Interactive Poster: Interactive Symbolic Visualization of Semi-automatic Theorem Proving

Chandrajit Bajaj, Shashank Khandelwal, J Moore, Vinay Siddavanahalli*
Center for Computational Visualization,
Department of Computer Sciences and Institute for Computational Engineering & Sciences,
University of Texas, Austin Texas 78712

Abstract

We present an interactive visualization environment for semi-automatic theorem provers in an attempt to help users better steer their theorem proving process. The augmented theorem proving environment provides synchronized multi-resolution textual and graphical views and direct navigation of large expressions or proof trees from either of the twin interfaces. We identify three levels of the proof process at which synchronized multi-resolution textual and graphical visualizations enhance user understanding.

1 Introduction

User interaction with theorem provers remains mostly text based. When a proof attempt fails, the user needs to diagnose the problem and then come up with new theorems, lemmas or hints to continue. This requires a thorough understanding of each proof attempt. Theorem provers typically generate large amounts (megabytes) of text during proof attempts; making intermediate expression navigation in the proof process a significant challenge. A command line text interface is used with most theorem provers. Pretty printing and text based primitives like searching are the main tools available to help reduce or manage visual complexity. The challenge is to integrate text-based interfaces with synchronized graphical visualization to speed comprehension and interaction. We identify three levels at which the command line interface could be augmented with synchronized graphical visualization for enhanced user understanding:

1. The overall proof attempt can be visualized by a graph of the theorems used during a particular proof attempt.
2. The structure of the proof can be visualized, by displaying the subgoals created at each step and indicating which subgoals could be proved or not.
3. Examining failed subgoals is critical towards understanding why a proof attempt failed. Following the progress of similar subgoals through the proof attempt is useful. Graphical visualization would help quickly identify similar subgoals and their locations within the overall proof.

We choose ACL2 [Kaufmann and Moore 1997], an industrial strength theorem prover for our case study.

2 Related Work

We provide a couple of relevant references here to previous work done in the visualization of output from theorem provers. The remaining are cited in the full version of the paper [Bajaj et al. 2003]. Paper [Thiry et al. 1992] discusses the need and requirements for a user friendly interface to theorem provers, but do not visualize the information inherent in the proof process as a way to understand the proof attempt. In [Goguen 1999], we see an attempt to use visualization to understand the structure of proofs, and a complete system for developing a user interface. Their system is designed for readers of proofs (as opposed to specifiers or provers). Web pages explain each proof with links to background material and tutorials. Their system is designed with distributed collaboration in mind. Our system is designed to be used by theorem *provers*, working alone.

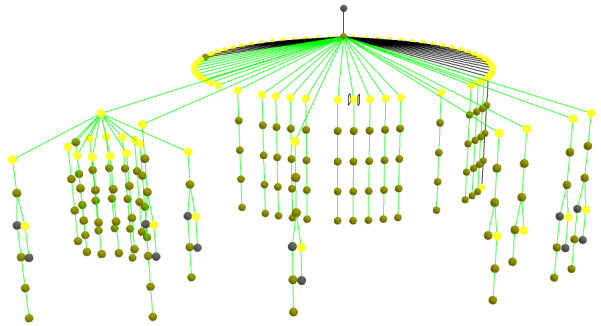

Figure 1: *Proof tree visualization. This is a proof related to a Java Virtual Machine.*

3 Symbolic Visualization

We provide details and justifications for the visualizations at each level of the hierarchy.

Visualization of Theorems Used A theorem is proved by using previously verified theorems and lemmas. These verified theorems and lemmas are used as a knowledge base for the theorem prover. By looking at the theorems and lemmas used in the proof of a previously verified theorem, a user may gain insight on how to steer a current proof attempt. The theorems and lemmas used during a proof attempt, when arranged to show inter-dependency, form a directed acyclic graph. This can be visualized using a simple node-link diagram.

Proof Tree Visualization Most proof attempts have a tree structured approach; with the main theorem being proved as the root of the tree. A theorem prover either proves/disproves the theorem, or divides the theorem into subgoals. Each of these subgoals is then tackled in an order determined by the particular theorem proving system. ACL2 tends to use depth first search. See figure 1.

We provide a synchronized multi-view representation of the proof tree to the user. Since proof attempts tend to be large, taking possibly hours to finish, users prefer being given synchronized

*{bajaj, shrew, moore, skvinay}@cs.utexas.edu

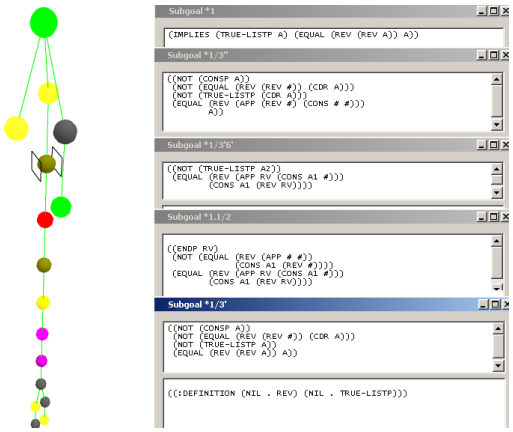

Figure 2: *Multi-view text and graphical visualization of a proof attempt. The text windows on the left contain the contents of some nodes from the proof tree on the right. The screen shot is from the proof of the proposition that the reverse of the reverse of a list is the list itself (given certain conditions and definitions).*

feedback in both textual and graphical views of the current state of the proof. We use a variant of the cone tree algorithm [Carriere and Kazman 1995] to render, annotate and provide interactive navigation for our trees. ACL2 has a model in which the subgoals can be reduced using generalization, induction, simplification, etc. These actions are limited and distinct and can be visualized by the current node's color, as shown in figure 2.

Expression Visualization The main hurdle in finding out why the theorem prover could not prove a theorem is understanding the critical node at which the theorem prover failed or deviated from the expected path. The expression trees of formulas at a subnode can be visualized as a 2D tree. In theorem proving, larger proof attempts are cumbersome to follow. From one goal to another, the theorem prover performs some actions, modifying the expressions at each stage. In order to follow changes, pattern matching can be applied to the expressions (after suitably representing them as trees).

The tree matching algorithm we use is similar to the recursive algorithm presented by [Hoffmann and O'Donnell 1982]. Our heuristics for matching are domain specific. A Lisp expression E can be represented as a function symbol F operating on a set of parameters $p_1, P_2, \ldots, P_n$. In a binary tree representation, the left child of the root node contains the function symbol F. The parameters are then the left children of all the nodes of the right side path from the root to a leaf. Two trees which have different function symbols result in a low match. The match is also proportional to the distance from the root of the differences between the trees. A permutation in the parameters of a function symbol results in a high match.

The visualization of the results from the pattern matching has been implemented in both text and graphics. In figure 3 we see two sets of texts. A sub expression from column 2 is matched with the entire expression on the right. The font color indicates how similar an expression is to the search expression. Unselected text is light gray, while selected text is black. The results from pattern matching are shown by varying the font color from bright red (high match) to dark red (low match). The graphical expression visualization interface also shows the same results (the first column in figure 3). The unselected sections are gray, while selections are cyan. Again, bright to dark red is used to show high to low matches between the patterns. The third tree in the left column is a zoomed-in view of the outlined box in the second tree.

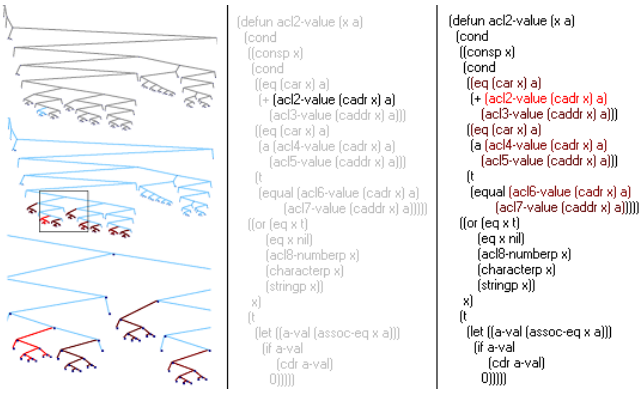

Figure 3: *A synchronized view of text and graphics visualizations from level 3. Pattern matching of expressions from a proof: A composition of screen shots from our implementation.*

4 Conclusion

We have presented some details of our interactive visualization environment for semi-automatic theorem provers. Further details are available from the full version of the paper [Bajaj et al. 2003], (with system animations), from our Symbolic Visualization web page (http://www.ices.utexas.edu/CCV/projects/VisualEyes/SymbVis/).

5 Acknowledgments

We are grateful to Robert Krug for writing the socket code that helps us communicate with ACL2. Research supported in part by grants from NSF CCR-9988357and ACI 9982297.

References

BAJAJ, C., KHANDELWAL, S., MOORE, J., AND SIDDAVANAHALLI, V. 2003. Interactive symbolic visualization of semi-automatic theorem proving. In *CS and ICES Technical Report*.

CARRIERE, J., AND KAZMAN, R. 1995. Interacting with huge hierarchies: Beyond cone trees. In *proceedings of IEEE Information Visualization*, 74–78.

GOGUEN, J. A. 1999. Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing 11*, 3, 272–301.

HOFFMANN, C. M., AND O'DONNELL, M. J. 1982. Pattern matching in trees. *Journal of the ACM (JACM) 29*, 1, 68–95.

KAUFMANN, M., AND MOORE, J. S. 1997. An industrial strength theorem prover for a logic based on common Lisp. *Transactions on Software Engineering 23*, 4, 203–213.

THIRY, L., BERTOT, Y., AND KAHN, G. 1992. Real theorem provers deserve real user-interfaces. In *proceedings of the fifth ACM SIGSOFT symposium on Software Development Environments*, ACM Press, 120–129.