Traf: A Graphical Proof Tree Viewer Cooperating with Coq Through Proof General

Hideyuki Kawabata(✉), Yuta Tanaka, Mai Kimura, and Tetsuo Hironaka

Hiroshima City University,
3-4-1 Ozuka-higashi, Asa-minami, Hiroshima 731-3194, Japan
kawabata@hiroshima-cu.ac.jp

Abstract. Traf is a graphical proof tree viewer that cooperates with the Coq proof assistant and is controlled through Proof General. Among other proof tree viewers and tools for browsing proof scripts, Traf is well suited for daily proving of Coq problems as it is easy to use, non-disturbing, and helpful. Proof trees dynamically updated by Traf during interactive sessions with Proof General are informative and as readable as Gentzen-style natural deduction proofs. Traf facilitates browsing and investigating tactic-based proof scripts, which are often burdensome to read. Traf can also be used for typesetting proof trees with LaTeX. The current version of Traf was developed as an extension to the Prooftree proof tree viewer and makes use of many of its facilities. Traf provides functionalities that are useful to both novice Coq users and experienced Proof General users.

Keywords: Proof tree viewer · Interactive theorem prover · Coq
Proof General · Readability of proof scripts

1 Introduction

Proof assistants are widely used for proving mathematical theorems [14,15] and properties of software [1,4] and for developing dependable software [22]. The power of mechanized verification by using proof assistants has been accepted, and such verification is now thought to be indispensable. Therefore, the readability and maintainability of proof scripts have become major concerns [10].

Among the many proof assistants [26], there are two major styles for writing proof scripts; the tactic-based style and the declarative style [17,25]. Although the former is preferable for writing concise proofs interactively by making use of the theorem prover's automation facilities, it is burdensome to read the proof scripts. Conversely, although proof scripts written in the latter style are informative and readable without tools, writing intermediate formulae could be laborious. To alleviate this situation, several tactic-based systems have been extended to accept declarative proofs [8,13,25], and several systems offer a facility for rendering tactic-based proof scripts in a pseudo-natural language [5,9,12].

S. Ryu (Ed.): APLAS 2018, LNCS 11275, pp. 157–165, 2018.
https://doi.org/10.1007/978-3-030-02768-1_9

Since a proof is not usually in a single-threaded structure, visualizing proofs in graphical representations could be an effective complementary approach for improving the readability of proof scripts. There have been many studies on graphical representations of proofs; IDV [24] can graphically render derivations at various levels of granularity. ProofWeb [18] uses the Coq proof assistant with specialized tactics to help the user learn Gentzen-style natural deduction proofs. ProofTool [11] offers a generic framework for visualizing proofs and is equipped with a method for visualizing large-scale proofs as Sunburst Trees [19]. ViPrS is an interactive visualization tool for large natural deduction proof trees [7]. Mikiβ [20] offers a set of APIs for constructing one's own proof checker with facilities for building proof trees by using a GUI. Pcoq [3] had a GUI for proving lemmas by using a mouse, but it is no longer available. The Prooftree proof tree viewer [23] dynamically draws a proof tree while the user interacts with Coq through Proof General, although the shape of the tree is rather abstract.

In this paper, we present a graphical tool called *Traf* that constructs proof trees automatically while the user is interacting with Coq through Proof General. Traf is different from ordinary proof viewers and proof translators in that it is designed to guide interactive theorem proving by using a full-fledged proof assistant through a standard tactic-based interface. In other words, Traf is a helper tool for enhancing both the writability and readability of proofs. The proof tree shown in Traf's window looks like a readable Gentzen-style natural deduction proof. The user does not have to worry about operating Traf since the tree dynamically grows as the proving process proceeds. Traf reorganizes the layout of the tree adaptively in accordance with changes in the proof structure caused by modifications to the proof script. It can automatically shrink unfocused branches, enabling the user to concentrate on information related to the current subgoal of a potentially large proof tree. Traf's window serves as an informative monitor that displays details of the steps in the proof.

Traf can also be used as a proof script viewer. Arbitrary subtrees can be shrunk so as to enable the entire structure of the proof to be grasped. Detailed information such as the assumptions and the subgoal at each proof step can be examined later. Since no information for the corresponding proof script is lost, the constructed proof tree can be directly used as proof documentation. With Traf the user can obtain a LaTeX description of the tree for documentation.

The rest of the paper is organized as follows. In Sect. 2, we describe the structure of a tree constructed by Traf. We discuss the usages and effectiveness of Traf in Sects. 3 and 4. In Sect. 5, we summarize the strengths and weaknesses of Traf. We conclude in Sect. 6 with a brief summary and mention of future work.

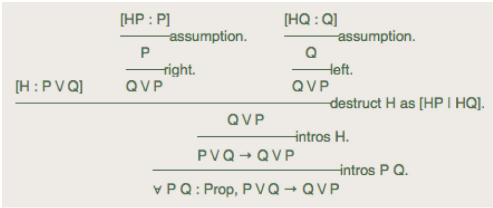
The current version of Traf was constructed based on Prooftree [23] and is available at https://github.com/hide-kawabata/traf.

2 Visualization of a Proof Script as a Proof Tree

Figure 1 shows a proof script for Coq and the corresponding proof tree constructed by Traf. As shown in Fig. 1(b), a proof tree constructed by Traf looks

```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```

(a) Proof script for Coq



(b) Proof tree constructed by Traf

Fig. 1. Proof script for Coq and corresponding proof tree constructed by Traf.

$$\frac{A}{A \vee B}\,(\vee\text{-intro 1}) \qquad \frac{B}{A \vee B}\,(\vee\text{-intro 2}) \qquad \frac{A[y/x]}{\forall x.A}\,(\forall\text{-intro})$$

$$\frac{\begin{array}{ccc} & [A] & [B] \\ & | & | \\ A \vee B & C & C \end{array}}{C}\,(\vee\text{-elim}) \qquad \frac{\begin{array}{c} [A] \\ | \\ B \end{array}}{A \to B}\,(\to\text{-intro})$$

Fig. 2. Natural deduction inference rules.

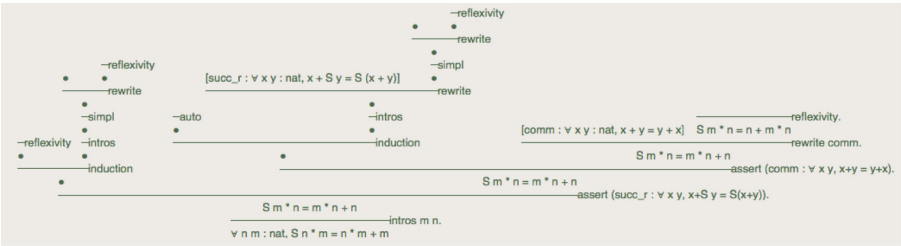
like an ordinary proof tree for Gentzen-style natural deduction: it is apparent that the natural deduction inference rules shown in Fig. 2 are combined for constructing the tree shown in Fig. 1(b). However, the details are different. A proof tree used in proof theory is a tree in which each node is a statement (or *subgoal*), and each line with a label indicates the application of the inference rule or axiom identified by the label. In the case of a proof tree constructed by Traf, the label attached to a line is not the name of an inference rule but rather is a *proof command* given to Coq at the proof step. Nodes written over a line are subgoals generated by the application of the proof command to the subgoal written under the line. When a complicated proof command combined by tacticals or a tactic that invokes an automated procedure is applied to a subgoal, the effect might not be as readily understandable as a Gentzen-style proof. However, a proof tree constructed by Traf is much more informative than the corresponding proof script.

Since some commands change only assumptions (and not subgoals), all the subgoals that appear in the course of a proof and all the proof commands used in the proof together and using them to construct a proof tree is not enough to enable the user to mentally reconstruct the proof session by simply looking at the proof tree. For example, the user will not recognize the application of the command "`apply H.`" unless the meaning of `H` is apparent. Traf makes a proof tree as readable as possible by

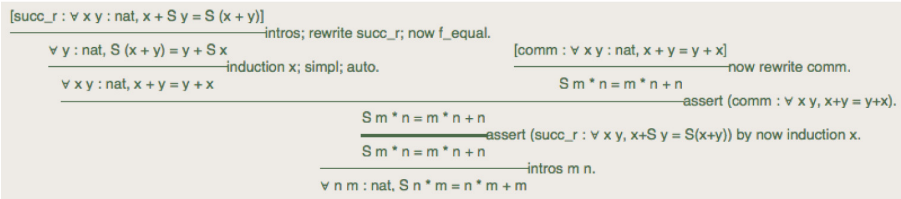
1. showing the assumptions used explicitly as branches of the *proven subgoals* over the line when a command refers to assumptions and

2. indicating the steps that do not change subgoals, i.e., the steps that modify only assumptions, by using a line in a specific style, such as bold ones.

The first measure results in proof trees that resemble Gentzen-style natural deduction proofs where the discharged assumptions are explicitly indicated. Although the second measure might not be the best way of illustrating proof scripts, it does ensure that each proof step actually taken is clearly recognizable.



(a) Verbose proof script makes proof tree complicated



(b) Use of tacticals and automation could simplify proof tree

Fig. 3. Two proof trees constructed by Traf corresponding to two versions of proof for same lemma.

Figure 3 shows two proof trees constructed by Traf corresponding to two versions of proof for the same lemma. In Fig. 3(a), some nodes are shrunk. The proof corresponding to the tree in Fig. 3(b) is essentially the same as that in Fig. 3(a), but the latter tree is smaller due to the use of tacticals. The shape of a proof tree constructed by Traf corresponds exactly to the structure of the proof script.[1] Unlike tools such as Matita [5], which generates descriptions of proofs by analyzing proof terms, Traf simply reflects the structure of a proof script in the proof tree.

The example tree in Fig. 3(b) includes a proof step at which the subgoal does not change. The use of tactics such as `assert` for controlling the flow of a proof can be treated naturally, as shown in Fig. 3.

Figure 4 shows the proof tree for a proof script using SSReflect [16] tactics. As shown in Figs. 1(b), 3, and 4, the major tactics of Coq and SSReflect are recognized by Traf.[2] At each proof step, Traf extracts the identifiers used in

[1] Although non-logical tactics such as `cycle` and `swap` can be used in proof scripts, the resulting proof trees are not affected by their use.

[2] The use of goal selectors is currently not supported.

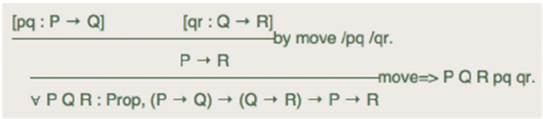
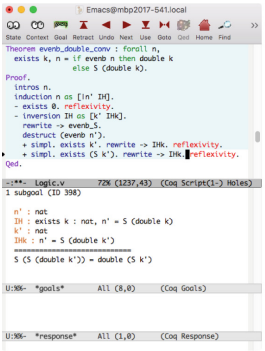


Fig. 4. Use of SSReflect [16] tactics.

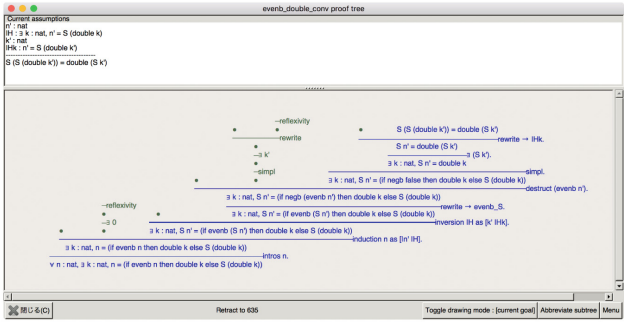
each proof command, checks whether they exist in the set of local assumptions available at that step, and explicitly displays the assumptions used. Externally defined identifiers are ignored.

3 Traf as a Proving Situation Monitor

Once Traf is enabled, a proof tree is constructed in an external window as shown in Fig. 5. The tree's shape changes as the proving process proceeds. The user does not have to interact with Traf's window while proving a theorem since Traf automatically updates the proof tree by communicating with Proof General.



(a) Proof General



(b) Traf's window for "Current Goal" mode

Fig. 5. Screenshots illustrating scene in proving process using Coq through Proof General accompanied by Traf. While user interacts with Coq through Proof General, as shown in (a), Traf automatically updates the corresponding proof tree, as shown in (b). (Color figure online)

The Traf window shows a summary of the situation facing the user during the process of proving a theorem. It has two panes, as shown in Fig. 5(b). The lower pane shows the proof tree as currently constructed. Finished subgoals are shown in green, and the path from the root to the current subgoal is shown in blue. Other subgoals to be proved, if any, are shown in black.[3]

[3] Colors can be specified by the user.

The upper pane of the window shows the assumptions and the subgoal at the current state. When the user selects a subgoal on the tree by clicking the mouse, Traf marks the subgoal on the tree as selected and shows the assumptions at the state facing the subgoal in the upper pane. When the user selects a proof command, Traf shows in the upper pane the corresponding raw text string without abbreviation that was entered by the user.[4]

When a proof command is entered, Traf draws a horizontal line segment over the ex-current subgoal and places the input proof command at the right end of the line. Subgoals generated by applying the command are placed over the line. When the user retracts proof steps, the tree's shape is also restored. A proof tree constructed by Traf can be seen as a record of proof step progress.

When an entered command finishes a subgoal, Traf changes the subtree's color to indicate that the branch is finished and switches its focus to one of the remaining subgoals in the proof tree window. At every moment, the current subgoal as well as every node on the path from the root to the subgoal is shown in blue in order to distinguish the current branch from other proven/unproven branches. We call the path from the root to the current subgoal the *current path*. Traf offers "Current Goal" display mode in which nodes that are not on the current path are automatically shrunk, as shown in Fig. 5(b).

Finishing a proof, i.e., entering the "`Qed.`" or "`Defined.`" command to Coq via Proof General, terminates communication between Traf and Proof General. Traf then freezes the proof tree in the window. The window remains on the screen, and the user can interact with it: clicking on a proof command node or a subgoal node invokes a function to display the detailed information at the corresponding proof step.

4 Traf as a Proof Script Browser

Traf can also be used as a tool for browsing existing proof scripts by transforming them into proof trees by using Proof General and Coq. In addition to checking each step by looking at explicitly displayed proof commands, assumptions, and subgoals, the user can consult Traf for all assumptions that were valid at any step in the course of the proof.

If the proof tree becomes very large, the complete tree cannot be seen in Traf's window. Traf thus offers, in addition to scrollbars, a facility for shrinking an arbitrary node, which is helpful for setting off specific portions of the tree. Any subtree can be shrunk/expanded by selecting its root node and pressing a button at the bottom of the window.

Traf can generate LaTeX descriptions of the displayed tree for typesetting by using `prftree` package.[5] The variation of the details of the tree, i.e., the existence of shrunk branches and/or unproven branches, is reflected in the rendered tree.

[4] Command texts that are longer than the predefined length are placed on the tree in an abbreviated form. The threshold length is an adjustable parameter.

[5] https://ctan.org/pkg/prftree.

5 Discussion: Strengths and Weaknesses of Traf

Many tools have been proposed for facilitating proof comprehension. Many of them visualizes proofs graphically [7,11,19,24], and some offer facilities for explaining proofs [6,12,21]. Some have graphical interfaces for interactive theorem proving [3,18,20]. Compared with these systems, Traf's strength is its usefulness as a graphical and informative monitor of the proof states while proving lemmas by using a tactic-based language through a standard interface. In addition, Traf is easy to use and requires no cost for Proof General users. It can be used with the Emacs editor by adding the settings for Traf to the Emacs configuration file.

As a viewer for proof scripts, Traf's approach resembles that of the Coqatoo tool [6] in the sense that both systems enhance the readability of tactic-based proof scripts by presenting the scripts with appropriate information. However, unlike Coqatoo, Traf can be used while proving theorems interactively.

ProofWeb [18] has functionality similar to that of Traf. Although its web interface is promising for educational use, its tree drawing facility is a bit restrictive and not very quick. It therefore would not be a replacement for the combination of Proof General and Traf.

One weakness of Traf mainly stems from its style, i.e., the use of trees for representing proof scripts. Complicated proofs might be better expressed in text format, and other approaches, such as those of Coqatoo [6] and Matita [5], might be more suitable. For browsing extremely large proofs, a method for visualizing large-scale proofs as Sunburst Trees [19] would be preferable. Nevertheless, Traf is appropriate for use as a proving situation monitor.

Another weakness is the environment required. Since the current version of Traf depends on the LablGtk2 GUI library [2], the techniques usable for the graphical representation are restricted. In addition, Traf's implementation depends on that of Proof General.

The current version of Traf is based on Prooftree [23], which was developed by Hendrik Tews. The facilities for communicating with Proof General, many of its basic data structures, and the framework for drawing trees were not changed much. Some of Traf's functionalities, such as those described in Sects. 3 and 4, are based on those in Prooftree. While Traf owes much to Prooftree, it offers added value due to the functionalities introduced for guiding interactive proving sessions by displaying informative proof trees.

6 Conclusion and Future Work

The Traf graphical proof tree viewer cooperates with Coq through Proof General. A user who proves theorems by using Coq through Proof General can thus take advantage of Traf's additional functionalities at no cost.

Future work includes enhancing Traf to enable it to manipulate multiple proofs, to refer to external lemmas and axioms, and to better handle lengthy proof commands.

References

1. The compcert project. http://compcert.inria.fr
2. Lablgtk2. http://lablgtk.forge.ocamlcore.org
3. Pcoq: a graphical user-interface for coq. http://www-sop.inria.fr/lemme/pcoq/
4. The sel4 microkernel. http://sel4.systems
5. Asperti, A., Coen, C.S., Tassi, E., Zacchiroli, S.: User interaction with the matita proof assistant. J. Autom. Reason. **39**(2), 109–139 (2007)
6. Bedford, A.: Coqatoo: generating natural language versions of coq proofs. In: 4th International Workshop on Coq for Programming Languages (2018)
7. Byrnes, J., Buchanan, M., Ernst, M., Miller, P., Roberts, C., Keller, R.: Visualizing proof search for theorem prover development. Electron. Notes Theor. Comput. Sci. **226**, 23–38 (2009)
8. Corbineau, P.: A declarative language for the coq proof assistant. In: Miculan, M., Scagnetto, I., Honsell, F. (eds.) TYPES 2007. LNCS, vol. 4941, pp. 69–84. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68103-8_5
9. Coscoy, Y., Kahn, G., Théry, L.: Extracting text from proofs. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902, pp. 109–123. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0014048
10. Curzon, P.: Tracking design changes with formal machine-checked proof. Comput. J. **38**(2), 91–100 (1995). https://doi.org/10.1093/comjnl/38.2.91
11. Dunchev, C., et al.: Prooftool: a GUI for the GAPT framework. In: Proceedings 10th International Workshop On User Interfaces for Theorem Provers (2013)
12. Fiedler, A.: *P.rex*: an interactive proof explainer. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 416–420. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45744-5_33
13. Giero, M., Wiedijk, F.: MMode, a Mizar Mode for the proof assistant coq. Technical report, Nijmegen Institute for Computing and Information Sciences (2003)
14. Gonthier, G.: A computer-checked proof of the four colour theorem (2006). http://www2.tcs.ifi.lmu.de/~abel/lehre/WS07-08/CAFR/4colproof.pdf
15. Gonthier, G., et al.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 163–179. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39634-2_14
16. Gonthier, G., Mahboubi, A.: An introduction to small scale reflection in coq. J. Form. Reason. **3**(2), 95–152 (2010)
17. Harrison, J.: A mizar mode for HOL. In: Goos, G., Hartmanis, J., van Leeuwen, J., von Wright, J., Grundy, J., Harrison, J. (eds.) TPHOLs 1996. LNCS, vol. 1125, pp. 203–220. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0105406
18. Hendriks, M., Kaliszyk, C., van Raamsdonk, F., Wiedijk, F.: Teaching logic using a state-of-the-art proof assistant. Acta Didact. Napoc. **3**, 35–48 (2010)
19. Libal, T., Riener, M., Rukhaia, M.: Advanced proof viewing in ProofTool. In: Eleventh Workshop on User Interfaces for Theorem Provers (2014)
20. Sakurai, K., Asai, K.: MikiBeta : a general GUI library for visualizing proof trees. In: Alpuente, M. (ed.) LOPSTR 2010. LNCS, vol. 6564, pp. 84–98. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20551-4_6
21. Tankink, C., Geuvers, H., McKinna, J., Wiedijk, F.: Proviola: a tool for proof re-animation. In: Autexier, S., et al. (eds.) CICM 2010. LNCS (LNAI), vol. 6167, pp. 440–454. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14128-7_37

22. Tesson, J., Hashimoto, H., Hu, Z., Loulergue, F., Takeichi, M.: Program calculation in coq. In: Johnson, M., Pavlovic, D. (eds.) AMAST 2010. LNCS, vol. 6486, pp. 163–179. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17796-5_10
23. Tews, H.: Prooftree. https://askra.de/software/prooftree/
24. Trac, S., Puzis, Y., Sutcliffe, G.: An interactive derivation viewer. Electron. Notes Theor. Comput. Sci. **174**(2), 109–123 (2007)
25. Wenzel, M., Wiedijk, F.: A comparison of Mizar and Isar. J. Autom. Reason. **29**, 389–411 (2002)
26. Wiedijk, F. (ed.): The Seventeen Provers of the World. LNCS (LNAI), vol. 3600. Springer, Heidelberg (2006). https://doi.org/10.1007/11542384