACP10]. `RegSTAB` is a STAB prover that refutes regular formula schemata. It outputs the tableau refutation in an XML file, which can be read by ProofTool.

By proof schemata we mean schematic sequent calculus proofs. In this calculus sequents are multisets of formula schemata and proofs are defined in a recursive way. The rules are

[3]Handy LK syntax: `http://www.logic.at/hlk/handy-syntax.pdf`

Features	ProofTool	`prooftool`
Zooming, scrolling	Yes	Yes
Display first- and higher-order proofs	Yes	Yes
Display sequent lists	Yes	Yes
Parse ceres XML/export in .tex	Yes	Yes
Export in .tptp	Yes	No
Proof schemata and related features	Yes	No
Trees and related features	Yes	No
Display definition lists	Yes	No
Marking cut-ancestors, extracting cut-formulas	Yes	No
Hiding sequent context	Yes	No
Search	Yes	No
Hiding structural rules	Yes	No
Split/unsplit, Substitute/unsubstitute	No	Yes

Figure 1: ProofTool vs `prooftool`.

similar to those of a usual sequent calculus but they operate on formula schemata. As a special axiom rule, proof links connect recursive instances of other proofs. A detailed description of schematic sequent calculus can be found in [DLRW12].

Since the CERES method for proof schemata is currently under development, the GAPT system is used for practical investigations. Therefore, it contains an implementation of the basic data-structures and algorithms for the schematic CERES method such as the *schematic struct*, the *schematic characteristic clause set*, the *schematic projection term* and their extraction from proof schemata.

2.3 The GAPT System

GAPT is a framework that aims at providing data-structures, algorithms and user interfaces for analyzing and transforming formal proofs. Although it is a quite complex system, here we focus on features that are integrated in ProofTool. We start our discussion from parsers and exporters.

There are several input formats parsed by the GAPT system but the most important ones are the (schematic) *proof input language*, the *ceres xml* format and a less restrictive, so called *simple xml* format. The proof input language will be discussed in the next section. Simple xml is a simplified version of ceres xml (i.e. the Document Type Definition (.dtd) file has a much simpler structure and less restrictions) and is general enough to communicate with other systems. While ceres xml requires that the proof is a sequent calculus proof, there is no such restriction in the simple xml format. Therefore, it can be used for exchanging any tree-like proofs with other systems, as was done with `RegSTAB`.

The current output format slightly differs from the *ceres xml* format. The reason is that the *ceres xml* format is tailored to a sequent calculus with permutation rules and the GAPT system represents sequents as multisets instead. Therefore at the moment the output XML can not be used as input. Since the modification of the *ceres xml* parser to accommodate this drawback is quite straightforward, we expect to add this feature in the near future.

Most of the other features of GAPT are related to the CERES method. It provides data structures for (schematic) first-order and higher-order formulas as well as different sequent and resolution calculi. Many algorithms like regularization, skolemization, matching and various unification algorithms are contained. Their use is shared with the interactive theorem prover TAP which is also part of the system. Other parts of the system are the libraries connecting external theorem provers like Vampire and Prover9. The computation of the struct, the characteristic clause set, and the proof projections now form the heart of GAPT. It is also accompanied by methods to extract the Herbrand Sequent and serves as a test-bed for cut-introduction methods currently under development.

3 Proof Input Language

Since proofs are the main input for the GAPT system, a comfortable proof input language is important. Unfortunately the Handy LK language has several shortcomings. The first and major one is that a user cannot input a given formal LK proof as it is, because originally Handy LK was designed for interactive proof search (e.g. it is not possible to specify structural rules except cuts, they are inferred by the `hlk` program and this may destroy intended structure of the proof). Second, it can not directly treat proof schemata. Although it allows proofs to be defined recursively, `hlk` can output only user specified instances of it. But transformations of schematic proofs require that the schema is kept throughout the whole process since the output format is again a schematic proof. These drawbacks supported the definition of a new language which we call *proof input language*.

The proof input language is designed to define (schematic) proofs in a form which is both easily human and machine readable. It is simple enough that an input proof can be written in any text editor. To differentiate it from other text files, the extension *.lks* is used. The full description of the grammar of the language as well as of the formal calculus, can be found in [DLRW12].

One *.lks* file must contain at least one proof definition, which has the pattern given in Figure 2. For an inductive proof definition, the `base` block describes the base case and the `step` block describes the recursive case. The *id*s are arbitrary labels that are unique within the scope of { . . . } blocks (i.e. the same labels can be used in the definition of base and step cases) and rules are tuples consisting of the rule's name, the ids of the premises and of the auxiliary formulas.

proof *name* proves *sequent*
base {
  $id_1 : rule_1$
    . . .
  $id_n : rule_n$
  root : $rule_{n+1}$
}
step {
  $id_1 : rule_1$
    . . .
  $id_m : rule_m$
  root : $rule_{m+1}$
}

Figure 2: Proof syntax of *proof input language*.

As the language would profit from syntax highlighting, having an editor for this language would be convenient. One solution is to use XText[4] and create such an editor using the grammar of the *proof input language*. An advantage of the grammar is that it is easy to give the exact line numbers where the parsing of a file fails because of a syntactic error.

4 ProofTool From a User Perspective

ProofTool is a graphical user interface, used to display objects generated by the GAPT system. These objects are: trees, proofs, sequents, formulas, etc. Below we describe features of ProofTool related to these objects.

4.1 Input/Output

ProofTool supports different kinds of parsers, one for *.lks* files and two for different XML formats, *ceres xml* and *simple xml* (possibly in gzipped form). As we saw in the previous sections *.lks* files are used to input schematic proofs. The ceres xml format is used for proofs in first- and higher-order sequent calculus. Simple xml format is a general format, where one can input arbitrary tree-like proofs (e.g. natural deduction, tableaux, etc). So far it is used to communicate with `RegSTAB`. There are disadvantages of using the *simple xml* format: since it is a very general format, structural details about the proof are absent. For instance the leaves of a proof will be displayed simply as the strings which occur in the XML file instead of an easier readable LaTeX rendering. This also means some advanced features of ProofTool might not work. Basic features zooming and scrolling are not affected but certain views might not be available. To open an input file, users should use the corresponding menu item of the File menu.

In ProofTool there are several exporters for objects of the GAPT system. One of them is the already mentioned *ceres xml exporter*, which can save proofs, sequent lists and definition lists in an *.xml* file. This is done from the menu items *save proof as XML* (which saves currently displayed proof) and *save all as XML* (which saves the whole database) of the File menu. There are also several sequent list exporters, which will be discussed later in this section.

4.2 Trees

Trees are binary trees in the system and they are displayed upside-down (see Figure 3). The user can manipulate the size of the tree by hiding/showing some branches or leaves. This is done by calling the corresponding menu items of the Edit menu, or by clicking on the vertex that should be hidden/shown.

For tree-like objects like the clause term or the projection term, the system contains transformations to trees. A simple example is a directed acyclic graph where the corresponding tree just has duplicates of the shared structures.

4.3 TreeProofs

A TreeProof is also a binary tree in the GAPT system which represents a tree-like proof. It is a Tree which is also a Proof. This means that we can expect that nodes are labeled with sequents. The reasons for the decision to display TreeProofs instead of Proofs are as follows: A Proof is only a directed acyclic graph, so additional arrows for shared structures would be necessary. Apart from sequent calculus proofs, at the moment only resolution proofs need to be displayed; The tree representations of these proofs can easily be obtained from the DAG-form.

[4]XText homepage: `http://www.eclipse.org/Xtext/`

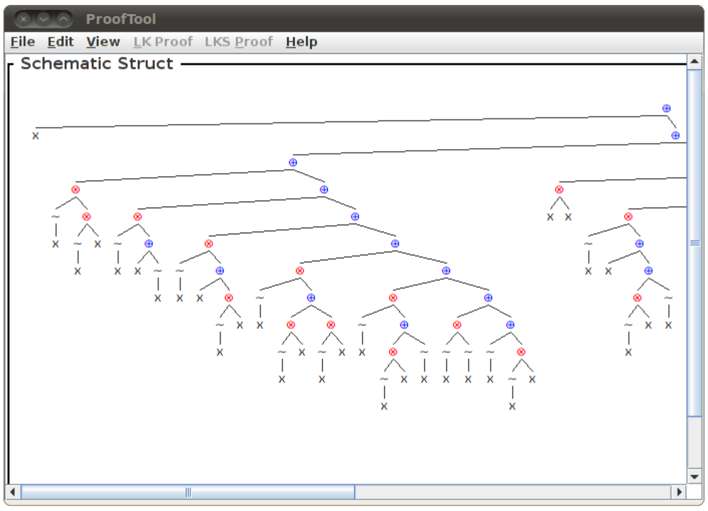
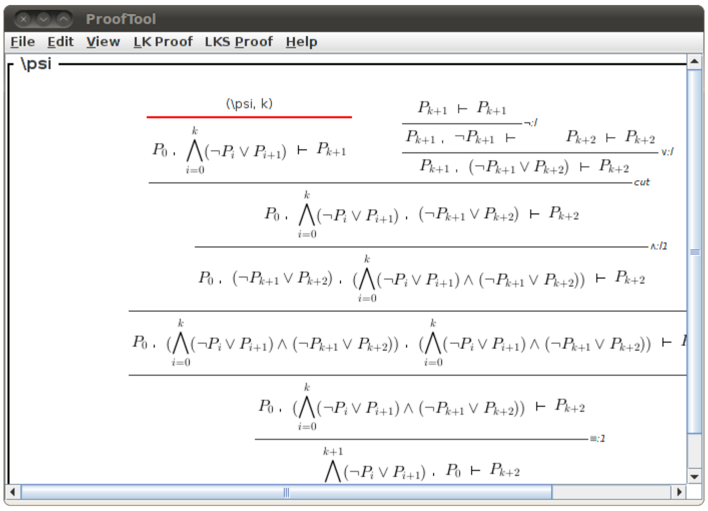


Figure 3: Tree in ProofTool (leaves are hidden).

In general, proofs in the GAPT system are sequent-like proofs. For an example of a proof displayed using ProofTool see Figure 4.



$$\dfrac{P_{k+1} \vdash P_{k+1}}{P_{k+1} , \neg P_{k+1} \vdash}\ \neg{:}l \qquad P_{k+2} \vdash P_{k+2}$$

$$\dfrac{P_{k+1} , \neg P_{k+1} \vdash \qquad P_{k+2} \vdash P_{k+2}}{P_{k+1} , (\neg P_{k+1} \vee P_{k+2}) \vdash P_{k+2}}\ \vee{:}l$$

$$\dfrac{P_0 , \bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \vdash P_{k+1} \qquad P_{k+1} , (\neg P_{k+1} \vee P_{k+2}) \vdash P_{k+2}}{P_0 , \bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) , (\neg P_{k+1} \vee P_{k+2}) \vdash P_{k+2}}\ cut$$

$$\dfrac{P_0 , \bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) , (\neg P_{k+1} \vee P_{k+2}) \vdash P_{k+2}}{P_0 , (\neg P_{k+1} \vee P_{k+2}) , (\bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \wedge (\neg P_{k+1} \vee P_{k+2})) \vdash P_{k+2}}\ \wedge{:}l1$$

$$\dfrac{P_0 , (\neg P_{k+1} \vee P_{k+2}) , (\bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \wedge (\neg P_{k+1} \vee P_{k+2})) \vdash P_{k+2}}{P_0 , (\bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \wedge (\neg P_{k+1} \vee P_{k+2})) , (\bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \wedge (\neg P_{k+1} \vee P_{k+2})) \vdash P}$$

$$\dfrac{P_0 , (\bigwedge_{i=0}^{k}(\neg P_i \vee P_{i+1}) \wedge (\neg P_{k+1} \vee P_{k+2})) \vdash P_{k+2}}{\bigwedge_{i=0}^{k+1}(\neg P_i \vee P_{i+1}) , P_0 \vdash P_{k+2}}\ \equiv{:}1$$

Figure 4: Proof in ProofTool.

There are two kinds of sequent-like proofs in GAPT: first- and higher-order sequent calculus (LK) proofs and schematic sequent calculus (LKS) proofs. In ProofTool there are the menus "LK Proof" and "LKS Proof" containing the possible operations for these proofs respectively. The available menu items are the following:

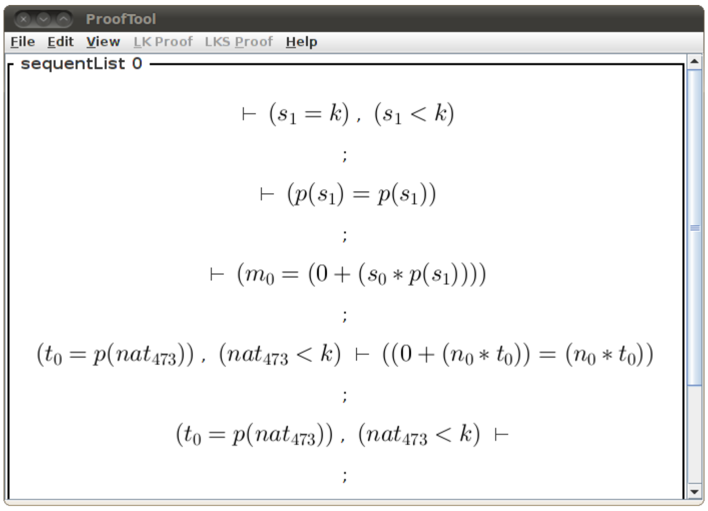
- From the LK Proof menu:
  - Compute the data-structures of the CERES method, such as struct and characteristic clause set.

  - Apply reductive cut-elimination (Gentzen's) method.
- From the LKS Proof menu:
  - Compute the data-structures of the Schematic CERES method, such as schematic struct, schematic characteristic clause set and schematic projection term.
  - Compute the instance of a schematic proof for the number given by the user.
- From the Edit menu:
  - Hide sequent context: for each rule in the proof show only the auxiliary formulas used in the rule.
  - Mark cut-ancestors: highlight all ancestors of all cut-formulas occurring in the proof.
  - Extract cut-formulas: extract list of all cut-formulas.

The menus currently supports only a subset of the capabilities of the GAPT system. The other functionalities will be added to ProofTool on a by-need basis.

4.4 Lists

Lists are very important data-structures and it is worthy to have specialized handling for them. In ProofTool each element of a list is displayed in a single line. Lines are separated with semicolons. The most commonly occurring lists in the GAPT system are *sequent lists* and *definition lists*. For an example of a sequent displayed using ProofTool see Figure 5.



$\vdash (s_1 = k)\ ,\ (s_1 < k)$

;

$\vdash (p(s_1) = p(s_1))$

;

$\vdash (m_0 = (0 + (s_0 * p(s_1))))$

;

$(t_0 = p(nat_{473}))\ ,\ (nat_{473} < k)\ \vdash ((0 + (n_0 * t_0)) = (n_0 * t_0))$

;

$(t_0 = p(nat_{473}))\ ,\ (nat_{473} < k)\ \vdash$

;

Figure 5: List in ProofTool.

In the GAPT system we have several exporters for sequent lists and ProofTool allows exportation in two different formats, either in *.tex* or *.tptp* files. This is done by calling the corresponding menu items of the File menu.

4.5 Features Under Development

In this subsection two useful features are discussed which are currently under development. The first one is search and the second one is the hiding of structural rules in a proof.

Searching is very useful when one has to deal with huge proofs or other objects, which is often the case in our research. So we decided to have a feature that will allow us to search in objects displayed by ProofTool. Currently, it is done in the following way: a user calls the Search dialog from the Edit menu and types (possibly LaTeX) a string representation of the object he/she wants to search. Then the string is searched for in the string representation of the displayed object and, if found, the corresponding part is highlighted.

A problem occurring during search is that the string representation of a node might be quite different from its rendering. Let us consider the search in the proof displayed in Figure 4. After typing `cut` in the search dialog, ProofTool will find the rule name and color it green. If one searches for the formula $\neg P_{k+1}$, then exactly the LaTeX representation `\neg␣P_{k+1}` needs to be entered. Counter-intuitively, the search for `\neg␣␣P_{k+1}` will fail even if the rendered formula looks the same. Some remedies for the problems are already under development but there is no straightforward solution maintaining the flexibility of the input formats.

Hiding structural rules is very useful to shorten the size of proofs. In most of the cases structural rules do not contain any valuable information and they can be hidden for the user. The Edit>Hide Structural Rules menu item allows access to this feature, although it sometimes hides too few inferences. Investigation into this matter is still in progress.

5 Implementation Details

ProofTool is implemented using the scala.swing library [Mai09]. It is a SimpleSwingApplication and consists of one frame, which contains a MenuBar and a ScrollPane. We continue with a brief description of the architecture of ProofTool, shown in Figure 6.

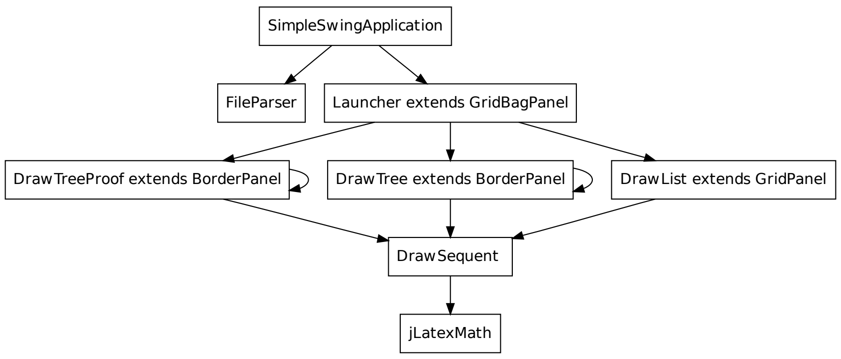

SimpleSwingApplication
FileParser
Launcher extends GridBagPanel
DrawTreeProof extends BorderPanel
DrawTree extends BorderPanel
DrawList extends GridPanel
DrawSequent
jLatexMath

Figure 6: Architecture of ProofTool.

ScrollPane has a component called Launcher which extends GridBagPanel and takes the following parameters:

- A pair (String, AnyRef), where AnyRef is an object that should be displayed and String is a name of the object. The object has TitledBorder around it and the name is the title.
- An Int, which is a size of font that is used to display an object.

Thanks to such a general design, the basic feature, zooming, can be applied to all objects easily (by calling the *zoom in* and *zoom out* menu items from the View menu).

When an object is passed to Launcher it uses Scala's matching mechanism to recognize its structure and instantiates the corresponding class responsible for the drawing of the object. Basically, Launcher differentiates between three kinds of objects: trees, proofs and lists. The rendering classes are DrawTree, DrawProof and DrawList respectively.

DrawTree extends the BorderPanel class and displays a tree in the following way: A leaf node just renders the vertex, whereas an inner node draws the root node at the top and then creates new DrawTree instances for the child trees. For a binary node, the branches of the tree are rendered side-by-side on a frame and, for a unary node, the branch is rendered in the center. The root is connected to its children by straight lines.

DrawProof also extends the BorderPanel class and behaves similar to DrawTree. The difference is that the desired output looks like a sequent calculus proof. This means that while DrawTree puts the vertex at the top and the branches below, DrawProof puts the vertex at the bottom and the branches on top of it. Then it draws horizontal lines between vertices and puts the rule names next to the lines.

DrawList extends the GridPanel class having only one column and puts each element of the list in a separate cell.

In ProofTool a sequent is displayed by a FlowPanel which contains the representation of each formula as a separate Label.

To handle formulas it was decided to use jLatexMath[5] . It is a Java API which displays mathematical formulas written in LaTeX as images. It was used in the GUI of Scilab[6]. Drawback of this library is that having an image for every formula is quite memory consuming.

For each formula displayed a label is created. Then the formula is transformed into a LaTeX string and rendered using jLatexMath. The resulting image is assigned as an icon to the label. Since this rendering is expensive, we use it only when necessary and display simple string representations otherwise. For example, any vertex containing a higher-order expression of type HOLExpression (which also includes first-order and schema formulas) is rendered by jLatexMath. Other types of vertices are displayed simply as their Scala string representation which often suffices since the Unicode character-set used includes Greek letters as well as other logical symbols.

ProofTool strongly profits from the scala.swing wrapper library which simplifies event handling significantly. Since there are only Publisher and Reactor elements instead of Java's complicated event handling mechanism, for instance menu items only need to listen to the ProofToolPublisher to adjust their activation status accordingly. In the case a new file is loaded or the proof database is dynamically changed the ProofDbChanged event is issued. Since also the View>View Proof, View>View Clause List and View>View Term Tree menus listen to this event, they can refresh their list of MenuItems.

6 Related and Future Work

This paper presented a graphical user interface for the GAPT framework. It is able to display various new and legacy proof formats, especially when they are tailored towards humans. ProofTool is also flexible enough to render sequent calculus and resolution proofs as well as other tree-like structures in appealing ways and acts as a graphical shell to the functionality of GAPT.

Of course there are related works to be mentioned, for example LOUI [SHB$^+$99], IDV [TPS06] and the like. These viewers are quite developed, but, for our applications, they have several shortcomings. The major one is that they display only DAGs and cannot

[5]jLatexMath homepage: `http://forge.scilab.org/index.php/p/jlatexmath/`
[6]Scilab homepage: `http://www.scilab.org/`

handle sequent-like tree proofs. Two other shortcomings are sole support of TPTP as input format and the rendering of formulas only in TPTP-representation.

Still there are numerous things on the agenda. The proof input language will be extended to first-order schematic proofs. Also the input language should unify parsing of schematic and classical sequent calculus proofs. It is planned to change the parser such that it automatically inserts weakening and contraction rules left out by the user, which can ease the proof specification process. Finally, an auto-propositional feature should be integrated in the parser, that will prove propositional sequents automatically if the user does not want to give the exact proof of that part.

Regarding ProofTool, more commands already available in the *command line interface* (e.g. skolemization, regularization, Herbrand sequent extraction, theorem prover integration, etc.) will be added to the GUI. Also efficiency issues will be tackled since at the moment 20MB of stack and 2GB of heap memory barely suffice to run GAPT's algorithms on formalizations of real mathematical proofs like Fürstenberg's proof of the infinity of primes in [BHL+08] or schematic proof of an n-bit adder is commutative (see [DLRW12]).

Finally a user manual and an on-line help system are on the agenda.

References

[ACP09] Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier. A schemata calculus for propositional logic. In *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 5607 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2009.

[ACP10] Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier. RegSTAB: A SAT-Solver for Propositional Iterated Schemata. In *International Joint Conference on Automated Reasoning*, pages 309–315, 2010.

[ACP11] Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier. Decidability and undecidability results for propositional schemata. *Journal of Artificial Intelligence Research*, 40:599–656, 2011.

[BHL$^+$08] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. CERES: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403:160–175, 2008.

[BL00] Matthias Baaz and Alexander Leitsch. Cut-elimination and redundancy-elimination by resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

[DLRW12] C. Dunchev, A. Leitsch, M. Rukhaia, and D. Weller. About Schemata And Proofs web page, 2010–2012. `http://www.logic.at/asap`.

[HLW11] Stefan Hetzl, Alexander Leitsch, and Daniel Weller. CERES in higher-order logic. *Annals of Pure and Applied Logic*, 162(12):1001–1034, 2011.

[HLW12] Stefan Hetzl, Alexander Leitsch, and Daniel Weller. Towards algorithmic cut-introduction. In *LPAR*, pages 228–242, 2012.

[HLWWP08] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Herbrand sequent extraction. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics*, volume 5144 of *Lecture Notes in Computer Science*, pages 462–477. Springer Berlin, 2008.

[Mai09] Ingo Maier. The scala.swing package. `http://www.scala-lang.org/sid/8`, 2009.

[McC10] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[OSV10] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide.* Artima, Inc., 2nd edition, 2010.

[SHB$^+$99] Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. LOUI: Lovely OMEGA User Interface. In *Formal Aspects of Computing*, pages 326–342, 1999.

[TPS06] Steven Trac, Yury Puzis, and Geoff Sutcliffe. An Interactive Derivation Viewer. In *Proceedings of the 7th Workshop on Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning, volume 174 of Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.