The *Theorema* Environment for Interactive Proof Development

Florina Piroi[1] and Temur Kutsia*[2]

[1] Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences, A-4040 Linz, Austria
`Florina.Piroi@oeaw.ac.at`

[2] Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria
`Temur.Kutsia@risc.uni-linz.ac.at`

Abstract. We describe an environment that allows the users of the *Theorema* system to flexibly control aspects of computer-supported proof development. The environment supports the display and manipulation of proof trees and proof situations, logs the user activities (commands communicated with the system during the proving session), and presents (also unfinished) proofs in a human-oriented style. In particular, the user can navigate through the proof object, expand/remove proof branches, provide witness terms, develop several proofs concurrently, proceed step by step or automatically and so on. The environment enhances the effectiveness and flexibility of the reasoners of the *Theorema* system.

1 Introduction

In general terms, it is agreed that mechanized theorem proving is about using computers to find a formal proof [1]. A rough classification of theorem provers divides them in automatic provers, where close to no human assistance is needed, and interactive provers, which require human assistance in developing the proof [18]. An extensive list of both automatic and interactive provers can be inspected at [3].

The goal of the *Theorema* project [8] is to provide support to the entire process of mathematical theory exploration. By default, *Theorema* tries to solve given reasoning problems automatically. However, since many mathematical theorems are hard to prove completely automatically, it is helpful to have an environment that supports interactive reasoning. This paper describes the current status of an experimental version of such an environment in *Theorema*. Although *Theorema* has support also for computing and solving, the environment is currently used only for proof development. It allows a finer grained interaction between a human user and the system. The environment aims at three groups of users. For the first one the environment has a didactic value: it can be used

* Temur Kutsia has been supported by the Austrian Science Foundation (FWF) under Project SFB F1302 and F1322.

to train formal proving. In the second group are those users who are already familiar with formal proving techniques and with the details of *Theorema*. For them, the environment enriches the proving power of the system by allowing them to use their creative ideas and intuition (for example, providing witness terms). The third group of users is the *Theorema* developers group, for which the environment is used as a tool for testing the provers that are still in development.

The first attempts to integrate interactivity into *Theorema* are described in [9] and some of those ideas were a starting point for the current interactive environment. Prior to this work, in [27] it is shown how interactive proving was to be integrated in the architecture of *Theorema*, but no implementation was done. Another attempt to provide user-system interaction is described in [17] and [16].

Shortly, our main contribution to the previous implementations are improved proof tree and proof situation management, a schematic representation of the proof tree, and multiple interconnected views of the underlying data structures.

This paper is organized as follows: In Section 2 we discuss general requirements for an interactive proof development environment. Section 3 gives an overview of the *Theorema* system and describes the experimental implementation of the *Theorema* interactive environment. In Section 4 an example of interactive proof development in *Theorema* is given. We overview some related work in Section 5 and we end with conclusions and future work in Section 6.

2 Requirements for an Interactive Environment for Proof Development

Design principles for interfaces to (interactive) provers, as well as the functionalities such interfaces should offer, have already been formulated by a number of authors; see [6, 12, 13, 28]. We do not intend to give yet another set of principles, but we will just gather user actions that correspond to the already formulated principles and classify them into logical and abstract interaction actions.

Our classification is based on the levels of abstraction described in [1]: a logical, an abstract interaction, and a concrete interaction level are considered to be necessary to characterise the interaction with an automated reasoning system. In this paper, we do not consider the actions at the concrete interaction level. We give, however, some considerations in this respect in Section 3.4. (For more usage and implementation details see [21].)

At the logical level the user actions are sketched only in terms of logical concepts [1], like for example the activity of reducing a mathematical expression to its canonical form. Other actions that are to be included in the class of logical level activities are providing witness terms, adding and removing formulae from the list of formulae used during a reasoning session, selecting formulae and/or proof strategies that are to be used in the next reasoning chain. At this level, a mathematician using an automated theorem prover must be given the possibility to save and restore proving sessions, to abandon proof attempts, and to work on several partial proofs at the same time. Additionally, it is also important that the user has quick access to information relevant to the development of the proof and that she is not burdened with unnecessary information. Proof navigation should be available and as simple as possible. A bonus for any interactive system is the presence of a comprehensive help system which gives users hints on how to use the system's commands and answers to their actions.

These activities do not assume having a good knowledge about automated reasoners, but only basic knowledge on doing proofs, which any user with a background in mathematics is supposed to have.

At the abstract interaction level users manipulate visual objects in order to communicate with the system. At this level no implementation details are considered, this is done at the third level, the concrete interaction level (which will not be discussed in this paper).

To realize the logical actions of the interactive reasoning systems, at the abstract interaction level, we have to provide means for structure manipulation and we should make use of objects representing logical knowledge. For example a directed graph structure can be used for visualising and navigating in a proof or for representing hierarchically composed theories of mathematical knowledge. Manipulating such structures requires maintaining connections between objects as data structures and their displays (tree representation or textual, user-friendly proof explanation). In order to facilitate users to store and restore proving sessions, the designer of an interactive proving system will have to provide mechanisms for script management to record, store, and maintain a history of user actions. Commands for developing proofs have to offer default behaviour in case they are incompletely specified. Articulating commands by various means (mouse clicks, typing, etc.) is also a feature which interactive proving systems should supply.

Finally, we remark that an action performed at the logical interaction level can be seen as an explanation and motivation for an action at the abstract level [1].

3 *Theorema*'s Interactive Environment

3.1 An Overview of the *Theorema* System

Theorema[3] is implemented in the programming language of the Mathematica system. The development is carried out since mid nineties under the guidance of Bruno Buchberger. A user exploring theories using *Theorema* interacts (automatically or semiautomatically) with three blocks of system components: reasoners, organizational tools, and libraries of mathematical knowledge [8].

Basic building blocks of the system's reasoners are inference rules that operate on reasoning situations—goals and knowledge bases. The rules are implemented as Mathematica functions. They can be grouped into modules and then combined into reasoners by various strategies. The reasoning process is guided by a *common search procedure*. The output of this procedure is a *global reasoning*

[3] See `www.theorema.org`

object that follows a common structure which allows a homogeneous display of the output independent of which reasoner was used. The object is an AND-OR tree which, during the search procedure, is expanded top-down, and the root contains the original reasoning situation. Terminal nodes on successful or failed branches and non-terminal nodes are labeled by (certain encodings of) the reasoning steps performed. Terminal nodes on the other branches are labeled by reasoning situations.[4]

The language of *Theorema* is an untyped higher order language extended with sequence variables. Type (or sort) information is in general handled by unary predicates or sets (if one decides to work in a set theory). However, particular reasoners may implement rules to deal with such information in a special way.

Theorema advocates efficient reasoning in special theories—like geometry, analysis, combinatorics—using algebraic algorithms as black box inference rules. For this purpose several special reasoners have been developed, e.g. the PCS prover [7] (standing for 'Prove Compute Solve') which implements a heuristics for elementary analysis and uses Collins's Cylindrical Algebraic Decomposition algorithm [10] as a solver. Another example of a special reasoner is the solver and simplifier for two-point linear boundary value problems [22]. *Theorema* currently contains 19 reasoners and is linked to 11 external reasoning systems and to the TPTP library; see [8].

During a *Theorema* session reasoners are accessed by a call of the form

$$\textit{Reason}[\textit{entity}, \text{ using} \rightarrow \textit{knowledge-base}, \text{ by} \rightarrow \textit{reasoner}, \textit{options}],$$

where *Reason* is Prove, Compute, or Solve; *entity* is the mathematical entity to which *Reason* applies, e.g. a proposition in the case of proving or an expression in the case of computing; *knowledge-base* is the knowledge with respect to which the reasoning should be performed; *reasoner* is the concrete (internal or external) reasoner we want to use. There are two groups of *options*: those specific to reasoners, which give means to influence their behaviour, and those that control the general search mechanism and the eventual post-processing tools (presentation, simplification, etc.). For convenience default values for each of the options are available. Information and usages of the available *Theorema* reasoners and options can be displayed with the Mathematica '?*symbol*' command.

In the sequel we concentrate on proof development only, i.e., the concrete reasoners are provers. A sample *Theorema* proving session consists of the following steps. First, Mathematica must be started and then *Theorema* loaded. Next, the knowledge the user wants to use (e.g. *formula*, *knowledge-base*) must be made available to the system. This can be done either by typing it in a Mathematica notebook[5] and evaluating it, or by loading a previously stored file. Finally, the corresponding *Reason* command should be (typed and) evaluated. The output is given in a separate notebook in a pretty-printed, textbook-style syntax.

[4] Those who are familiar with the NuPRL proof object may notice that the *Theorema* reasoning object and the NuPRL proof object are quite similar.

[5] Notebooks are part of the Mathematica front end. They are complete interactive documents combining text, tables, graphics, calculations, and other elements.

If the proof does not succeed the user may re-start the proof search process with different premises (additional knowledge, different options of the used prover, different prover of *Theorema*). However, we would like to have the possibility to guide the proof search routines *during* the proof search. For example, we would like to hint to the prover that it should use certain instances for specific quantified variables at various points in the proof. In the following sections we will describe the tools that support such a user-system interaction.

The components of the *Theorema* interactive interface are working files, windows for displaying messages and logs, and menu-palette windows (toolbars).

The working files are usual Mathematica notebooks in which the users write and store the mathematical knowledge employed in a reasoning session (interactive or not). Special notebooks are The Proof Window, presenting proofs in a user-friendly style, and The Proof Tree Window which shows the tree structure of the proof. These two windows are maintained and updated by the system. By combining selections of cells in The Proof Window, in the working files, selections in The Proof Tree Window, and button clicks on the toolbars users can navigate within the proof, introduce new proof variants, give witness terms, etc.

Whenever an action could not be accomplished, the *Theorema* interactive interface makes use of notification dialogs with short explanation messages on why the action could not be performed. Also, a log window is present, where environment and proof information, actions performed by the user, etc. are displayed. The content of this window can be saved but, at the moment, to restore a proving session the users have to do the actions themselves, as recorded in the stored log, one by one.

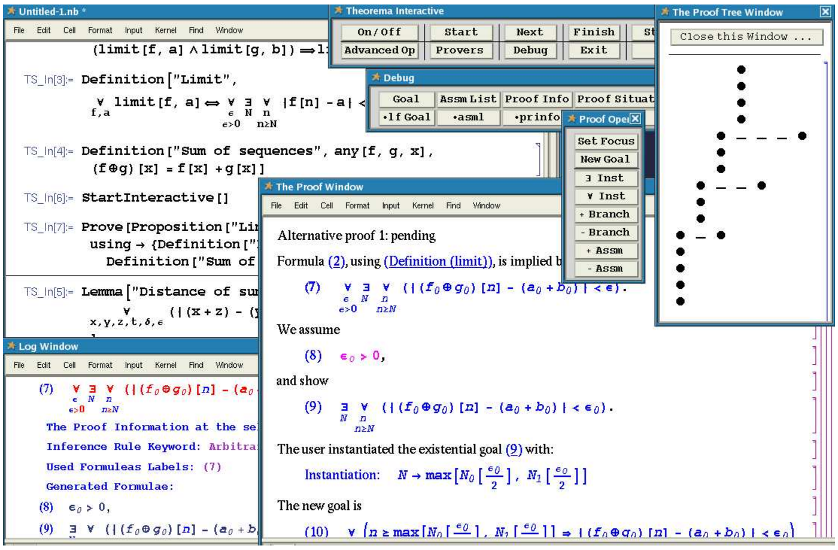
The commands that realize the various actions for the interactive proof development can be articulated either with the help of the toolbars or by typing the commands in the working notebooks and sending them to the Mathematica kernel for evaluation.

3.2 Managing the Proof Tree

In the non-interactive mode, the *Theorema* provers apply the inference rules automatically. The inferences are repeatedly applied until either a proof is obtained or no further inferences can be applied. The users only see the final output of this process. In contrast, when searching for proofs in the interactive environment, the system is compelled to stop after each application of an inference rule, to present the proof produced so far, and to wait for a decision from the user.

In the interactive mode, the proofs are gradually developed starting from an initial proof tree: the root node that contains the proof problem as given by the user (goal formula and assumption formulae, if any) and, additionally, internal information specific to the provers and to the proof search routines of *Theorema*. Initially, the root is an unexplored node, or in *Theorema* terminology: a pending node. The information stored in an unexplored node is called a "proof situation".

The node expansion is done by calling a prover to apply one of its inferences to the node's proof situation. An inference rule application will produce none, one, or more proof situations that are inserted into the proof tree as unexplored children of the now expanded node. The proof search mechanism will add to the information stored in the expanded node a trace of the inference rule application. When a proof under development has more than one unexplored node the user can select which one to expand next. If an expansion action is performed but no prior explicit selection of an unexplored node is made, the system will choose the leftmost unexplored one. Currently we are working on giving users the possibility to see a list of inference rules and proof methods that are applicable to a proof situation, as well as means to choose an inference rule and to select the formulae the rule should be applied on. The proof tree can be displayed in two variants, shown in Fig. 1: an english textual explanation produced from the traces of the inference rule applications, with pretty-printed formulae (in The Proof Window), or a schematic tree representation (in The Proof Tree Window). In both views, users can select nodes in the proof. If the selected node is expanded, the user can choose whether to start a new proof variant by adding a branch to the proof tree at the selection point, or to abandon the reasoning chain below the selection point by removing it. Removing a reasoning chain in the proof and continuing with a different one at the removal point can be seen as an undo operation. Users can also work on different proofs at the same time. The way this can be done in the interactive environment of *Theorema* is the following: For a newly added branch in the proof tree, the user has the possibility to state a different goal that is not necessarily derived from the originally given one. The assumptions available in the prove session when stating the new goal may be used in its proof, but new assumptions can also be added.



```
      (limit[f, a] ∧ limit[g, b]) ⟹ l

TS_In[3]:= Definition["Limit",
               ∀   limit[f, a] ⟺  ∀   ∃  ∀   |f[n] - a| <
              f,a                  ϵ   N  n
                                  ϵ>0    n≥N

TS_In[4]:= Definition["Sum of sequences", any[f, g, x],
              (f⊕g)[x] = f[x] + g[x]]

TS_In[6]:= StartInteractive[]

TS_In[7]:= Prove[Proposition["Li
             using → {Definition["
               Definition["Sum of

TS_In[5]:= Lemma["Distance of su
                    ∀          (|(x + z) - (y
              x,y,z,t,δ,ϵ
```

Alternative proof 1: pending

Formula (2), using (Definition (limit)), is implied b

(7) $\underset{\epsilon>0}{\underset{\epsilon}{\forall}}\ \underset{N}{\exists}\ \underset{n\geq N}{\underset{n}{\forall}} (|(f_0 \oplus g_0)[n] - (a_0 + b_0)| < \epsilon)$.

We assume

(8) $\epsilon_0 > 0$,

and show

(9) $\underset{N}{\exists}\ \underset{n\geq N}{\underset{n}{\forall}} (|(f_0 \oplus g_0)[n] - (a_0 + b_0)| < \epsilon_0)$.

The user instantiated the existential goal (9) with:

Instantiation: $N \to \max[N_0[\frac{\epsilon_0}{2}], N_1[\frac{\epsilon_0}{2}]]$

The new goal is

(10) $\forall (n \geq \max[N_0[\frac{\epsilon_0}{2}], N_1[\frac{\epsilon_0}{2}]] \Rightarrow |(f_0 \oplus g_0)[n] - (a_0 + b_0)| < \epsilon_0)$

(7) $\underset{\epsilon>0}{\underset{\epsilon}{\forall}}\ \underset{N}{\exists}\ \underset{n\geq N}{\underset{n}{\forall}} (|(f_0 \oplus g_0)[n] - (a_0$

The Proof Information at the se

Inference Rule Keyword: Arbitra

Used Formuleas Labels: (7)

Generated Formulae:

(8) $\epsilon_0 > 0$,

(9) $\exists\ \forall (|(f_0 \oplus g_0)[n] - (a_0 + b$

Fig. 1. The *Theorema* interactive environment: the working notebook, the most used menu-palettes, the two proof view windows, and the log window.

At the end of the proof development the proof tree will contain the traces of the user's actions during interactive development of proofs. However, this data structure does not record the order these actions were performed.

3.3 Managing the Proof Situation

In the previous sections we have mentioned that *Theorema* provers' inference rules take as input proof situations, i.e., a goal formula, a list of assumption formulae, and some local context storing facts and additional proof strategy information used by the provers of the system. For example, one such fact is keeping track of which formulae in the list of assumptions were matched against the goal formula. Another example is the storage of the names of the metavariables introduced by certain inference rules and their dependencies.

One of the specific difficulties in algorithmic proof generation is finding appropriate instances (at appropriate moments) for quantified variables. Within the interactive environment of *Theorema* it is possible to give the system witness terms which should be used for certain variables. If the variable is existentially quantified, a user-given instance will be taken into consideration only if the formula in which it occurs is the current goal and the quantifier that binds the variable is the outermost one. For universally quantified variables, a user-given instance is accepted only if the formula occurs in the list of current assumptions and if the quantifier binding the variables is the outermost one.

When we prove a theorem with pen and paper we use, for a start, only few definitions, properties, etc. of the notions occurring in the theorem we try to prove. As we proceed with the proof we usually recall other lemmata, properties, etc. which we use in the attempt to complete the proof. At the same time we may discard some formulae. *Theorema*'s interactive interface does allow users to add and remove formulae from the assumption list of an unexpanded node.

The natural language representation of the proof displays, for each proof step, the result of inference rule applications, namely, which formulae were used, which were generated, the used instantiations (if any), etc. Obviously, this does not reflect all the content of the nodes in the proof. One reason for this is that part of the information stored in the nodes is not relevant for the user, but only for the provers of the system. However, it is often the case that we are interested in the whole content of the proof node. We may want to know, for example, which are the formulae that are or were available when an inference rule was applied. The developers of the *Theorema* system may want to check the prover specific information to help them to develop and improve their provers. For this reasons, the interactive interface to *Theorema* provides access to the additional information stored in a node.

3.4 Comments on Implementation

Until recently, *Theorema* was used mainly in an automated mode and no interaction with the system during the proof search was possible. The first solution chosen to provide interaction was to suspend the execution of the proof search routine after one inference rule application. This was done by starting a Mathematica subprocess that collected the user actions [16].

In the current implementation we have opted for a different, simpler solution. We have introduced a system-global boolean variable which keeps track of the current proving mode (interactive or non-interactive), and a step-counter that controls the number of proof steps to be performed by the proof search routine.

In the non-interactive proving mode, the step-counter variable is ignored and the proof search routine proceeds until either a successful proof is obtained or no inference rule can be applied anymore. In the interactive mode, every time the proof search routine is invoked the step-counter is, first, set to a predefined value. With each inference rule application this value is decreased by one. As soon as the step-counter reaches zero, the proof search routine stops, and returns the proof developed sofar which is, then, presented to the user, in The Proof Window, The Proof Tree Window being also updated. When the user chooses to further expand the proof, the search routine will continue with expanding the left-most unexplored situation in the proof tree, unless otherwise indicated. The default value of the step-counter in the current implementation is set to 1, which means that the proof search stops after one inference rule application.

We mention here two important advantages of this solution. One is that only few modifications of the main proof search routines of the system were necessary: First, a check of the step-counter value was added to the termination conditions of the proof search routine and, second, certain *Theorema* specific variable initialization are by-passed when the proof search is invoked in the interactive mode. (For example, we do not want the proof-tree to be re-initialized, as in the non-interactive mode, but we want to expand it further). The second important advantage of the solution chosen by us is that no alteration of the existing provers of the *Theorema* system had to be done in order to use them for proving in the interactive mode.

Until version 5.1, Mathematica did not have facilities for developing user interfaces. Therefore, with the exception of buttons, the elements of the interactive environment interface do not include drop-down lists, dynamic menus, check boxes, context-sensitive menus, etc. Also, to our knowledge, in Mathematica, we cannot track the mouse actions. In other words, we cannot determine user inputs by tracking the mouse clicks and movements. The solution we have chosen to overcome this difficulty is to (require users to) make selections in notebooks and, on button clicks, manipulate the notebooks in the Mathematica kernel.

Within any open notebook, the front end always maintains a current selection (see [30], Section 2.11.3). Selections can be done by user clicks or by issuing commands from the kernel. Mathematica also provides commands for extracting the content of a selections in a notebook. So we are able to retrieve user input when the user makes selections in notebooks. The retrieved input is passed to the routines implementing the tools of the interactive environment. The routines process the input correspondingly to the tool they implement, e.g. add an assumption to the current proof situation, delete a branch in the proof-tree, provide witness terms, etc.

4 An Example

Assume now that we, as *Theorema* users, want to prove that the limit of the sum of two sequences is the sum of their limits. First, we formalize the proposition and the corresponding definitions in a *Theorema* notebook:

Proposition[“Limit of sum”,

$$\underset{f,a,g,b}{\forall} ((\mathrm{limit}[\mathrm{f},\mathrm{a}] \wedge \mathrm{limit}[\mathrm{g},\mathrm{b}]) \Rightarrow \mathrm{limit}[\mathrm{f} \oplus \mathrm{g}, \mathrm{a}+\mathrm{b}])\ ].$$

Definition[“Limit”,

$$\underset{f,a}{\forall} (\mathrm{limit}[\mathrm{f},\mathrm{a}] \Leftrightarrow (\underset{\substack{\varepsilon\\ \varepsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n\\ n\geq N}}{\forall} |\mathrm{f}[\mathrm{n}]-\mathrm{a}| < \varepsilon))\ ].$$

Definition[“Sum of sequences”,

$$\underset{f,g,x}{\forall} ((f \oplus g)[x] = f[x] + g[x])\ ].$$

This is exactly how it would look in the notebook: *Theorema* has a human-oriented, two-dimensional syntax. Next, we activate the interactive proving mode by evaluating the command `StartInteractive[ ]` which will open the necessary menu-palettes (see Fig. 1). We want to prove the proposition by one of the *Theorema* provers (PredicateProver), using the given definitions. For this we type, in a working notebook, the corresponding `Prove` command, as below, select it, and press the `Start` button on the *Theorema* Interactive palette (see Fig. 1).

```
Prove[Proposition[“Limit of sum”],
      using→{Definition[“Limit”], Definition[“Sum of sequences”]},
      by→PredicateProver]
```

The system will show the user The Proof Window with the following content, where ‘Pending proof of (*formula_label*)’ represents an unexpanded node:

Prove:

(Proposition(Limit of sum))

$$\underset{f,a,g,b}{\forall} ((\mathrm{limit}[\mathrm{f},\mathrm{a}] \wedge \mathrm{limit}[\mathrm{g},\mathrm{b}]) \Rightarrow \mathrm{limit}[\mathrm{f} \oplus \mathrm{g}, \mathrm{a}+\mathrm{b}])$$

under the assumptions:

(Definition(Limit)) $\underset{f,a}{\forall} (\mathrm{limit}[\mathrm{f},\mathrm{a}] \Leftrightarrow (\underset{\substack{\varepsilon\\ \varepsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n\\ n\geq N}}{\forall} |\mathrm{f}[\mathrm{n}]-\mathrm{a}| < \varepsilon))$,

(Definition(Sum of sequences)) $\underset{f,g,x}{\forall} ((f \oplus g)[x] = f[x] + g[x])$.

Pending proof of (Proposition(Limit of sum)).

Here we can simply proceed by clicking the `Next` button. The prover applies the first rule applicable to the current proof situation (∀-Right rule). In The Proof Window the last line (pending proof) is replaced by the following output:

For proving (Proposition(Limit of sum)) we take all variables arbitrary but fixed and prove:

$$(1) \quad \mathrm{limit}[\mathrm{f_0}, \mathrm{a_0}] \wedge \mathrm{limit}[\mathrm{g_0}, \mathrm{b_0}] \Rightarrow \mathrm{limit}[\mathrm{f_0} \oplus \mathrm{g_0}, \mathrm{a_0} + \mathrm{b_0}]$$

Pending proof of (1).

After several default steps this proof attempt will fail. The reason the proof fails is manifold. The main one, however is that the knowledge we started with is not sufficient for proving the goal formula. Secondly, the prover we have used is not strong enough and we would like to use a different one that, implicitly, uses some special knowledge on real numbers and, in addition, applies a particular strategy for handling formulae with alternating quantifiers. Therefore, we undo the proof, with the help of the `-Branch` button, and start again by using the Pcs prover. (We could also have started an alternative proof by adding a branch at a properly chosen point in the proof tree, using the `+Branch` button, and by selecting another prover to continue with, e.g., Pcs. In this way the previous failed attempt would still be present in the proof tree, giving us the possibility to see how different provers act on the same proof problem.)

From the previous failed proof attempt we, as humans, conclude that additional knowledge about absolute values and distances between points may help:

Lemma["Distance of sum",

$$\underset{x,y,z,t,\delta,\varepsilon}{\forall} (|(x+z)-(y+t)| < (\delta+\varepsilon)) \Leftrightarrow (|x-y| < \delta \wedge |z-t| < \varepsilon) \,].$$

After several proving steps the content of The Proof Window is:

Prove: ... (The initial proof problem is omitted for space reasons.)
We assume

$$(1) \quad \mathrm{limit}[\mathrm{f_0}, \mathrm{a_0}] \wedge \mathrm{limit}[\mathrm{g_0}, \mathrm{b_0}]$$

and show

$$(2) \quad \mathrm{limit}[\mathrm{f_0} \oplus \mathrm{g_0}, \mathrm{a_0} + \mathrm{b_0}].$$

Formula (1.1), by (Definition(Limit)), implies:

$$(3) \quad \underset{\substack{\varepsilon \\ \varepsilon>0}}{\forall} \; \underset{N}{\exists} \; \underset{\substack{n \\ n\geq N}}{\forall} \; |f_0[n] - a_0| < \varepsilon.$$

By (3), we can take an appropriate Skolem function such that

$$(4) \quad \underset{\substack{\varepsilon \\ \varepsilon>0}}{\forall} \; \underset{\substack{n \\ n\geq N_1[\varepsilon]}}{\forall} \; |f_0[n] - a_0| < \varepsilon.$$

Formula (1.2), by (Definition(Limit)), implies:

$$(5) \quad \underset{\substack{\varepsilon \\ \varepsilon>0}}{\forall} \; \underset{N}{\exists} \; \underset{\substack{n \\ n\geq N}}{\forall} \; |g_0[n] - b_0| < \varepsilon.$$

By (5), we can take an appropriate Skolem function such that

(6) $\underset{\varepsilon>0}{\underset{\varepsilon}{\forall}}\ \underset{n\geq N_2[\varepsilon]}{\underset{n}{\forall}}\ |g_0[n]-b_0|<\varepsilon.$

Formula (2), using (Definition(Limit)), is implied by:

(7) $\underset{\varepsilon>0}{\underset{\varepsilon}{\forall}}\ \underset{N}{\exists}\ \underset{n\geq N}{\underset{n}{\forall}}\ |(f_0\oplus g_0)[n]-(a_0+b_0)|<\varepsilon.$

We assume

(8) $\varepsilon_0>0$

and show

(9) $\underset{N}{\exists}\ \underset{n\geq N}{\underset{n}{\forall}}\ |(f_0\oplus g_0)[n]-(a_0+b_0)|<\varepsilon_0.$

At this point we, as users, decide to influence the proof by providing an appropriate witness term for N. Selecting the formula (9) and clicking the button $\exists$ `Inst`, a dialog window opens where the witness term can be specified. We type in $\max[N_1[\varepsilon_0/2],N_2[\varepsilon_0/2]]$, and the proof proceeds:

Instantiation: $N\to\max[N_1[\frac{\varepsilon_0}{2}],N_2[\frac{\varepsilon_0}{2}]]$.

The current goal is

(10) $\underset{n}{\forall}\ ((n\geq\max[N_1[\frac{\varepsilon_0}{2}],N_2[\frac{\varepsilon_0}{2}]])\Rightarrow|(f_0\oplus g_0)[n]-(a_0+b_0)|<\varepsilon_0).$

We assume

(11) $n_0\geq\max[N_1[\frac{\varepsilon_0}{2}],N_2[\frac{\varepsilon_0}{2}]]$

and show

(12) $|(f_0\oplus g_0)[n_0]-(a_0+b_0)|<\varepsilon_0.$

Formula (12), using (Definition(Sum of sequences)), is implied by:

(13) $|(f_0[n_0]+g_0[n_0])-(a_0+b_0)|<\varepsilon_0.$

Formula (13), using (Lemma(Distance of sum)), is implied by:

(14) $\underset{\delta+\varepsilon=\varepsilon_0}{\underset{\delta,\varepsilon}{\exists}}\ (|f_0[n_0]-a_0|<\delta\wedge|g_0[n_0]-b_0|<\varepsilon).$

Here we interact again by instantiating δ and ε with $\varepsilon_0/2$.

Instantiation: $\delta\to\frac{\varepsilon_0}{2},\ \varepsilon\to\frac{\varepsilon_0}{2}$.

The current goal is

(15) $\frac{\varepsilon_0}{2}+\frac{\varepsilon_0}{2}=\varepsilon_0\wedge|f_0[n_0]-a_0|<\frac{\varepsilon_0}{2}\wedge|g_0[n_0]-b_0|<\frac{\varepsilon_0}{2}.$

Formula (15) is implied by

(16) $|f_0[n_0]-a_0|<\frac{\varepsilon_0}{2}\wedge|g_0[n_0]-b_0|<\frac{\varepsilon_0}{2}.$

Formula (16), by (4) is implied by

(17) $\frac{\varepsilon_0}{2} > 0 \wedge n_0 \geq N_1[\frac{\varepsilon_0}{2}] \wedge |g_0[n_0] - b_0| < \frac{\varepsilon_0}{2}$,

which by (6) is implied by

(18) $\frac{\varepsilon_0}{2} > 0 \wedge n_0 \geq N_1[\frac{\varepsilon_0}{2}] \wedge n_0 \geq N_2[\frac{\varepsilon_0}{2}]$.

Formula (18), by (8), is implied by

(19) $n_0 \geq N_1[\frac{\varepsilon_0}{2}] \wedge n_0 \geq N_2[\frac{\varepsilon_0}{2}]$.

Here we notice that another assumption is needed that we add immediately to the knowledge base used by the proof search routine. To add the assumption we use the `+Assm` button. The proof proceeds then as follows:

The user added the assumption:

(Lemma(Max))

$$\underset{m,M_1,M_2}{\forall} (m \geq \max[M_1, M_2] \Rightarrow (m \geq M_1 \wedge m \geq M_2)).$$

Formula (19), by (Lemma(Max)), is implied by

(20) $n_0 \geq \max[N_1[\frac{\varepsilon_0}{2}], N_2[\frac{\varepsilon_0}{2}]]$.

Formula (20) is proved because it is identical to (11).

To summarize, this example demonstrates various ways of interaction with the system: cutting a branch and backtracking, changing the prover, adding assumptions, and providing witness terms.

5 Related Work

A concise historical overview of interactive systems is given in [19]. Though in most of the cases the design principles listed in [6, 12, 13] or [28] were not specifically followed, many interfaces have common functionalities. We briefly describe some of these systems (mathematical assistants), insisting on those features similar to the ones present in *Theorema*. For more details on the described systems we direct the reader to the literature (e.g. [29], or the forthcoming issue on Mathematics Assistance Systems of the Journal of Applied Logic [4]).

One of the interactive systems with the largest pool of users is the Hol system [14], now at version 4. It is an environment for interactive theorem proving in higher-order logic and has a wide variety of uses from formalizing mathematics (see for example [15]) to verification of industrial hardware. It has high degree of programmability through the meta language Ml which allows extending the system to provide more functionality. Thus, packages for proof tree administration, goal tracking, script save and replay, etc. are available within Hol. As a theorem proving system, Hol has a command-line interface. As in *Theorema*, the system permits adding assumptions to a proof in development, but no removing of formulae is possible. The proofs are done in a goal directed style but tacticals that do forward inferencing are also present in the system. If users decide that wrong proof steps were done, they can undo to the previous proof state. There is also a restart possibility by backtracking to the root. Switching between goals is possible both in HOL and in *Theorema*. Several graphical interfaces were implemented for the system, like Tk-HOL [26], `xhol` [23], and Emacs modes are widely used. The interfaces provide theory browsing and searching, graphical views of the proof state (similar to the schematic proof tree representation in *Theorema*'s interactive environment), etc.

The Isabelle system [20] is a generic proof assistant which allows defining different logical calculi and using them for proving. It is closely integrated with the Proof General [2] for editing and developing proofs and, similar to the HOL system, allows proof storage and replay, undo and revert operations, proof states display, etc. (Proof General [2] is a generic tool for proof development that provides a uniform interface and interaction mechanism not only for Isabelle but for other proof assistants as well, like Coq, LEGO, and, experimentally, with HOL, ACL2 and λClam.) However, the system does not maintain a proof object and only one view of the proof is available (the one shown in the Proof General window). In its latest version, Isabelle provides a tool for searching theorems in the system's library of theories by simple patterns.

Coq [5], another logical framework system, is a proof assistant for the Calculus of Inductive Constructions. It allows the interactive construction of formal proofs, and also the manipulation of functional programs. A variety of user interfaces are provided for it. For example CoqIde is a graphical user interface based on gtk which allows proof tree navigation, structural editing of formulae and commands, and has an autocomplete facility for command articulation. Pcoq, CtCoq, and Proof General are other interfaces for Coq. Lately, an integration of Coq into TeXmacs is also available.

Ωmega [24] is an interactive proof development system. The system has two main components: a proof planner, and an integrated collection of tools for formulating problems, proving subproblems, and proof presentation. LΩui is an interface for Ωmega which combines features for graphical display of proofs as a graph, hypertext facilities for term browsing, proof and proof plan presentation in natural language. It also has an editor for adding and maintaining the knowledge base, and a command suggestion mechanism; see [25].

NUPRL [11] supports the interactive creation of proofs, formulas, and terms. Based on Martin-Löf type theory, it is a system for implementing mathematics. NUPRL has a multi-window graphical environment and a keyboard-based proof navigation tools.

6 Future Work and Conclusions

The current status of the *Theorema* interactive environment allows users to select a proof situation in the proof; inspect the content of a selected proof situation; add or remove assumptions in a selected proof situation; suggest witness terms; add or remove branches in the proof tree; model concurrent proof development; select one from several provers to continue the proof, eventually change its options; make the system expand the proof by one inference rule application; ask the system to automatically complete the proof.

In future versions of the environment we plan to include a tool for inference rule selection. Namely, for a selected proof situation, the tool should present, on request, a list of applicable inference rules. The user can select one or more inferences to be applied in the next step. Selecting more than one inference from the list means that the user intends to investigate several proof alternatives for the given proof situation, one alternative for each inference rule selected. However, to implement such a inference selection tool, important modifications of the *Theorema* system are necessary. For example, inference rules need to be uniquely identifiable among all the inferences of the system. We also plan to include facilities for specifying tactics.

Other tools we plan to include in the environment are possibilities to store and load proof sessions, extracting proof strategies from a proof session. A variant of the latter tool can be used to help the developers of the Theorema provers compose new provers based on the sequence of inferences used in an interactive proof session. To achieve this, we will have to analyze the proofs obtained in the interactive mode, in order to extract the relevant proof steps and inference rules.

References

1. J. S. Aitken, P. D. Gray, T. F. Melham, and M. Thomas. Interactive theorem proving: An empirical study of user activity. *J. Symb. Comp.*, 25(2):263–284, 1998.
2. D. Aspinall. Proof general: A generic tool for proof development. In *Proc. TACAS'2000*, volume 1785 of *LNCS*, pages 38–42. Springer, 2000.
3. A database of mechanized reasoning systems.
`http://www-formal.stanford.edu/clt/ARS/systems.html`.
4. C. Benzmüller, editor. *Mathematics Assistance Systems*. Special Issue of J. Applied Logic. Elsevier, 2005. To appear.
5. Y. Bertot and P. Castèran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
6. R. Bornat and B. Sufrin. Jape's quiet interface. In *Proc. Second Workshop on User Interfaces for Theorem Provers (UITP'96)*, 1996.
7. B. Buchberger. The PCS prover in Theorema. In R. Moreno-Díaz, B. Buchberger, and J. Luis Freire, editors, *EUROCAST'01*, volume 2178 of *LNCS*, pages 469–478. Springer, 2001.
8. B. Buchberger, A. Crăciun, T. Jebelean, L. Kovács, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. *Theorema*: Towards computer-aided mathematical theory exploration. *J. Applied Logic*, 2005. To appear in [4].
9. B. Buchberger, T. Jebelean, and D. Văsaru. Theorema: A system for formal scientific training in natural language presentation. In T. Ottmann and I. Tomek, editors, *Proc. ED–MEDIA/ED–TELECOM'98*, pages 174–179, 1998.
10. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conference on Authomata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

11. R. Constable. *Implementing Mathematics Using the Nuprl Proof Development System.* Prentice Hall, 1986.
12. K. Eastaughffe. Support for interactive theorem proving: Some design principles and their application. In *Proc. 4th Workshop on User Interfaces for Theorem Provers (UITP'98)*, 1998.
13. J. A. Goguen. Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing*, 11(3):272–301, 1999.
14. M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.
15. T. C. Hales. A computer verification of the Kepler conjecture. In *Proceedings of the ICM*, volume 3, pages 795–804, Beijing, China, 2002.
16. F. Kossak. An interface for interactive proving with the mathematical software system Theorema. Technical Report 99–19, RISC, University of Linz, 1999.
17. F. Kossak and K. Nakagawa. User–System Interaction within Theorema. Technical Report 99-37, RISC, University of Linz, 1999.
18. R. Moten. Just the facts, Jack: Truths and myths of automated theorem provers. *Contemporary Mathematics*, 252:31–48, 1999.
19. T. Nipkow and W. Reif. An introduction to interactive theorem proving. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications.* Kluwer Academic Publishers, 1998.
20. L. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.
21. F. Piroi. *Tools for Using Automated Provers in Mathematical Theory Exploration.* PhD thesis, RISC, Johannes Kepler University, Linz, Austria, August 2004.
22. M. Rosenkranz. A new symbolic method for solving linear two-point boundary value problems on the level of operators. *J. Symb. Comp.*, 39(2):171–199, 2005.
23. T. Schubert and J. Biggs. A tree-based, graphical interface for large proof development. In *HOL Theorem Proving and Its Applications, 7th Int. Workshop*, 1994.
24. J. Siekmann, C. Benzmüller, and S. Autexier. Computer Supported Mathematics with *Ω*MEGA. *J. Applied Logic*, 2005. To appear in [4].
25. J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. L*Ω*UI: Lovely *Ω*mega User Interface. *Formal Aspects of Computing*, 11(3):326–342, 1999.
26. D. Syme. A new interface for HOL—ideas, issues and implementation. In E. T. Schubert, P. J. Windley, and J. Alves-Foss, editors, *HOL Theorem Proving and Its Applications, 8th Int. Workshop*, number 971 in LNCS, pages 324–339. Springer, 1995.
27. E. Tomuţa. *An Architecture for Combining Provers and its Applications in the Theorema System.* PhD thesis, Johannes Kepler University, Linz, July 1998.
28. N. Völker. Thoughts on requirements and design issues of user interfaces for proof assistants. *Electronic Notes in Theoretical Computer Science*, 103:139–159, 2004.
29. F. Wiedijk. Comparing mathematical provers. In A. Asperti, B. Buchberger, and J. H. Davenport, editors, *Proc. MKM'03*, volume 2594 of *LNCS*, pages 188–202. Springer, 2003.
30. S. Wolfram. *The Mathematica Book.* Wolfram Media Inc., 5th edition, 2003.