

The User Interface of the KIV Verification System — A System Description

Dominik Haneberg, Simon Bäumler, Michael Balsler,
Holger Grandy, Frank Ortmeier, Wolfgang Reif,
Gerhard Schellhorn, Jonathan Schmitt, Kurt Stenzel

*Lehrstuhl für Softwaretechnik und Programmiersprachen
Institut für Informatik, Universität Augsburg
86135 Augsburg Germany*

*E-Mail: {haneberg, baeumler, balsler, grandy, ortmeier, reif, schellhorn,
jonathan.schmitt, stenzel}@informatik.uni-augsburg.de*

Abstract

This article describes the sophisticated graphical user interface (GUI) of the KIV verification system. KIV is a verification system that works on structured algebraic specifications. The KIV GUI provides means for developing and editing structured algebraic specifications and for developing proofs of theorems. The complete development process is performed through the GUI with two exceptions. For editing the specification files XEmacs is used, and for the management of the structured algebraic specifications we use daVinci, an extendable graph drawing tool. As proving is the most time-consuming part of formal verification, the most important part of the KIV GUI is our user interface for proof development. The proof is represented as a tree and can be manipulated through context menus. The main proof work is done in a proof window where the sequent of the current goal, the applicable rules and the main menu are displayed. Which rules are applicable depends on the current goal. KIV also supports the context-sensitive application of proof rules.

Key words: Graphical User Interfaces, Theorem Proving,
Verification

1 Introduction

Graphical user interfaces (GUI) can greatly improve the power of an interactive theorem proving system. A good user interface supports the proof engineer and simplifies the proving process compared to a proof development by issuing text commands on a shell-like command console. Providing an intuitive and usable interface for the verification system is as important as the development of heuristics or improving the simplifier.

The GUI of KIV [BRS⁺00, BRSS99, RSSB98, RSS97] is very elaborate. It was developed over several years as an integral part of the verification system. Different from other provers and GUI developments, e.g. ProofGeneral [Asp00] or *LOUI* [SHB⁺99] for the Ω system, the GUI of the KIV system was not subsequently added to the core prover but instead it was part of the development from its very beginning. The first version of the KIV GUI was written in Motif, the recent versions were developed in Java.

The KIV GUI allows the user to manage the algebraic specification, view and manipulate proof trees and prove theorems. In the following sections the GUI will be described in detail. Section 2 gives an overview of the KIV system, Section 3 describes the tool for managing structured specifications. Section 4 describes the user interface for constructing proofs. A short description of the online help is given in 5. Section 6 concludes.

The KIV system is available to the scientific community. For more information please contact the authors or look at our web page¹. KIV comes with a documentation that contains a detailed description of the features of the KIV system and a practical introduction to its usage. This practical course is used in the education of students at the University of Augsburg and the University of Leipzig.

2 Overview of the KIV System

The KIV system is an advanced tool for engineering high assurance systems. It provides an economically applicable verification technology, and supports the entire design process from formal specifications to executable verified code. At the basis of the development are structured algebraic specifications. The core prover is an interactive prover for higher-order logic based on a Gentzen-style sequent calculus [HHRSS86]. For verifying implementation correctness the KIV system features Dynamic logic rules for a Pascal-like imperative language [HRS89] as well as for imperative Java [Ste04]. For the analysis of concurrent systems, temporal logic is supported.

Figure 1 shows the structure of the KIV system, it consists of three parts: The core prover, the GUI for the prover and a tool for visualizing and managing the structured specification. The core prover is written in PPL² and communicates with the GUI over a TCP socket. The GUI is written in Java.

Most tasks of the verification process are executed using the Java GUI of KIV, but there is one exception. For the management of the specification structure a general purpose graph drawing tool (daVinci [FW94]) is used. The daVinci tool was developed at the University of Bremen. It can be customized and can send data to other applications. We have added KIV specific commands to the daVinci menu and daVinci is used to send input to the KIV

¹ <http://www.informatik.uni-augsburg.de/swt/kiv>

² PPL is a ML-like functional programming language.

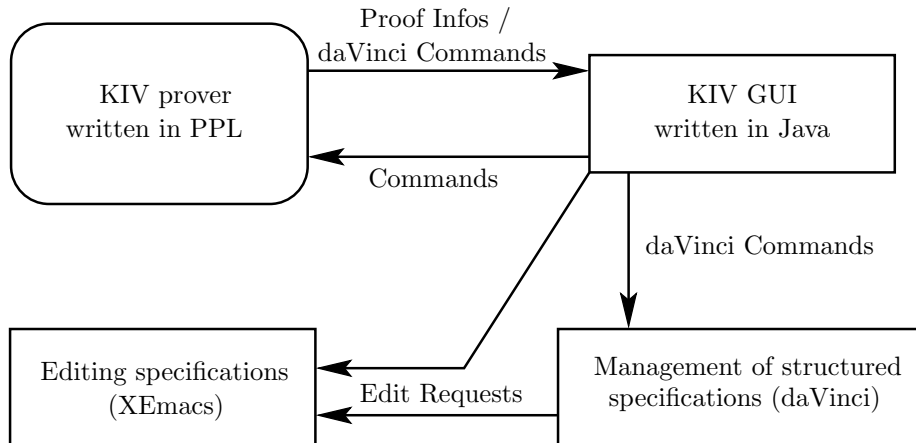


Fig. 1. Overview of the KIV System

system core via the GUI which is responsible for the startup and control of daVinci.

In the following section the two parts of the user interface of the KIV system are described in more detail.

3 Management of Structured Algebraic Specifications

A structured algebraic specification is a hierarchy of data type definitions and structuring operations (union, actualization, enrichment, ...). It can be represented as directed acyclic graph. Therefore we use a tool for graph drawing and layout to represent the specification hierarchy. Figure 2 shows a screenshot of the daVinci tool containing a small specification as well as the context menu added by the KIV system. An element of a structured specification can be in different states which are represented by different colors and forms of the nodes in the daVinci graph. Specifications imported from a library appear in orange, specifications that contain only proved lemmas can be put in “proved state” and are colored green, blue specifications are not completely proved. Nodes that contain programs (so called “Modules”) have a rhombical form.

The context menu (on the lower left of Figure 2) appears after clicking on a node of the graph of the structured specification, it contains operations that manipulate the state of this node, e.g. reloading a specification file (“Reload”) or begin proofs in this specification (“Work on ...”). “Work on ...” loads the theorem base³ of all necessary specifications and opens the main window of the KIV GUI (described in Section 4).

³ The theorem base is the set of all available theorems, i.e. axioms and user-defined lemmas. The theorem base contains proved as well as unproved lemmas. The KIV system does not enforce a bottom-up proving, the proof engineer may use unproved lemmas in his proofs and the correctness management guarantees that finally all used lemmas are actually proved and do not have cyclic dependencies.

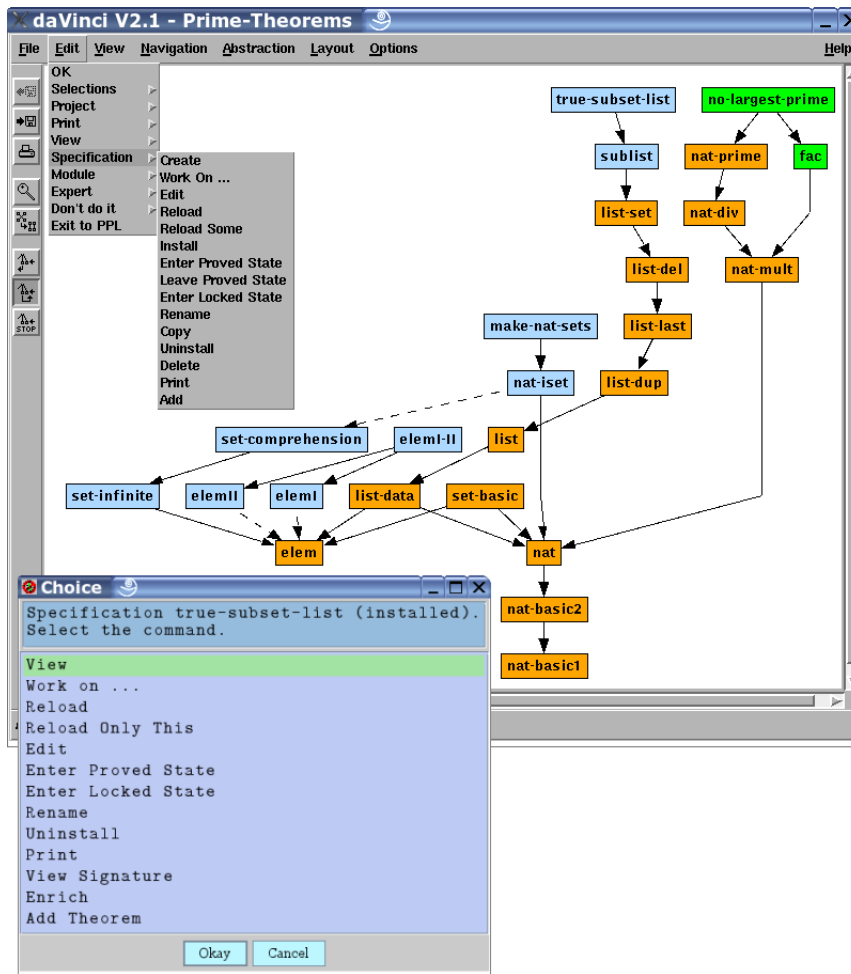


Fig. 2. The daVinci Tool used for Managing Structured Specifications

The menu of daVinci can be extended to fit specific needs. In the “Edit” menu the KIV-specific menu items are added. There are menu items for e.g. importing library specifications, to create or delete specifications or to generate \LaTeX versions of the specifications for documentation purposes.

The basis of each part of a structured specification is a textual input file that may contain a sort definition, axioms or information for a structuring operations, e.g. a renaming of signature elements. We did not include a text editor in the KIV GUI, instead we use XEmacs with a special font to display mathematical characters. In Figure 3 you can see a screenshot of XEmacs with some lemmas on the right hand side and a specification for sets on the left. The input files are parsed and converted into the internal KIV representation after an “Install” or “Reload” command.

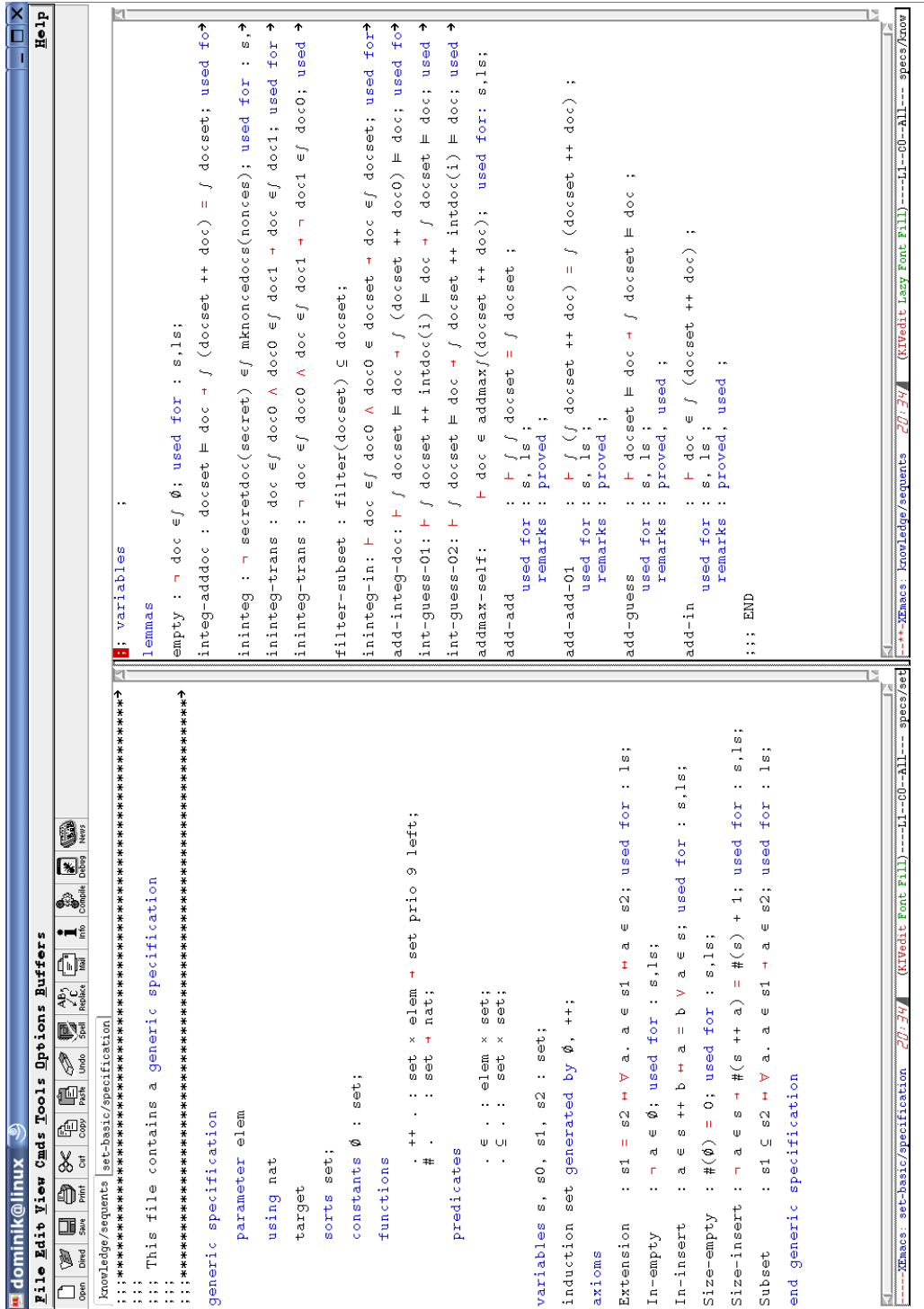


Fig. 3. Editing Specifications in XEmacs

4 The User Interface for Proving Theorems

In this section the central parts of the KIV GUI will be described. These parts are the main window of the KIV GUI and the windows with the proof trees.

In these windows most of the work is done. The following paragraphs describe the general functions of the main window (Section 4.1), the proof frame of the main window (Section 4.2) and the tree window (Section 4.3). Figure 4 gives an impression of the KIV GUI. Most aspects of the GUI are customizable,

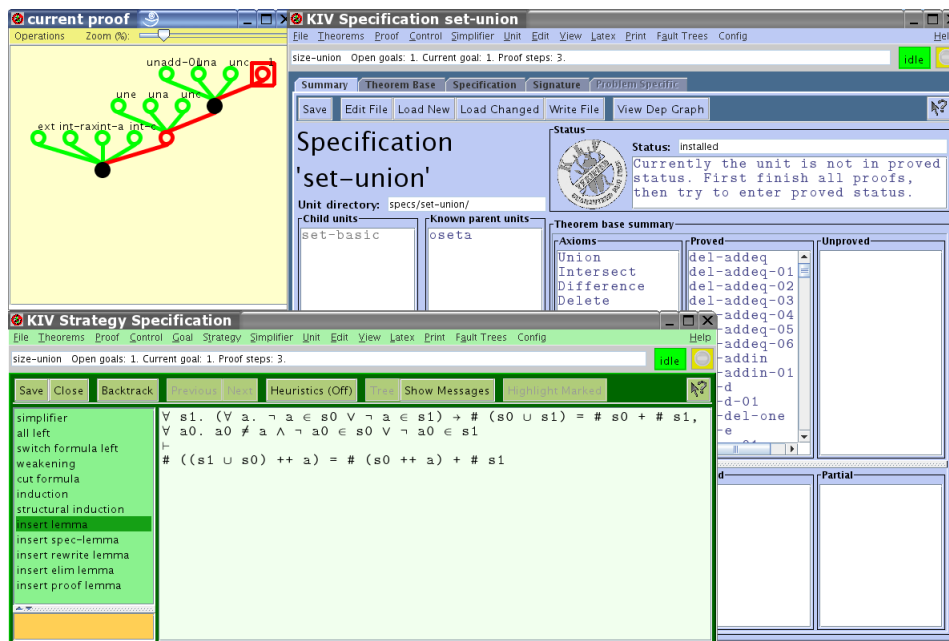


Fig. 4. The Windows of the KIV GUI

such as colors and fonts, using a “Config” dialog.

4.1 The Main Window

The main window of the GUI (the upper right window in Figure 4) is opened after the user selected “Work on...” for a specification in daVinci. This window offers a lot of information about the specification, such as the axioms in this specification, list of proved, unproved and invalid lemmas, a list of sub-specifications, the complete signature of all sub-specifications and the theorem base of the current specification (see Figure 5). The main window is also used to modify the theorem base, for example by adding or deleting theorems or by defining various types of simplifier-rules. Various options can be selected in an options dialog, statistics about the theorem base are available as well. To modify theorems without using XEmacs and loading the modified theorem from the theorem input file, a theorem editor window with support for the input of special characters is available. This window can be seen in Figure 6.

4.2 The Proof Frame

The proof frame, also called strategy frame, is the most important frame of the main window. All proofs are done in the strategy frame. It can be seen

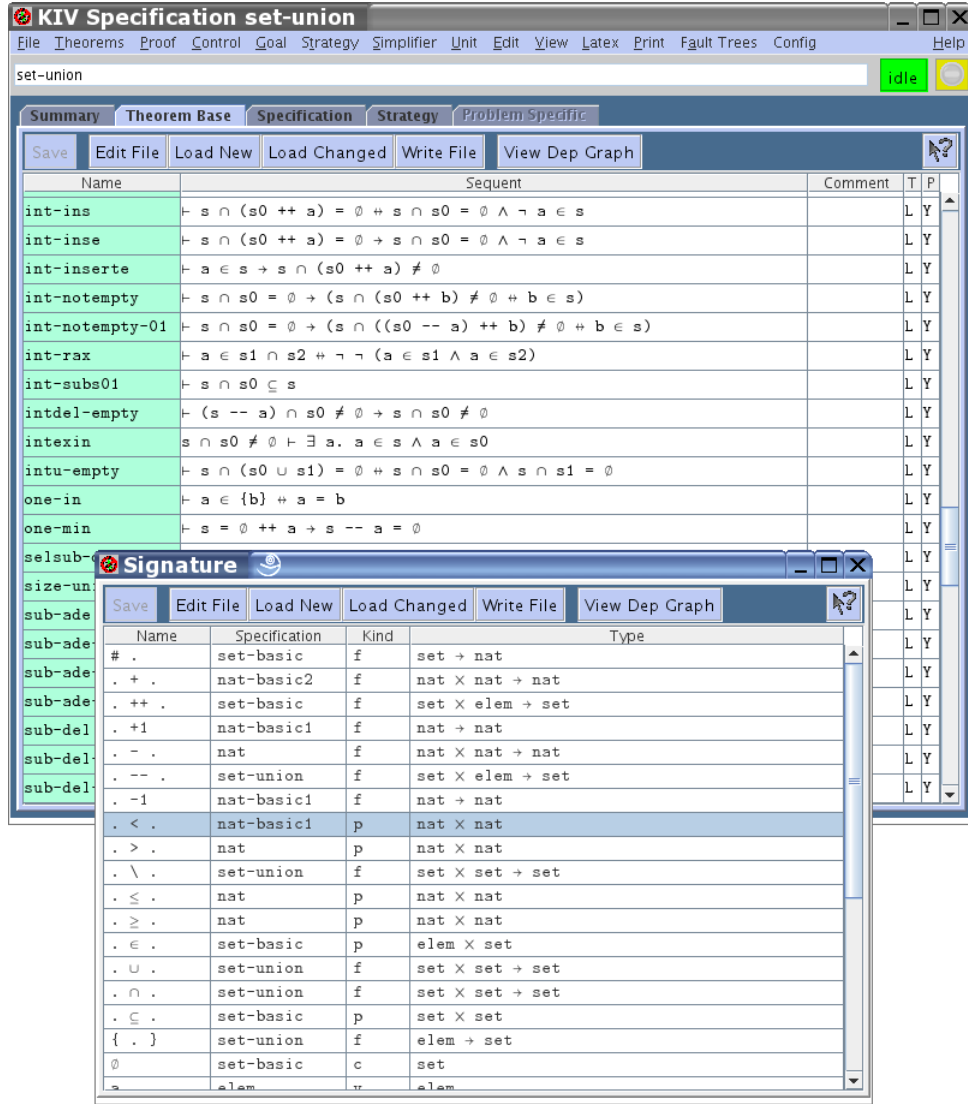


Fig. 5. Viewing the Theorem Base and the Signature Entries

in Figure 7. The large area on the right hand side contains the sequent of the current goal, i.e. what currently is to be proved. On the left side is the list of calculus rules that can be applied to the current goal. The status line under the menu bar provides information about the proof, the name of the lemma, the number of open goals, the number of the current goal and the total number of proof steps in this proof.

A very important feature of the strategy frame can be seen in Figure 8. The KIV GUI supports the context-sensitive application of calculus rules and most important the application of rewrite lemmas. When the user moves the mouse over the sequent, the KIV systems permanently determines the smallest (sub-)expression of the sequent containing the mouse cursor and marks it with a different color as seen in Figure 8: the (first) expression in the succedent is marked red. For displaying the context menu, the leading function or predicate

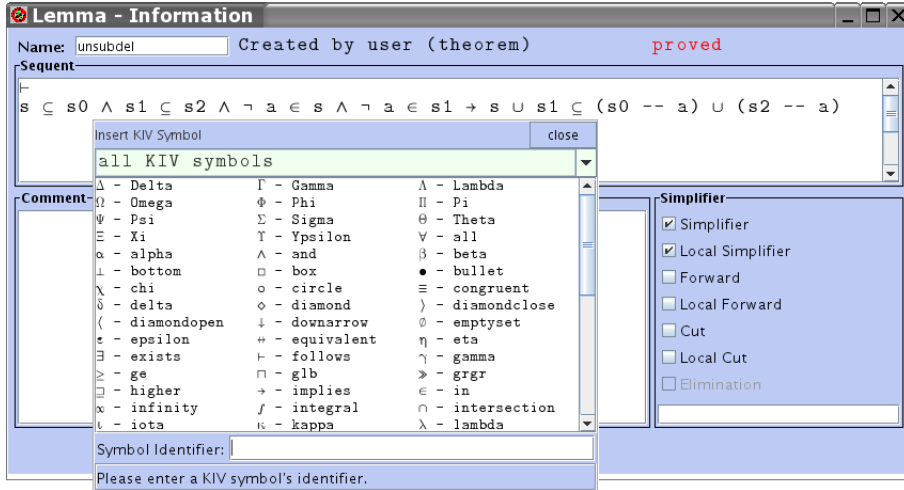


Fig. 6. The Window for Editing Theorems

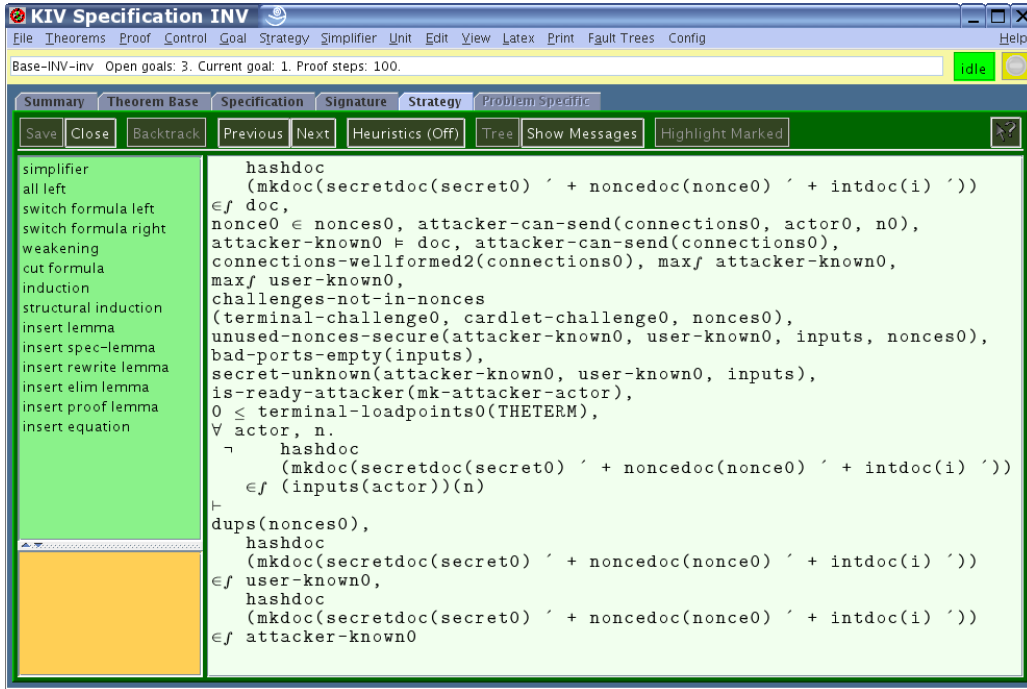


Fig. 7. The Strategy Frame for Proving

symbol or logical connective of the selected (sub-)expression is determined and KIV opens a context-menu with a list of rewrite lemmas applicable to the (sub-)expression or in case of a logical connective, the rules for dealing with this connective, e.g. “quantifier instantiation” for quantifiers. Figure 8 shows the result of a click on the predicate \subseteq in the succedent. The surrounding (sub-)expression is $s_1 \subseteq s_2$. When the user selects a rewrite lemma from the context-menu, the necessary substitution for the free variables is computed in order to match the rewrite lemma with the selected (sub-)expression of the sequent. Given all this information the “insert rewrite lemma” rule of

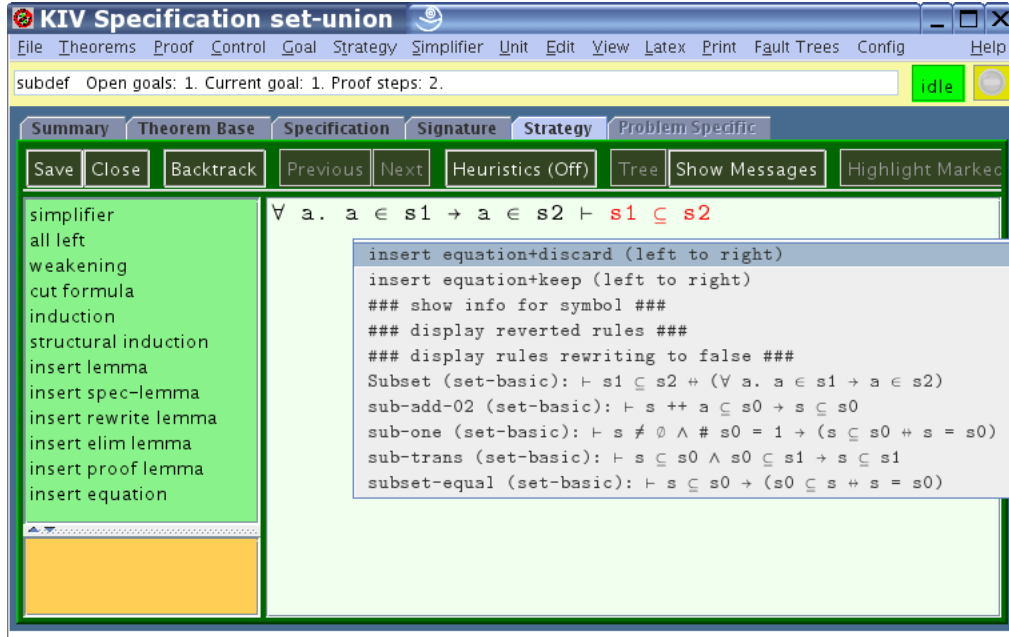


Fig. 8. Context-sensitive Application of Rules

the calculus is applied to the current goal. In large verification projects the theorem base quite often contains some thousands of theorems distributed over dozens of specifications. Without support finding the lemma you need in a proof is quite nasty. The context-sensitive computation of applicable lemmas for a selected expression, as described above, greatly reduces this problem. For example in Figure 8 only five applicable lemmas are listed in the context-menu but the theorem base contains more than 40 theorems that deal with the subset relation.

In the bottom-left corner of the strategy frame (Figure 8) another useful feature of the KIV GUI can be seen. There are two lists of lemmas, the so-called “recent lemmas” and the “hot lemmas”. Whenever a lemma is applied, either by the user or the heuristics, the used lemma is inserted into the list of recent lemmas. If the list has reached the configured maximal length the least used lemma is dropped from the list. If a lemma is very important and should not be discarded it can be marked as permanent using the context-menu of the lemma list. The permanent lemma is then moved from the “recent lemmas” into the “hot lemmas” list. The “hot” and “recent” lemmas are very useful when continuing a failed proof after correcting an error in the theorem as well as when proving a set of theorems dealing with the same function or predicate, since often a number of lemmas is needed in all these proofs.

As theorem proving is a process that often leads into dead ends, a mechanism to step back and remove wrong decisions is necessary. The KIV system offers two such mechanisms. One is the pruning of complete subtrees of the proof tree (this is described in Section 4.3), the other is a chronological backtracking. With every click on “backtrack” in the strategy frame the last proof

step is undone. This works for interactive steps as well as for steps performed by a heuristic.

4.3 The Tree Window

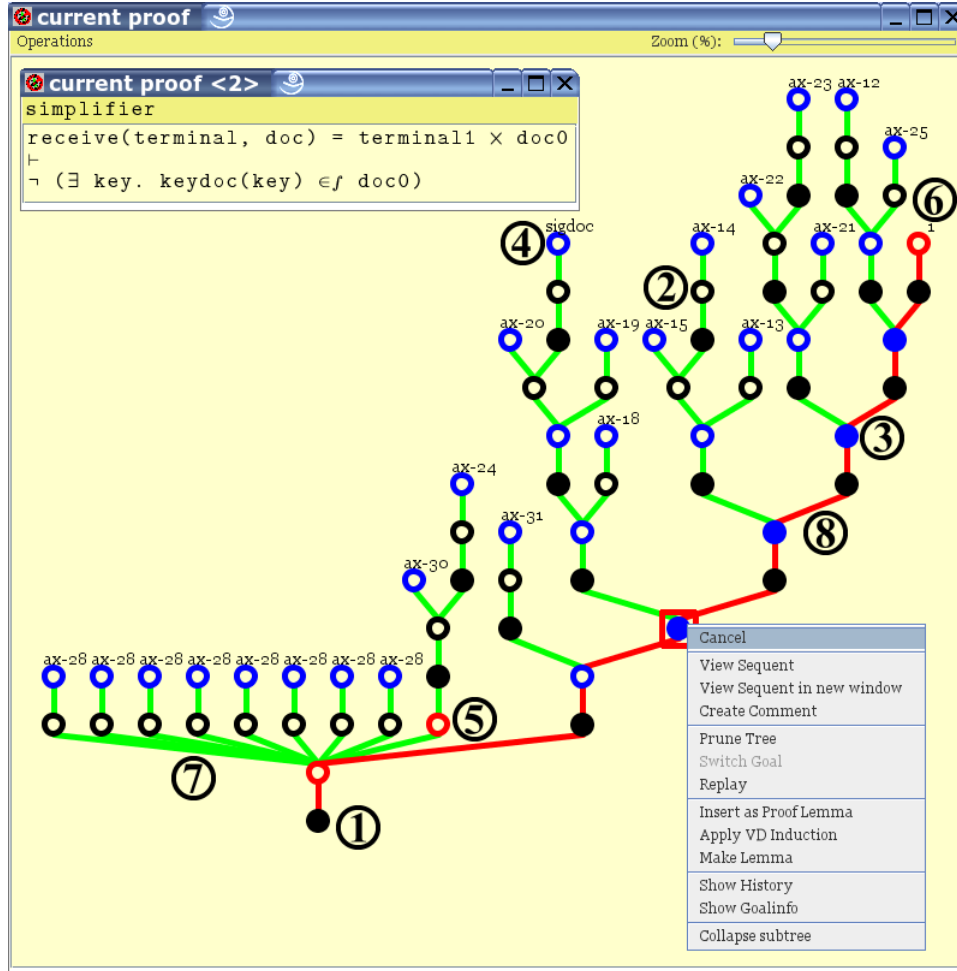


Fig. 9. An Example of a Proof Tree

A central concept in the KIV system is the proof object. Unlike most other verification systems known to us (PVS[ORS92], HOL[Gor88], ACL2[KM97], Isabelle[NPW02], etc.), the KIV system constructs an explicit proof object that contains all informations about the proof while proving a theorem. In the proof object for every step of the proof the current sequent, the applied proof rule, used lemmas and simplifier rules, the used substitution for the variable instantiation, the active heuristics and a lot of internal informations are stored. A proof is mostly not linear, in most cases it is a tree, because of calculus rules with more than one premise. The proof can be graphically represented as a proof tree where each node is a proof step and the edges connect one proof step with its successors. Figure 9 shows an example of a proof tree. Depending on the theorem, proofs come in very different sizes:

between one and some ten-thousand proof steps. To support the proof engineer when working with large proofs, the proof window offers a couple of interesting features. The proof window has scrollbars to move the visible area and the proof window can automatically track the current goal, i.e. the proof window always shows the open goal that the user currently works on. Additionally the proof window is zoomable. Figure 10 shows the proof tree for a proof with more than 3300 proof steps.

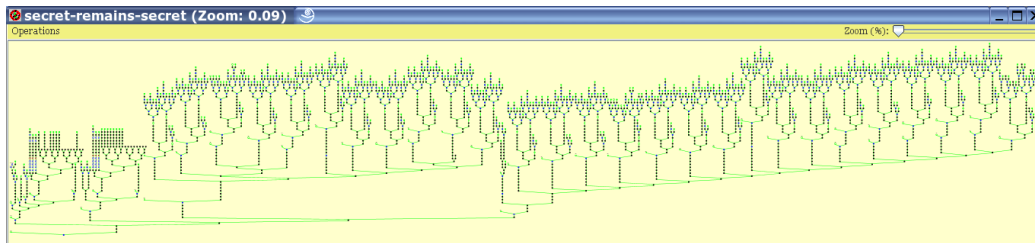


Fig. 10. A Proof Tree with 3396 Steps

As seen in Figure 9 the nodes of the proof tree have different forms and colors and the edges also have different colors. Filled out circles (① in Figure 9) represent steps performed by a heuristic, the others (②) represent interactive steps. Normal proof rules are represented by black circles (①), blue circles represent case distinctions (③) or lemmas (④), red circles represent induction steps (⑤) or open goals (⑥) and green circles represent quantifier instantiations. The edges of the proof also have different colors. Edges leading to an incomplete subtree are red (⑧), the edges of completely proved subtrees are green (⑦). At the nodes representing the usage of a lemma the name of the lemma is added to the node (④). KIV supports the usage of specification libraries with a large number of theorems by means of an elaborate correctness management [RSSB98].

The proof tree is an active object. Left-clicking on a node opens a new window with the sequent of this proof step and the applied proof rule (cf. upper-left region of Figure 9). A right-click on a node opens a context-menu with additional functions. The context-menu can be seen on the lower-right corner of Figure 9. Among other things the context-menu allows the user to directly make the selected goal the current goal, the user can prune a tree at the selected node, i.e. the complete subtree beginning at the selected node is deleted. This is an important feature for dealing with failing proofs (what happens very often in verification). With pruning a faulty decision is cut off the proof, and the user can correct it. Often wrong decision become obvious only after quite some additional proof steps, so pruning is more beneficial and flexible than chronological backtracking. Using the context-menu it is also possible to replay a part of the proof. By replaying a part of a proof tree all the proof steps in this subtree are appended to the current goal. Replay of (parts of) proofs is very useful and supports the proof engineer greatly. It is part of the KIV system since its early versions in the 1980 years and was

subsequently suggested for example in [MH97].

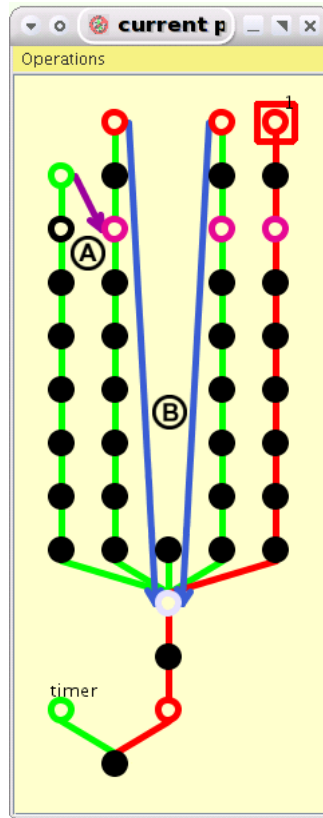


Fig. 11. Proof Tree with Applications of Special Proof Rules

Recently support for temporal logic has been added to KIV. The approach to handle recursive programs and temporal logic operators [BDRS02] is inspired by the Verification Diagrams as proposed by [MP94]. Temporal logic proofs allow — under certain preconditions — that the proof tree contains cycles, thus making it a proof graph. Proof graphs with cycles correspond to the graphs of verification diagrams which may have cycles too. Cycles in proof trees are constructed by two special proof rules. They cannot be applied on a single sequent but need two different sequences within the current proof. They are applied using the proof tree window.

One of the rules is the “Insert proof lemma”, the other rule is “Apply VD induction”. Insert proof lemma can be seen as a pointer in the proof tree, to indicate, that two nodes are essentially the same. It can be applied by first selecting the more specialized goal and making it active. Afterwards the “Insert proof lemma” menu item has to be selected out of the context-menu of the more generalized goal. After the rule application there remain two proof obligations. First the implication between the more specialized and the more generalized goal has to be shown, second the more generalized goal has to be proven. To prevent invalid proofs, a cycle check is included within the application of this rule.

The VD induction is a special form of induction, where the induction term is not determined when the user starts it. Instead, the user determines a node, where he thinks the application of an induction will be possible at a later time and determines a placeholder for the induction term, which is inserted in the sequent. The induction term will be calculated once the induction is applied. Therefore it is not sufficient to locally apply the induction rule within one node, as the induction term cannot be determined out of the sequent but only out of the combination of two sequents. Therefore, with the node where the induction should be applied marked active, the user has to select another node, which will represent the begin of the induction. After rule application the core logic calculates the induction term and tries to find a suitable substitution for all variables on the sequent. Afterwards the correctness of the induction application has to be shown.

Figure 11 shows a proof tree (to be more precise a proof graph) which contains applications of these two special rules. The purple arrow near \textcircled{A} represents the usage of a proof lemma, the two arrows at \textcircled{B} symbolize VD inductions.

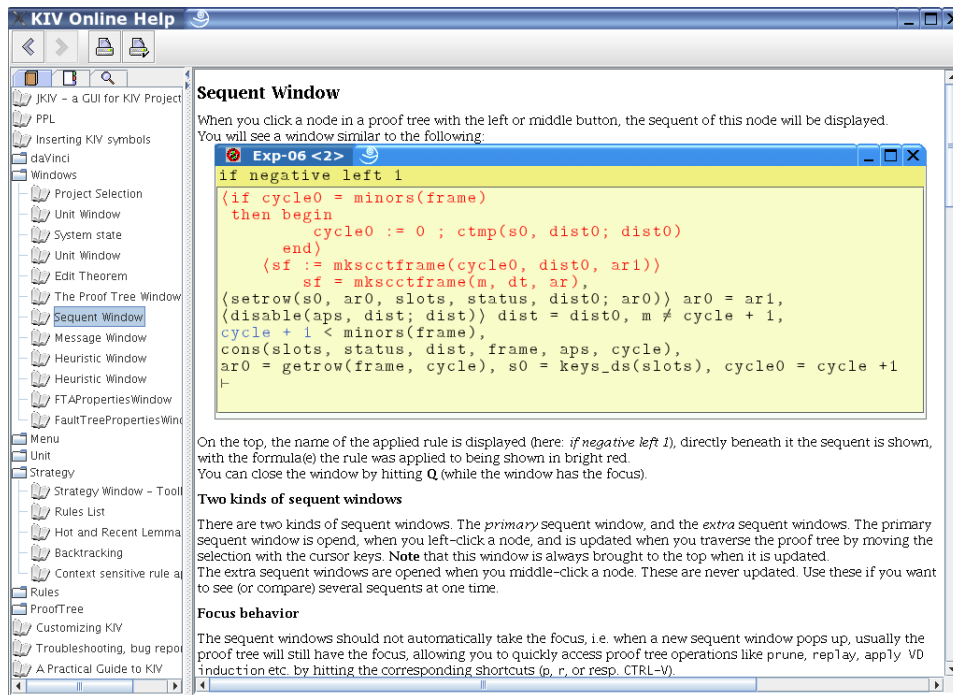


Fig. 12. The Help Window

5 Online Documentation

The KIV system has a hypertext documentation that explains the features of the system and the usage of the GUI. This documentation is available as online documentation directly in the GUI. Figure 12 gives an impression of

the online documentation.

6 Conclusion

The GUI is an integral part of the KIV system. Without such an elaborate GUI mastering large verification challenges would not be possible. A good GUI for a theorem prover improves the productivity of a proof engineer in the same way the productivity of a programmer is improved by a good GUI for his programming language.

In this paper we have given an overview over the concepts and features of the GUI developed for the KIV verification system. The central concepts presented include:

- Graphical visualization of structured specifications combined with an elaborate correctness management.
- Display and direct manipulation of proof trees.
- Context-sensitive application of proof rules.

References

- [Asp00] David Aspinall. Proof General: A Generic Tool for Proof Development. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 38–42. Springer, 2000.
- [BDRS02] M. Balsler, C. Duelli, W. Reif, and G. Schellhorn. Verifying concurrent systems with symbolic execution. *Journal of Logic and Computation*, 12(4):549–560, 2002.
- [BRS⁺00] M. Balsler, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, number 1783 in LNCS, pages 363–366. Springer-Verlag, 2000.
- [BRSS99] M. Balsler, W. Reif, G. Schellhorn, and K. Stenzel. KIV 3.0 for Provably Correct Systems. In *Current Trends in Applied Formal Methods*, LNCS 1641. Boppard, Germany, Springer-Verlag, 1999.
- [FW94] M. Fröhlich and M. Werner. Demonstration of the interactive graph visualization system *daVinci*. In R. Tamassia and I. Tollis, editors, *DIMACS Workshop on Graph Drawing '94. Proceedings*, LNCS 894, Berlin, 1994. Princeton (USA), Springer. <http://www.informatik.uni-bremen.de/~davinci/>.

- [Gor88] M. Gordon. HOL: A Proof Generating System for Higher-order Logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification and Synthesis*. Kluwer Academic Publishers, 1988.
- [HHR86] R. Hähnle, M. Heisel, W. Reif, and W. Stephan. An Interactive Verification System Based on Dynamic Logic. In J. Siekmann, editor, *8th International Conference on Automated Deduction. Proceedings*. Springer LNCS 230, 1986.
- [HRS89] M. Heisel, W. Reif, and W. Stephan. A Dynamic Logic for Program Verification. In A. Meyer and M. Taitlin, editors, *Logical Foundations of Computer Science*, LNCS 363, pages 134–145, Berlin, 1989. Logic at Botik, Pereslavl-Zalessky, Russia, Springer.
- [KM97] M. Kaufmann and J Moore. An industrial strength theorem prover for a logic based on common lisp. *IEEE Transactions on Software Engineering*, 23(4), April 1997.
- [MH97] N. Merriam and M. Harrison. What is wrong with GUIs for theorem provers. In *Proceedings of User Interfaces for Theorem Provers (1997)*, 1997.
- [MP94] Z. Manna and A. Pnuelli. Temporal verification diagrams. In M. Hagiya and J. Mitchell, editor, *International Symposium on Theoretical Aspects of Computer Software*, volume 789, pages 726–765. Springer Verlag, 1994.
- [NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- [ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
- [RSS97] W. Reif, G. Schellhorn, and K. Stenzel. Proving System Correctness with KIV 3.0. In *14th International Conference on Automated Deduction. Proceedings*. Townsville, Australia, Springer LNCS 1249, 1997.
- [RSSB98] W. Reif, G. Schellhorn, K. Stenzel, and M. Balsler. Structured specifications and interactive proofs with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications*, pages 13 – 39. Kluwer Academic Publishers, Dordrecht, 1998.
- [SHB⁺99] Jörg H. Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. *LOUT: Lovely OMEGA User Interface*. *Formal Asp. Comput.*, 11(3):326–342, 1999.
- [Ste04] K. Stenzel. A formally verified calculus for full Java Card. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Algebraic Methodology and*

Software Technology (AMAST) 2004, Proceedings, Stirling Scotland, July 2004. Springer LNCS 3116.