Electronic Notes in Theoretical Computer Science 23 No. 3 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume23.html 14 pages

User - System Interaction Within *Theorema* [1]

- Extended Abstract -

Koji Nakagawa and Felix Kossak

Research Institute for Symbolic Computation
Linz University
A4232 Castle of Hagenberg, Austria
E-mail: {Koji.Nakagawa, Felix.Kossak}@risc.uni-linz.ac.at

Abstract

The purpose of *Theorema* is to provide various ranges of users, from students to experts of mathematics, with an integrated environment for computing, solving, and proving. To make the system convenient, it is necessary to have sophisticated user - system interaction. In this paper we discuss the user - system interaction of *Theorema* from two aspects, namely remote interaction with *Theorema* over the internet and user - prover interaction within *Theorema*. The remote interaction increases the possibilities of using the system in various situations and of collaborative work. The user - prover interaction allows us to combine the proving power of fully automated provers and human creativity as well as to train students.

1 Introduction

The *Theorema* system (http://www.theorema.org) provides an integrated interface for computing, solving, and proving. *Theorema* was designed in early papers by Bruno Buchberger, see e.g. [1] and [2]. A survey on *Theorema* is also contained in [3]. *Theorema* is implemented on the top of *Mathematica* (http://www.wolfram.com).

In the present paper, we describe possible ways of user - system interaction within *Theorema* on two different levels:

- remote interaction with *Theorema* over the internet,
- user - prover interaction within *Theorema*.

Of course, arranging knowledge bases and the calls of various provers for proving desired goal statements is also an important user - system interac-

[1] This research is supported by the Austrian Science Foundation (FWF)—project FO-1302 (SFB).

tion. Strategies for this type of "coarse grain" interaction with *Theorema* are discussed in the talk by Bruno Buchberger at the same workshop, see [4].

2 Remote Interaction with Theorema over the Internet

We provide two ways of access to *Theorema* over the internet which are distinguished by the kind of interface between a user and a remote server that controls *Theorema*. The first way is built on the assumption that *Mathematica* is available at the site of the user. The second way is possible for users who do not have *Mathematica* but have, at least, access to a web browser. Both ways of access are meant for potential users who want to try out demo versions of *Theorema* before actually purchasing the system or, for example, for demos of *Theorema* in a classroom environment for students.

2.1 Mathematica Interface

Obviously the first approach is quite natural because *Theorema* is implemented on top of *Mathematica*. What the users have to do is just to download a *Mathematica* package file `'Client.m'` from the *Theorema* home page. The package file contains some functions which enable the users to connect to a remote *Theorema* server.

The connection procedure is the following:
In a new *Mathematica* session, the library file has to be loaded:

```
<<Client`
```

To connect to a *Theorema* server, the following function call is needed:

```
ConnectTheorema[]
```

After the execution of the function call, the user can use *Theorema* in the same way as the original *Theorema* system without installing it at the site of the user. When the user wants to finish the *Theorema* session, the user can disconnect the session by the following function call:

```
DisconnectTheorema[]
```

2.2 Web Interface

Nowadays most computers are equipped with web browsers (Netscape, Internet Explorer etc.). Hence, for wide access, it is a good strategy for *Theorema* to provide a web interface. However, under the web interface, we can not expect as nice interactivity as under *Mathematica*. *Mathematica* has a sophisticated way to input mathematical expressions and display them in high quality. By using Java one might be able to write an application comparable to the *Mathematica* expression handling capabilities. However, this would take an enormous effort. Therefore, as an immediate solution, we use the facilities of HTML to simulate some of the functionality of the *Mathematica* expression handling capabilities. With the planned availability of MathML (http://www.w3.org/Math/), our task will be greatly facilitated. However, still, remote input of *Theorema* expressions will need some of the functionality we describe in the current paper.

This web interface is appropriate for testing examples by using a previously defined knowledge base or demonstrating mathematical results rather than inputting large theories, because interactivity is low due to the slow connection to the web server.

2.2.1 Constructing Knowledge Bases

A *Theorema expression* is a (higher-order) predicate logic expression which has a fixed syntax. *Theorema formal text* is hierarchically structured text composed of labelled *Theorema* expressions with a key word like `Theorem`, `Definition`, or `Theory`. *Theorema* formal text is used to compose knowledge bases. One can then use these knowledge bases for proving, computing, and solving by the commands `Prove`, `Compute`, and `Solve`. Some examples are shown in the rightmost side of Fig. 1 and Fig. 2.

To input and display *Theorema* expressions and formal text we use the basic facilities of HTML (⟨FORM⟩ tag for input, ⟨TABLE⟩ tag for display). We provide a web page in which the user can input arbitrarily complex *Theorema* expressions and *Theorema* formal text by composing it from smaller expressions.

The web page for constructing knowledge bases is shown in Fig. 1. It consists, from left to right, of three areas: the template area, the work area, and the submission area. If the area is too small to display everything, scrollbars show up.

- Template Area
  In this area the user can type simple expressions and compose complex expressions by selecting the structure of the expression. The user types simple mathematical expressions into the input boxes and presses the button `'ok'`. The expression is then transferred to the work area. Characters like $\wedge$, $\vee$, $\Rightarrow$, etc. can be represented by composed ascii characters like `'&&'`, `'||'`, `' =>'` or characters with names of the form `'\[And]'`, `'\[Or]'`, `'\[Implies]'` as used in *Mathematica*. The expressions sent to the work area can then be referred to by `'%n'`, where `n` is a sequential number generated as the label of an expression in the work area. In the same way, *Theorema* formal text can be composed. *Theorema* formal text describes both the knowledge base and the goal of the current proof, solve, or compute problem.

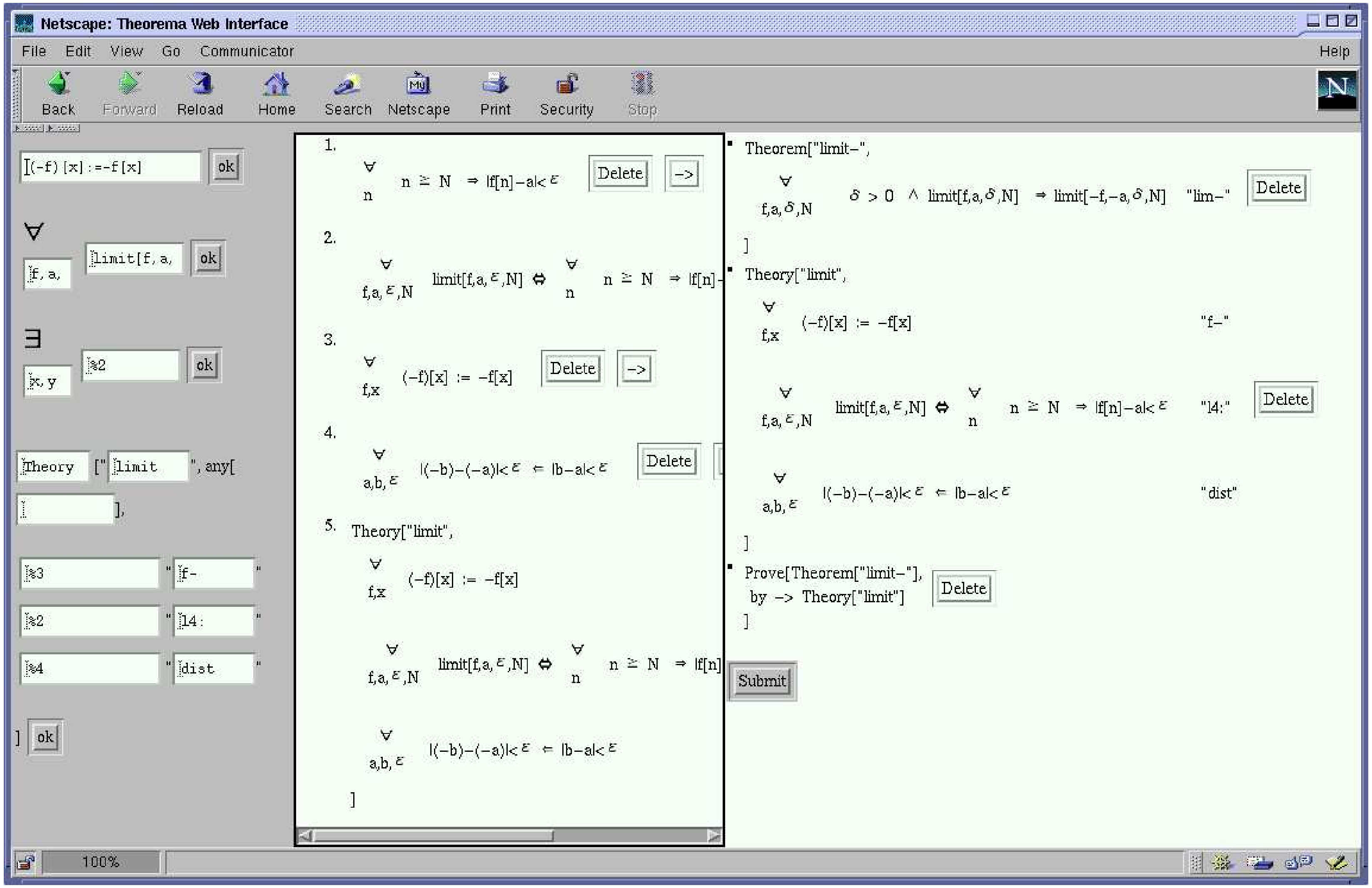


Fig. 1. Screen image of the web interface

- Work Area
  Here one can save some *Theorema* expressions to compose more complicated expressions. As described above, each expression has a unique number and it can be referred to by this number to construct more complex expressions in the template area. An expression can be deleted if it becomes unnecessary by pressing the button with the label `'Delete'`. An expression is sent to the submission area by pressing the button `'->'`.
- Submission Area
  This area stores formal text and commands which the user wants to send to a *Theorema* server. A formula can be deleted by pressing the corresponding button. By pressing the submit button in the bottom of this area, contents of this area are sent to a *Theorema* server which runs at a remote site.

2.2.2 Flow of the Processing

The CGI (Common Gateway Interface) scripts of the web server at the remote site accept the requests (*Theorema* knowledge bases, *Theorema* proof, solve, or compute goals) from web clients and save them. A CGI script accepts the knowledge base and invokes a *Theorema* server which produces the proof (or solves the goal or evaluates the goal). The *Theorema* server returns the result to the user by the facility of *Mathematica* to convert notebook format to HTML format. The web page generated in this way is then sent back to the user.

2.2.3 Example

Theory["limit",

$$\underset{f,x}{\forall}\ (-f)[x] := -f[x] \qquad \text{"f-"}$$

$$\underset{f,a,\epsilon,N}{\forall}\ \mathrm{limit}[f,a,\epsilon,N] \Leftrightarrow \underset{n}{\forall}\ (n \geq N \Rightarrow |f[n]-a| < \epsilon) \qquad \text{"l4:"}$$

$$\underset{a,b,\epsilon}{\forall}\ |(-b)-(-a)| < \epsilon \Leftarrow |b-a| < \epsilon \qquad \text{"dist"}$$

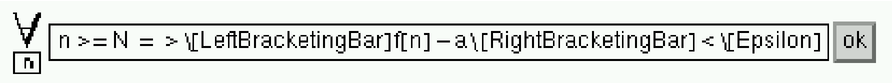
]

Fig. 2. Example of *Theorema* Formal Text

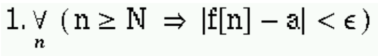
The *Theorema* formal text in Fig. 2 can be composed in the following way. First, the construction of the *Theorema* expressions labeled by `"f-"`, `"l4:"` and `"dist"` is needed. As an example, we describe the construction of the expression labeled with `"l4:"`.

This expression is composed from simpler expressions, namely '$\mathtt{f,a,}\epsilon\mathtt{,N}$' and '$\mathtt{limit}[\mathtt{f},\mathtt{a},\epsilon,\mathtt{N}] \Leftrightarrow \underset{\mathtt{n}}{\forall}(\mathtt{n} \geq \mathtt{N} \Rightarrow |\mathtt{f}[\mathtt{n}]-\mathtt{a}| < \epsilon)$'. The subexpression '$\mathtt{f,a,}\epsilon\mathtt{,N}$' is not needed to be composed beforehand because it can be simply input into the input box as '`f,a,\[Epsilon],N`'.

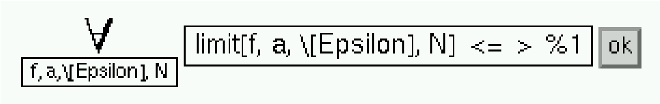
Then the subexpression ' $\mathtt{limit}[\mathtt{f},\mathtt{a},\epsilon,\mathtt{N}] \Leftrightarrow \underset{\mathtt{n}}{\forall}(\mathtt{n} \geq \mathtt{N} \Rightarrow |\mathtt{f}[\mathtt{n}] - \mathtt{a}| < \epsilon)$' has to be composed beforehand. To input this subexpression, the subsubexpression ' $\underset{\mathtt{n}}{\forall}(\mathtt{n} \geq \mathtt{N} \Rightarrow |\mathtt{f}[\mathtt{n}] - \mathtt{a}| < \epsilon)$' has to be composed beforehand. It can be composed by using the input form in the template area like this:



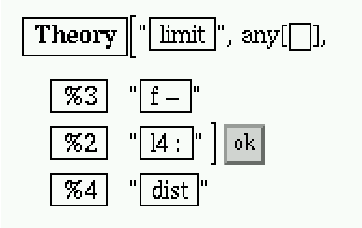
By pressing the `'ok'` button, the expression is sent to and shown in the work area. Suppose for this expression the number 1 is dispatched. Then the appearance is the following:



Then the *Theorema* expression labeled by `"l4:"` can be composed like this:



Suppose for this expression the number 2 is dispatched. And for the expressions `"f-"` and `"dist"`, numbers 3 and 4 are dispatched. Then the expected *Theorema* formal text can be composed as follows in the template area:



Then the *Theorema* formal text will be shown in the work area with a dispatched number.

3 User - Prover Interaction Within Theorema

So far, the *Theorema* group has put considerable effort into fully-automated proving. However, interaction with the user to increase the proving power and for the use of the system in mathematical education was an essential goal of *Theorema* from the outset.

Currently we develop an interface for interactive proving which makes use of existing fully-automated provers. The user shall be able to switch between automated proving only, semi-automated proving where the user is asked for input at critical or predefined points, and fine-grain interactive proving, in which the user can control every single inference step.

The system is still being developed. However, at the conference, we will be able to present fully-interactive proving where the user can determine each step.

3.1 Motivation

There are two major reasons why we wanted to introduce interaction to proving in *Theorema* : in order to *increase the proving power* through co-operation between the creative human and the ever correct and indefatigable machine, and in order to *train students* in an environment which cannot be 'cheated' nor overlooks any fault, yet always is helpful and never unnerved no matter how slow progress may seem.

Our implementation has so far concentrated on the educational aspect, as the Polytechnic for Software Engineering in Hagenberg, Austria plan to use this in their maths courses starting this autumn. Also an employment at the University of Linz is planned. However, features especially useful for the mathematical researcher or for programme verification will follow soon.

Interactive proving is not something new (for an overview, see [5]). However, our goal was to integrate such a facility into an existing, powerful environment for automated proving as it is represented by the *Theorema* system. By this, we hope to contribute to the efforts of Bruno Buchberger and the *Theorema* team to combine computer power for routine parts and human power for the creative parts of proving.

3.2 Kinds of Interaction

There are three possible modes of interaction when fully-automated provers are employed: no interaction (full automation), semi-automated proving, and stepwise (fully interactive) proving.

Semi-automated proving is designed to let an automated prover stop at 'critical points' where the proof situation calls for a creative step. Such a step can be, for instance, finding a suitable term to substitute an existentially quantified variable of the proof goal or a universally quantified variable of an assumption.

So far we have been implementing fully interactive proving where the user can determine every step of the proof, with the system checking the applicability and executing the step. We will describe the most important features in the next paragraphs.

The basic idea is that for every step, the user starts with a window which depicts the proof situation, ie the current proof goal (which can be a 'subgoal' of the initial goal) and the knowledge base consisting of the currently available assumptions. When going through long proofs by hand, such an overview of the current situation is often hard to maintain.

A tricky problem for automated provers is to select an appropriate assumption from the knowledge base for the next proof step when transformations of the goal alone are insufficient. In our interactive system, the user can select any formula from the proof situation window to make it play the key role in the next step.

When a formula has been selected, the user is presented with a list of the available inference rules to choose from. Alternatively, the selected formula can be deleted from the knowledge base.

Many inference rules require additional input. An important creative step in the course of a proof is to find a suitable term as an example that satisfies an existentially quantified formula which is to be proved or when using a universally quantified formula of the assumptions. When the respective rule is chosen, the user is prompted to give such a term in the syntax of a *Theorema* expression.

Other rules require a second formula from the knowledge base as additional input. For instance, if you select 'Modus Ponens' for an assumption that has the form of an implication, you will be prompted to select another formula that matches the right hand side.

If the selected rule is not applicable to the selected formula with the additional information given, the system gives an error message without aborting the proof. The proof situation remains unchanged, waiting for new input.

Our implementation has so far concentrated on the applicability of the system for teaching. Therefore the applicability of rules is only checked after selection. For the professional user, it would be convenient if only those rules would be listed which are applicable to the selected formula. We will implement this feature later.

So far we have implemented the following additional options (available in the proof situation window):

- Perform next step automatically,
- Finish automatically,
- Add text to the proof,
- Abort the proof, and
- Add a new subgoal.

Adding a new subgoal means this subgoal will be the proof goal in the new proof situation. If the subgoal is proved, it will be added to the knowledge base, and the proof continues with the original goal. If the subgoal cannot be proved, the proof continues with the original proof situation.

3.3 The User Interface

The main features of the user interface are the proof window and the Interactive Proof Situation window. In the proof window, the proof develops step by step with a natural-style text. By default, a new window (*Mathematica* notebook) is created for this purpose, but with the option 'in-notebook→Current', only a link for displaying the proof output is written into the notebook from where the Proof function was called.

The Interactive Proof Situation window depicts the current goal and the assumptions of the knowledge base as well as some options. The label of each formula is designed as a button to be clicked for selection - see the example given below.

The available inference rules are listed in a new window as soon as a formula is selected. The rules are identified by short names and also designed as buttons. For additional user input, again, a new window will open.

3.4 Example

In the following proof example we want to give an impression of how an interactive proof develops, and what the user interface looks like. The first lines are needed to start *Theorema* and load the required environment:

```
Needs["Theorema‘"]
StartTheorema[Prover->None,Evaluator->None]
Needs["Theorema‘Provers‘UserProvers‘PredicateProver‘"]
```

Now we consider the proof of

Proposition["sum of zero sequences", any[f, g],
$((f \longrightarrow 0) \wedge (g \longrightarrow 0)) \Rightarrow ((f \oplus g) \longrightarrow 0)$ "sum of zero sequences"]

under the assumptions

Definition["zero sequence", any[f],
$(f \longrightarrow 0) \Leftrightarrow \left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \; \underset{N}{\exists} \; \underset{n>N}{\forall} (|f[n]| < \epsilon) \right)$ "zero sequence"]

Definition["sum of sequences", any[f, g, n],
$(f \oplus g)[n] = f[n] + g[n]$ "sum of sequences"]

(For reasons of simplicity, we omit the basic properties of >', '<', and '+' over the natural and the real numbers.)

We start the proof by the call

```
Prove[Proposition["sum of zero sequences"],
  using->{Definition["zero sequence"],Definition["sum of sequences"]},
  by->PredicateProver,ProverOptions->{Interaction->"stepwise"}]
```

Starting the proof opens a new window - the proof window - displaying the initial proof situation, ie it lists the goal and the assumptions. Here, every proof step will be recorded in a natural-style language, see Fig. 3.

Another new window is the Interactive Proof Situation notebook, see Fig. 4.

Proving step by step, we first want to apply the definition of "zero sequence" to the proposition. In the proof situation window, we click the proposition "sum of zero sequences" with the mouse. A dialog allows us to select a rule from those offered by the prover. We choose

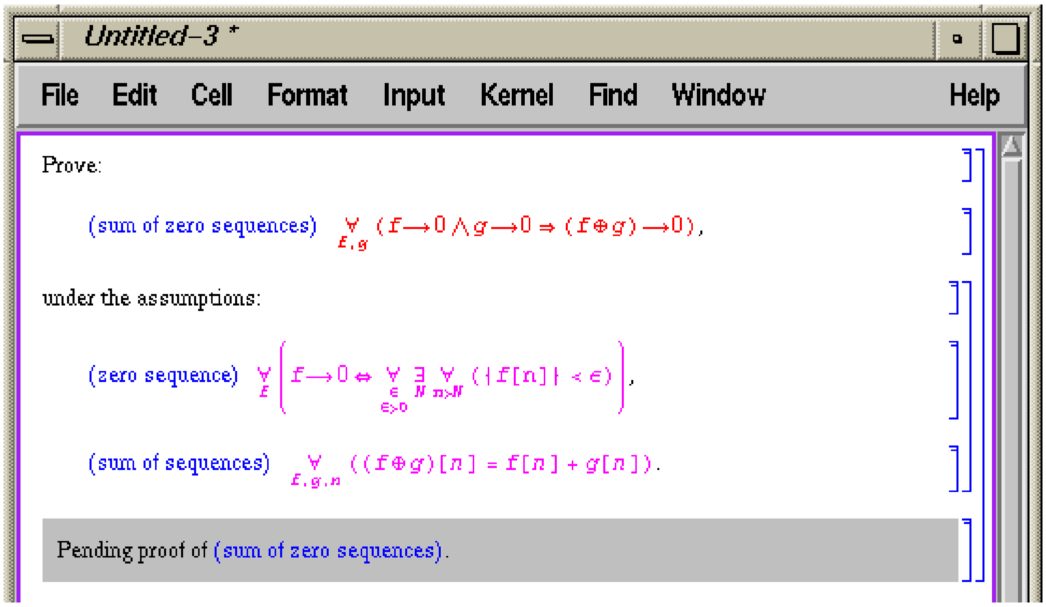


Fig. 3. Proof Window

Replace using equivalence .

The programme then checks whether further input is needed. In the case of replacement, we are prompted to select a formula to be used for the replacement; in this case, we choose definition 'zero sequence'. The proof situation window shows the resulting formula as the new sub-proof goal:

$$(1)\quad \forall_{f,g} \left(\left(\left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|f[n]| < \epsilon) \right) \bigwedge \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|g[n]| < \epsilon) \right) \right) \Rightarrow \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|(f \oplus g)[n]| < \epsilon) \right) \right)$$

Applying replacement using the 'sum of sequences' works likewise, only that now we have to choose the rule 'Replace using equality':

$$(2)\quad \forall_{f,g} \left(\left(\left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|f[n]| < \epsilon) \right) \bigwedge \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|g[n]| < \epsilon) \right) \right) \Rightarrow \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|(f[n] + g[n]| < \epsilon)) \right) \right)$$

The outermost symbol is 'for all'. In order to proof formula (2), we want to introduce new constants f' and g' arbitrary but fixed. We click formula (2) in the proof situation window and select 'Arbitrary but fixed'. The system chooses new but similar symbols for the new constants. The new proof goal is:

$$(3)\quad \left(\left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|f_0[n]| < \epsilon) \right) \bigwedge \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|g_0[n]| < \epsilon) \right) \right) \Rightarrow \left(\forall_{\substack{\epsilon \\ \epsilon>0}} \exists_{N} \forall_{n>N} (|(f_0[n] + g_0[n]| < \epsilon)) \right)$$

These three steps could have been done in a different order. Actually, *Theorema* automated provers usually start with the logically outermost symbol, so the last step eliminating the universal quantifier would have been the first. In interactive proving, we have more freedom.

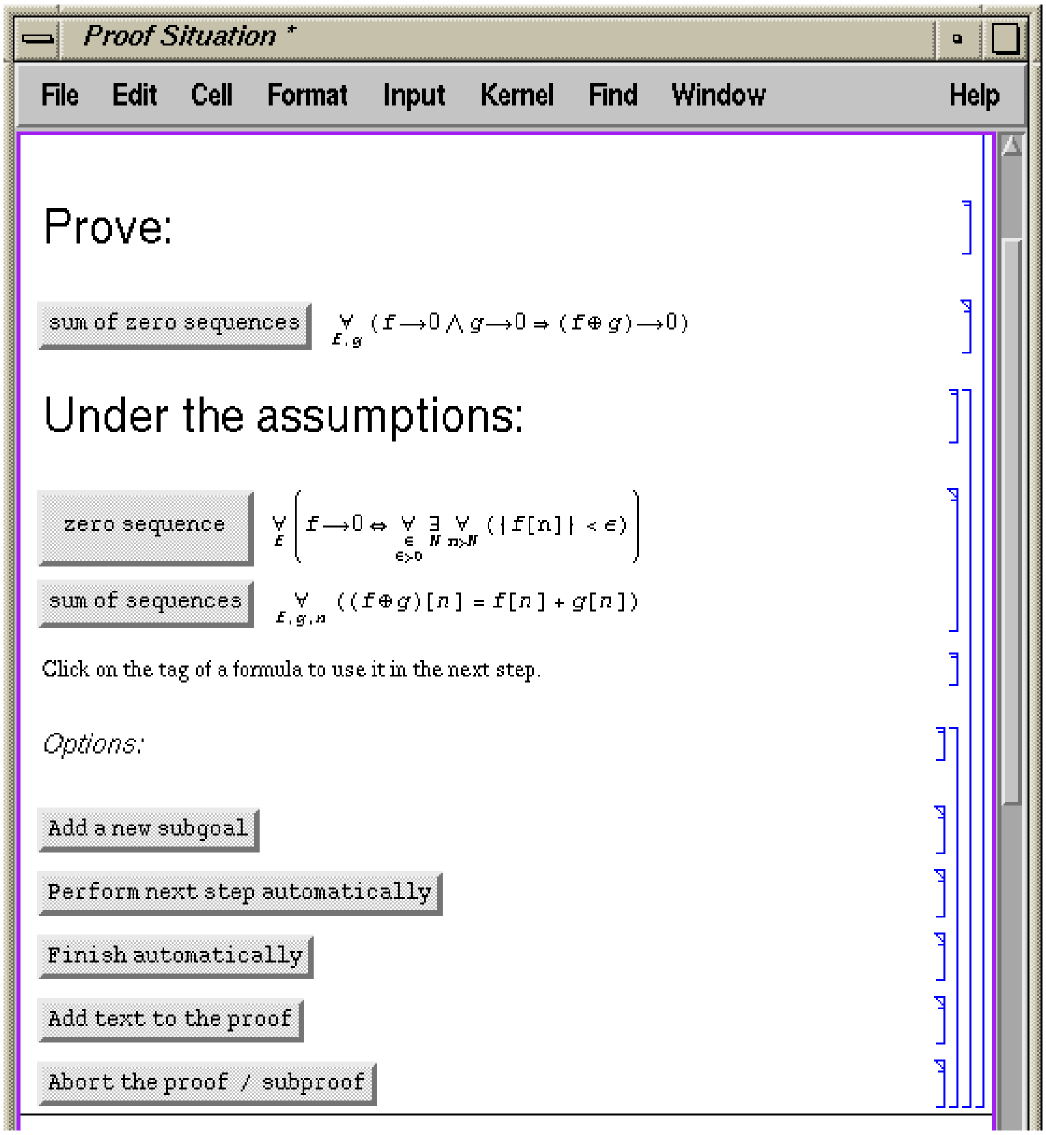


Fig. 4. Interactive Proof Situation Window

The next step will be to assume the left-hand side of the implication and take the right-hand side as the new subgoal. We click formula (3) in the proof situation window. From the available rules, we select 'Deduction rule'. The new goal is shown in the proof situation window:

(5) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|f_0[n] + g_0[n]| < \epsilon)$

as is the new assumption:

(4) $\left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|f_0[n]| < \epsilon)\right) \bigwedge \left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|g_0[n]| < \epsilon)\right)$

Eliminating the outermost quantifier of (5) works as shown above. We show the complete proof situation after this step:

Prove:

(6) $(\epsilon_0 \ngtr 0) \bigvee \underset{N}{\exists} \underset{n>N}{\forall} (|f_0[n] + g_0[n]| < \epsilon_0)$

Under the assumptions:

(zero sequence) $(f \longrightarrow 0) \Leftrightarrow \left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|f[n]| < \epsilon) \right)$

(sum of sequences) $(f \oplus g)[n] = f[n] + g[n]$

(4) $\left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|f_0[n]| < \epsilon) \right) \bigwedge \left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|g_0[n]| < \epsilon) \right)$

The next steps involve creativity, where automated provers will usually fail. If we were in semi-automated mode, the prover should stop now and ask for instructions. We try to obtain an N' which satisfies the respective subformula of (6) by manipulating assumption (4). The outermost symbol is 'and'. We click (4) and select the rule 'Use conjunction'. The new proof situation contains two new assumptions:

(7) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|f_0[n]| < \epsilon)$

(8) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \underset{N}{\exists} \underset{n>N}{\forall} (|g_0[n]| < \epsilon)$

Now the really creative step is to choose a suitable ϵ_1 and ϵ_2 for (7) and (8), respectively. We click (7) and select 'Specialise universally quantified formula'. We are prompted to choose a term to replace ϵ. We insert $\frac{\epsilon_0}{2}$. We do exactly the same with (8). Here are the two new formulae in the assumptions of the new proof situation:

(8) $\left(\frac{\epsilon_0}{2} \ngtr 0\right) \bigvee \underset{N}{\exists} \underset{n>N}{\forall} \left(\left|f_0[n]\right| < \frac{\epsilon_0}{2}\right)$

(9) $\left(\frac{\epsilon_0}{2} \ngtr 0\right) \bigvee \underset{N}{\exists} \underset{n>N}{\forall} \left(\left|g_0[n]\right| < \frac{\epsilon_0}{2}\right)$

We leave this proof here, having shown the most important possibilities for interaction. We will give a live presentation of interactive proving at the workshop.

4 Future Work

4.1 Theorema Distributed Library

Theorema formal text can be saved in a distributed way over the internet and will be open to users. Let us call this distributed library of formal text the "*Theorema* Distributed Library". The library is not only accessible from the web interface, but also accessible from the *Theorema* implementation in *Mathematica*. Important and frequently used knowledge bases will be available stored through this library so that users need not compose them from scratch over and over again. Such a library can be used for education and collaboration by researchers.

Actually the idea of having a library of (proved) mathematical theorems goes back to the initiators of the MIZAR system, see [6]. Recently, the ILF group implemented a *Mathematica* interface to the MIZAR library, see [7]. Data exchange between existing electronic mathematical libraries like MIZAR, ILF and the *Theorema* Distributed Library would also be a practically interesting topic to which we will devote some effort in the future. MathML and OpenMath (`http://www.openmath.org/`) will make this exchange easier.

4.2 Interactive Proving

In the next step of our development, we will implement the option 'critical-points', where the professional user can combine the power of the automated provers with her or his creativity in an optimum way. Checking the applicability of inference rules before listing them will further increase the comfort of use.

Another feature we have planned shall enable the user to go back in the proof tree, starting anew at an arbitrary node of the proof tree if the previous attempt has failed. We are also thinking of an online help functionality, in particular for the inference rules.

Interactive proving with *Theorema* will first be tested in the next term at the Polytechnical University for Software Engineering in Hagenberg. In the first two semesters, students get a proof training in their mathematics courses which will be supported by our system. Martin Kirchner implemented the necessary modules to enable programme verification with *Theorema* [8]. The students will then have to proof the generated verification conditions interactively.

References

[1] Bruno Buchberger: Symbolic Computation: Computer Algebra and Logic. In: Frontiers of Combining Systems, F. Baader and K.U. Schulz (eds.), Kluwer 1996, pp. 193–220. (Also available as RISC Technical Report 96–36).

[2] Bruno Buchberger: Using Mathematica for Doing Simple Mathematical Proofs. In: 4th International Mathematica Users' Conference (Tokyo, November 1996), Wolfram Media Publishing, 1996.

[3] Bruno Buchberger, Tudor Jebelean, Franz Kriftner, Mircea Marin, Elena Tomuta, Daniela Vasaru: A survey on the Theorema project. In: Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation),

Maui, Hawaii, July 21–23, 1997, W. Kuechlin (ed.), ACM Press 1997, pp. 384–391. (Also available as RISC technical report 97-15).

[4] Bruno Buchberger: Automated Theorem Proving versus Computer-Supported Theory Generation in Theorema. Invited talk at the Calculemus workshop, July 11–12, 1999, Trento Italy.

[5] Tobias Nipkow and Wolfgang Reif: Introduction to Interactive Theorem Proving. In: Automated Deduction - A Basis for Applications, Vol. II, Part 1 (Eds. Wolfgang Bibel and Peter H. Schmitt). Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[6] Piotr Rudnicki: An Overview of the Mizar Project. In: Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, 1992.

[7] Ingo Dahn, Andres Haida, Thomas Honigmann, and Christoph Wernhard: Using Mathematica and Automated Theorem Provers to Access a Mathematical Library. In: Proceedings of the 15th International Conference on Automated Deduction, July 5–10, 1998, Lindau, Germany, Integration of Deductive Systems, pp. 36–43.
`http://www-irm.mathematik.hu-berlin.de/~ilf/mca.html`

[8] Martin Kirchner: Program Verification with the Mathematical Software System Theorema. Master thesis, Polytechnical University of Upper Austria, Department for Software Engineering, Hagenberg, Austria, 1999.