JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Interactive Tools for Inspecting Proofs in *Theorema*

MASTER'S THESIS

for obtaining the academic title

Master of Science

in

INTERNATIONALER UNIVERSITÄTSLEHRGANG
INFORMATICS: ENGINEERING & MANAGEMENT

composed at ISI-Hagenberg

Handed in by:
Zsuzsanna Sinka, 0856332

Finished on:
July, 2009

Scientific Advisor:
Univ.-Prof. Dr.phil.DDr.h.c. Bruno Buchberger

Hagenberg, July, 2009

Abstract

This thesis is the result of the author' s work with the Theorema theorem proving system. We want to aid mathematics and computer science students in studying logic.

Trough this work we investigate the opportunities offered by the current version of the Theorema system and extended it with resources for studying logical proofs. The extension presented here is designed to check the user' s understanding of the inference rules of the prover during the proof. On the other hand, it can be also used to check the correctness of an automatically generated proof.

The application of our work is illustrated by examples of increasing difficulty.

Keywords: automated theorem proving, Theorema, interactive proving, focus windows, rule inspector.

Acknowledgement

I want to give first thank my adviser, Bruno Buchberger. I am very happy, proud and grateful that I had the chance to work under the supervision of such a great mind, such an inspiring person. I have learned so much from his wisdom, the examples and experiences he shared with us, ISI students during those seemingly endless seminars, through the whole year. I am glad that he considered me worthy of that, believed in me and accepted me as a Master' s student to the International School for Informatics Hagenberg, integrated me to the Theorema project, and gave me the chance to be a member of the Theorema research group. And he also forced me to continue my work in a PhD, here, in this beautiful environment, he created. Thank you very much, professor!

I thank the members of the Theorema group to accepted me, supported me during my work with their constructive critics. And thanks for answering my sometimes naiv questions and pushing me to reach my goals.

Special thanks for Florina Piroi, "my jedi myster" who helped me out during my hard times with the system, shared her wisdom and knowledge with me and did not grudge her time and energy me, introduce me to the little secrets of this work. If you would not be there for me .. You know it. ;) Multumesc!

I thank my professors from Budapest, Pásztorné Varga Katalin and Horváth Sándor, giving me the strong bases of my knowledge, and encouraging, supporting me to apply and work here. I also thank for Horváth Zoltán, who told me this opportunity.

This is the friends turn. I am thankful that they were there for me, brought colors and happyness to the sometimes gray, depresssing , sunless austrian days. Either personally, or online. I also thatnk for the persistent proof - reading, and correcting my english - I improved a lot thank you. Especially, Olimpiu, Ioanna, Zsuzska and Kicsi. :)

I do not want forget about BetinaCurtis, the „second mother” of all ISI student. She worked in the background day and night, tirelessly to make our life easier and better during our stay here. Thanks a lot, Betina!

Finally, but not in a last row thanks a lot for my beloved ones. That they encouraged me, that I can do it, and they did not leave me deep down in the hard days. Giving financial and spiritual support to survive the troubles. Köszönök nektek mindent!

The research was supported partially by the ISI habenberg, partially by a scholarship from the Austrian Government (Stipendien der Stipendienstiftung der Republik of Österreich, ACM-2008-01609)

Eidesstattliche Erklärung:

Ich erkläre an Eides statt, das ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Affidavit:

I herewith declare that I wrote the thesis by myself and without any help. I did not use any other than the quoted sources and means and that I marked all those passages which I took from the sources either iterally or analogously as such.)

Hagenberg, 01.07.2009

.....

Unterschrift / Signature

Table of Contents

Chapter 1

Introduction

Nowadays, logic is conquering a more and more important position.This universal language provides an opportunity to share knowledge without being able to speak a common natural language.Those who speak' logic' can formulate the whole world of mathematics, although learning logic is not so easy.

The *Theorema* system already provides tools for the inspection of proofs generated by (the *Theorema*) provers.

In this thesis, we illustrate tools for training predicate logic proving with mathematics and computer scientist students. In addition, we implement and describe a slight extension of one of the *Theorema* proof inspection tools. This extension allows the student to choose at each proof step, from a library of inference rules, the appropriate inference rule, which then is compared with what the automated prover was doing. Several case studies are presented in detail. Also, for beginning students who did not have any exposure to proving and the *Theorema* system, an easy-to-read manual is presented.

The structure of this thesis is the following. We describe how to support the learning process of logic (with the *Theorema* system). In the Background chapter one can read about some related work on the field and other points of view. In the Manual chapter there is the illustration about the *Theorema* system and the tools that it offers to support teaching. The description of the tool we designed and implemented, the RuleInspector, is described in the Implementation chapter, and one can also find some working examples illustrated with pictures in the Application chapter. By the end of the document, one can find some comments about Future Work.

Chapter 2

Related Work

A comprehensive literature survey on all existing reasoning systems that allow computer-assisted mathematical theory exploration and knowledge management, is given in [MathAgents08].

From this survey, here, we only mention those systems that provide tools for inspecting proofs (generated by automated or semi-automated reasoners) in an interactive. Some of the descriptions below are just condensed versions of the descriptions given in [MathAgents08] with the consent of the authors.

ActiveMath

ActiveMath is an innovative web based solution for e-learning, providing mathematicians with an interactive learning environment.

While based on an extension of OMDoc, ActiveMath is implemented in Java technology as a web based application. The mathematical concepts, represented by expressions, text and hyperlinks, have associated mathematical and pedagogical annotations in the form of attributes and relations (the metadata). For managing, referencing and searching mathematical information, it mainly uses the OMDoc definitions, examples and theorems.

The interactive interface is provided by a tool named iCMap, which helps students to visualize relations between mathematical concepts. ICMap is also an interactive concept mapping tool, providing a suggestive representation of the learning process, with a graph having abstract symbols, theories and learning objects as nodes.

The system guides the student in the learning process, and adapts itself to individual knowledge preferences, personal interests and learning goals. ActiveMath integrates several functions and tools:

- computer algebra systems
- function plotter

concept map tool

- semantic search
- notes function

Students can visualize notions from the existing ActiveMath courses, define links between them, and validate them. An evaluator checks the student s solutions by verifying the mathematical input through a computer algebra system, providing helpful feedback such as coloring in green / red the correct / incorrect answers.

The homepage of ActiveMath is http://www.activemath.org/, for more papers on the system see the survey [MathAgents08], pages 13-19.

Coq

Coq is a proof development tool for the Calculus of Inductive Constructions logical framework, providing interactive proof methods, decision and semi-decision algorithms. It also allows the user to define proof methods, by using a tactic-specification language.

Coq is also capable of integrating external components as computer algebra system or theorem provers.

Pcoq is the graphical tool providing the interactive proof development interface to Coq. It offers a wide range of fonts and colors for a suggestive representation of formulas and commands. It supports a straight-forward way to edit formulae and commands and also the proof by pointing method. ProofGeneral is a component which can be integrated with Pcoq to offer a generic interface for proof systems based on Emacs and Emacs modes.

The homepage of Coq is http://coq.inria.fr/, for more papers on the system see the survey [MathAgents08], pages 31-40.

HELM

The HELM project s objective is to study and develop a technological infrastructure for the creation and maintenance of a virtual, distributed, hyper-textual library of formal mathematical knowledge . It is a tool meant to import formal libraries, process their content, transform it in MathML, and publish it on the internet for browsing and searching.

The classical HELM working libraries are libraries imported from other systems as Coq and NuPRL.

HELM offers an interactive proving environment with the help of Matita, an interactive prover, which is as an interface between the user and the mathematical knowledge library. Besides these, Matita is also an authoring environment, assist-

ing the user in his work with support for browsing or adding mathematical knowledge, or indexing and searching mathematical objects in the HELM library.

ProofGeneral, the component integrated with Coq, can also be used with HELM. It provides script editing features, with a view of the existing open goals, which need to be proved, and another one for messages.

The homepage of Helm is http://helm.cs.unibo.it/, for more papers on the system see the survey [MathAgents08], pages 63-69 .

Isabelle

Isabelle is another framework for assisted proof development. Users can express mathematical formulas in a formal language, and also prove those formulas in a logical calculus. The main application is the formalization of mathematical proofs and, in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

Isabelle offers several Graphical Tools and Interfaces. A graph browser tool is available for visualizing theory dependency graphs in two sub-windows: a directory tree window, and the graph window, where the graph itself is displayed. The user interface component is implemented in Java and can be used both as a standalone application and as an applet.

For the interactive proving functionality, an integration with Proof General is the most common approach. The X-Symbol package from Proof General provides a rich set of input methods to enter mathematical characters.

Another Isabelle graphical interface is XIsabelle, which uses the script language Tcl and the Tk Toolkit, and allows theory browsing, proof by clicking, also while giving advice on possible proof steps. These features allow the user to work with Isabelle without having to learn its low-level commands.

The homepage of Isabelle is http://isabelle.in.tum.de/, for more papers on the system see the survey [MathAgents08], pages 84-91.

Omega

Omega is an interactive theorem prover which allows the user to develop mathematical proofs in a suggestive and helpful user interface.

Omega contains more building blocks: a proof data structure (PDS), a distributed knowledge base (MBase), a user interface (LWUI), an explanation module (P.rex), and a mathematical web service that can connect Omega with external systems (MathWeb/MathServe).

The user interface is mainly provided via LWUI. LWUI offers a graphical display of information in a proof graph, a selective term browser, with hypertext facilities,

proof and proof plan presentation in natural language, and a knowledge base creation and maintenance facility. From the architectural point of view, the system is realized in an agent-based client/server architecture.

The proof explanation is carried out with an abstraction level corresponding to the assumed user knowledge, and it has the possibility to adapt itself when receiving user input. The user is also allowed to enter remarks, requests and questions.

The second graphical interface tool is P.rex, an interactive, user-adaptive proof explanation system and also a mathematical assistant tool, connected to Omega.

The homepage of Omega is http://www.ags.uni-sb.de/~omega/software-/omega/index.html, for more papers on the system see the survey [MathAgents08], pages 159-166.

PVS

PVS is an interactive solution for formal specification and verification. The system includes a specification language with a type checker, a code generator and a random tester. To assist the user in his proof development activity, it also includes an interactive theorem prover and a symbolic model checker. Several other utilities are offered, such as formalized libraries and predefined theories.

The PVS system is successfully used in areas like requirement checking, hardware verification, testing of practical fault-tolerant systems, compiler correctness.

The PVS proof assistant, meant to support the efficient development of readable proofs, provides the user with a set of powerful inference rules, a mechanism to compose them into proof strategies, and a mechanism to rerun proofs. Examples of supported inference rules include propositional and quantifier steps, beta-reduction, equality replacement, use of lemmata, etc.

For the GUI part, PVS uses Gnu or XEmacs to provide an integrated interface to its specification language and prover. The Emacs interface provides various views for the proof interaction, theories, help, etc. The user can also visualize the proof in a graphical proof tree view, implemented in Tcl/Tk. There s also the possibility to view theory hierarchies graphically, again, using Tcl/Tk.

The homepage of PVS is http://pvs.csl.sri.com/index.shtml, for more papers on the system see the survey [MathAgents08], pages 174-182.

Chapter 3

Manual for Interaction in *Theorema*

3.1 The Main Example Throughout the Paper

For explaining how to work with *Theorema*, and in particular with the tools for interaction available in *Theorema* and developed in this master's thesis, we take the following formula a the running example:

Proposition["ForAll Distributivity",

$$\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right) \quad \text{"}\forall d\text{"}\ ]$$

Using this example, we will explain

- how to type in formulae (knowledge bases and proof goals) in *Theorema*,
- how to call *Theorema* provers, and
- how to interact with *Theorema* provers.

Before doing this, we give a very short introduction to the *Theorema* system. For more details see [BuchbergerEtAl97], [BuchbergerEtAl00], [BuchbergerEtAl06]

3.2 What is *Theorema*?

The *Theorema* project, initiated and directed by Bruno Buchberger, aims at supporting the entire mathematical exploration cycle including the proving phase. Built on top of Mathematica [9], it implements a two-dimensional syntax of mathematical formulae similar to the syntax used by mathematicians, and a formal text language for defining new notions, properties of notions, problems, and algorithms, and for combining mathematical formulae in knowledge bases and for using it in proofs. The system contains various general provers (which can be used for generating proofs in arbitrary mathematical theories) and special provers (which can be used for generating proofs in specific theories), see [Windsteiger06], [BuchbergerEtAl06], [BuchbergJeb97] or [Buchberger01] for more details.

In the default setting, a *Theorema* reasoner tries to solve a given reasoning problem automatically by repeated inference rule application, until either a successful proof is obtained or no more inference rules can be applied. An inference rule takes as input a reasoning situation consisting of a reasoning goal and a knowledge base and returns a new reasoning situation.

Additional prover tuning is possible by specifying option values that guide the proof search. Users also have the possibility to prove in interactive mode, which supports a step-based reasoning [PiroiKutsia05], [Piroi04].

The result of a reasoning call - successful or not - is displayed in a user-friendly, textbook-style presentation as a separate Mathematica notebook. The *Theorema* proof presentation has the additional feature that the proof text is structured in nested cells. Each cell can be opened or closed. When it is opened one sees all the (proof) information in the cell. When it is closed, one only sees the first line. This feature can be used conveniently for structuring proofs in a main proof level, sub-proofs, sub-sub-proofs etc. allowing the user to browse through the proofs in various levels of detail. If, at a given moment of inspecting a proof, one is not interested in the details of a certain sub-proof, one just close the cell containing the sub-proof.

3.3 How to Start *Theorema*

After having started *Mathematica*, *Theorema* is started by typing and calling the following command into a *Mathematica* input cell (stand into the cell and press SHIFT ENTER):

```
Needs["Theorema`"]
```

3.4 Typing Formulae

Theorema offers two-dimensional syntax which is close to what could be considered the "usual" mathematical syntax used in mathematical papers. (Of course, a rigorous specification of this "usual" mathematical syntax does not exist. Thus, we also could say the *Theorema* syntax is an attempt to propose a nice two-dimensional mathematical syntax that should be easy to acquire by most mathematicians.) Note that, independent of whether or not a user likes the specific Theoerma syntax, formulae presented to *Theorema* in the *Theorema* syntax are automatically parsed internally in a way that is consistent with predicate logic so that, afterwards, logical inference rules can be applied to such formulae. (Also, we would like to mention that, if a user does not like the particular *Theorema* syntax implemented as the default, the *Theorema* system in conjunction with *Mathematica* provides a facility to specify and implement one's own favorite syntax.)

For helping typing *Theorema* formulae, a number of syntax patterns are available in a palette: Open Palettes -> *Theorema* Language, then open one of the sections like "Elementary Building Blocks", "Ranges" etc. and click on one of the available patterns.

Alternatively, it is possible to generate the special characters by either pressing the escape key, typing a short-cut for the desired mathematical character, and pressing the escape key again. Also, there are a couple of combinations of the Ctrl-key with number keys that allow to navigate to subscript, under-script, superscript etc. position.

As another alternative, one can generate special characters by going to the palette "Special Characters", finding the desired character there and clicking on it. In fact, if one goes over a special character in the "Special Character" palette, a small window is shown that tells one all the possible ways of typing this character by a combination of keys on the key board.

Example: Let's input the formula

$$\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right)$$

First alternative, using the *Mathematica* palette "Special Characters":

type (
click ∀ in the palette "Special Characters", sub-palette "Symbols", sub-sub-palette "×"
CTRL =
type x
CTRL SPACE
type (
type P
type [

type x
type]
click ⟹ in the palette "Special Characters", sub-palette "Symbols", sub-sub-palette "⇐⟹"
type Q
type)
type)
click ⇔ on the "Special Characters", sub-palette "Symbols", sub-sub-palette "⇐⟹"
type (
type (
click ∃ in the palette "Special Characters", sub-palette "Symbols", sub-sub-palette "×"
CTRL =
type y
CTRL SPACE
type P
type [
type y
type]
type)
click ⟹ in the palette "Special Characters", sub-palette "Symbols", sub-sub-palette "⇐⟹"
type Q
type)

Second alternative, using the special sequences of key strokes on the key board: Generate the formula by the following sequence :

(ESC f a ESC CTRL x CTRL SPACE (P [x] ESC = = > ESC Q)) ESC < = > ESC ((ESC e x ESC CTRL y CTRL SPACE P [y]) ESC = = > ESC Q)

Third alternative, using the *Theorema* palette "Theorema Language":

Type (using the tools explained above)

```
((P[x] ⟹ Q)) ⇔ ((P[y]) ⟹ Q)
```

then mark "(P[x] ⟹ Q)", go to the *Theorema* palette "TheoremaLanguage", sub-palette "Quantifiers", sub-sub-palette "Quantifiers from Formulae to Formulae" click on $\forall_{\square}$ and fill in x under the "for all" quantifier, then mark "P [y]", go to the *Theorema* palette "TheoremaLanguage", sub-palette "Quantifiers", sub-sub-palette "Quantifiers from Formulae to Formulae" click on $\exists_{\square}$ and fill in y under the "there exists" quantifier

3.5 Building up Knowledge Bases in *Theorema*

The formula

$$\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right)$$

is a "plain" predicate logic formula. In mathematical proofs etc., we often want to refer to such formulae by "labels". For this, *Theorema* provides various tools, which one can find in the palette "Formal Text Language".

Example: In order to generate the labeled formula

Proposition["ForAll Distributivity",

$$\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right) \quad \text{"}\forall\text{d"}\ ]$$

Mark the already typed in formula $\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right)$, than we go to the palette "Formal Text Language", sub-palette "Individual Environments" and click to □[□, , mark then delete the last row of the scheme.

□ □
□ □]

□["□",

$$\left(\forall_{x} (P[x] \Longrightarrow Q)\right) \Leftrightarrow \left(\left(\exists_{y} P[y]\right) \Longrightarrow Q\right) \quad \text{"□"}\ ]$$

and ...mark the first square and write the word "Proposition", mark the second square between the quotation marks and write there the name of the notion: "ForAll Distributivity", than click on the remaining square (between the quotation marks), here comes the label of the formula.

Such a labeled formula can now either be used as a proof goal or as a knowledge base from which other formulae can be proved. Of course, typically, in knowledge bases, we may have many formulae. For putting more than one formula into a knowledge base one may again use the "Formal Text Language" palette.

Example: In order to generate the formula

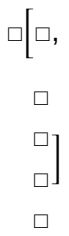
Theory["4−ary limit",

Definition["4−ary limit"]
Definition["sum of sequences"]
Proposition["distance of sum"]]
Proposition["max"]

visit the palette "Formal Text Language", sub-palette "Collection of Environments" and click the scheme □[□, .To add more lines to the scheme, point to the

□
□]

last branch of the scheme, after the comma, before the closing square bracket and either go to "Insert" menu, "Table/Matrix" then "Add Row" menu or just press SHIFT ENTER as many times as necessary. When the scheme with the 4 (or the desired number of lines) is ready,



one should only mark the squares and fill in the necessary content. (Note that in order to write a name containing the hyphen/dash, one has to type ESC - ESC , otherwise the *Mathematica* interprets it like a minus.)

3.6 Proving in *Theorema*

After one built up a knowledge base, one can try to prove a statement by using the system.

3.6.1 The Prove Command

For calling a *Theorema* prover, we have to specify

- the proof goal,
- the knowledge base,
- and the prover to be applied.

Correspondingly, the *Theorema* Prove command has the following basic structure:

```
Prove[proof goal,
  using → knowledge base,
  by → prover]
```

In fact, the *Theorema* Prove command has many more optional parameters, which one can find using the *Theorema* help mechanism, for example

```
Options[Prove]
```

```
{}
```

If one wants to know more about one of this parameters, one should type two question marks followed by the name of the option (no space between them) and press SHIFT ENTER, like

```
?? ProverOptions
```

These options can be given in a very flexible way. The only restriction on order is that the first parameter of the proof call has to be the statement

intended for proof. The others can follow in any order.

3.6.2 The User Provers

Looking at the *Theorema* help (Help menu, Documentation center, Add-Ons and Packages>>, *Theorema*), one can find all of the currently available provers with their description.

The system has "General Provers", like

- PropositionalProver,
- PredicateProver,
- PLM (Predicate Logic Prover with Meta-variables),
- PCS (Prove-Compute-Solve),
- SimplifierProver,
- EquationalProver, ...

and "Special Provers" collected to one specific field, like

- Natural Number Induction,
- Tuple Induction,
- Set Theory,
- Groebner Bases Prover, ...

3.7 Presentation of Proofs in *Theorema*

The showBy parameter of the Prove command can be used for choosing the display method of the generated proof. The possible options are the following:

3.7.1 Linear Presentation

By default, a generated proof is displayed in a user-friendly, textbook-style presentation as a separate Mathematica notebook. The *Theorema* proof presentation has the additional feature that the proof text is structured in nested cells. One can close/open this cells depending on the level of necessary details.

The proof is presented in natural language - the default language being English. (The language can be changed even during the work with the system, the files for different languages are grouped together in a special directory.)

During the proof, all of the formulae have a label from its first appearance and only this label will be shown later like mentioning, using the formulae. This labels are hyperlinks, clicking on them causes a window to pop up with the entire formula (and label also). If a label was already given to the formula the system uses that one, otherwise the label will be generated automatically. One can choose the style of the generated labels, as follows:

```
SetGlobals[LabelStyle → "Numeric"]
```

or even only for the certain proof, like

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver, label style → Numeric]
```

```
- ProofObject -
```

In the proof text, one can see different color-codes for differentiating the formulae; red for goals, magenta for assumptions, blue for labels. The cells have also different colors depending on their type and place in the proof, e.g. higher/lower level, branching, etc. (Note, that the presenting style is customizable, users can define their own set of colors, fonts, etc.)

At the bottom of the proof window, one can find the "Additional Proof Generation Information" section. The given section contains one can access information like "Proof Call", "Version Information" and "Formulae occurring during the proof".

Example:

If we input the command

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver]
```

```
- ProofObject -
```

then the following proof will be automatically generated:

Prove:

(Proposition (ForAll Distributivity): ∀d) $\underset{x}{\forall}(P[x] \Rightarrow Q) \Leftrightarrow \left(\underset{y}{\exists} P[y] \Rightarrow Q\right)$,

with no assumptions.
We prove (Proposition (ForAll Distributivity): ∀d) in both directions.
Direction from left to right:
We assume

(2) $\underset{x}{\forall}(P[x] \Rightarrow Q)$

and show

(1) $\underset{y}{\exists} P[y] \Rightarrow Q$.

Formula (2) is simplified to:

(5) $\underset{x}{\exists} P[x] \Rightarrow Q$.

We prove (1) by the deduction rule.
We assume

(6) $\underset{y}{\exists} P[y]$

and show

(7) Q.

By (6) we can take appropriate values such that:

(8) $P[y_0]$.

Because the goal (7)is identical to the conclusion of (5), we transform it into:

(9) $\nexists_x P[x]$.

Formula (9) is simplified to:

(10) $\forall_x (\neg P[x])$.

Formula (7) is proved because (8) and (10) are contradictory.

Direction from right to left:

We assume

(4) $\exists_y P[y] \Rightarrow Q$

and show

(3) $\forall_x (P[x] \Rightarrow Q)$.

For proving (3) we take all variables arbitrary but fixed and prove:

(11) $P[x_0] \Rightarrow Q$.

We prove (11) by the deduction rule.

We assume

(12) $P[x_0]$

and show

(13) Q.

Because the goal (13)is identical to the conclusion of (4), we transform it into:

(14) $\nexists_y P[y]$.

Formula (14) is simplified to:

(15) $\forall_y (\neg P[y])$.

Formula (13) is proved because (12) and (15) are contradictory.

□

3.7.2 The Focus Windows

In addition, *Theorema* offers a proof presentation tool, called "Focus Windows", introduced in [Buchberger00a] and implemented in [Piroi04], see also [BuchbergPiroi02]. The Focus Windows proof presentation method is similar to a big magnifying glass held over a particular point in a proof and showing the current reasoning situation (more precisely, exactly the formulae used as premises and as goal in the inference rule to be applied) and the resulting reasoning situations. In other words, in contrast to the usual "linear" presentation of proofs texts in mathematical papers, at a given moment in a proof, only those formulae are shown in the "focus window" that are relevant at the particular proof steps. Note that, in a linear presentation of a long proof, the relevant formulae in a given proof step may be distributed over many different pages, which may make it very difficult for the reader to check proofs. In contrast, in the focus windows proof presentation technique, the relevant information is always focused / concentrated / condensed in just one "window".

Example:

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver, transformBy → ProofSimplifier,
  TransformerOptions → {steps → Useful}, showBy → FocusWindows]
```

```
- ProofObject -
```

3.7.3 The Rule Inspector

The **RuleInspector** is an extension of the Focus Windows technique, first introduced in [SinkaPiroi09]. We designed it to help check the user's understanding of the inference rules of the prover being studied. It can be also used to check the correctness of an automatically generated proof. By calling a proof, another window appears besides the classical Focus Window, the Inference Rule Library. This is a collection of the possible inference rules of the prover used for proving. The user can click on the name of the inference rule in order to check if it was used in the current proof step or not. The RuleInspector uses the two-phase style of the Focus Window but it has additional features depending on its style (strict or free). More details are presented in the ***Implementation*** chapter.

Example:

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver, transformBy → ProofSimplifier,
  TransformerOptions → {steps → Useful}, showBy → RuleInspector]
```

(For screenshots about the proof and their explanation go to the ***Applications*** chapter and see the "Predicate Logic Example".)

Chapter 4

Implementation

4.1 The Goal of the Program RuleInspector

In a mathematical proof it is usually not written which proof rule was applied in a certain proof step. Although the ones who created the proof know exactly which one of the rules are necessary, for a new person on the field it takes time to acquire the methods, to get used to use them and to learn the small tricks. We tried to reduce this time to gain a quicker taste of the proving methodology. The **RuleInspector** was created to investigate the proofs, although it is also capable to help debugging the functioning of the provers.

By calling the **RuleInspector**, two windows will open: a proof window (**Focus Window**) with the Initial Proof Situation and another one with the collection of the proof rules for the currently used prover, the **Inference Rules Window**. The proof rules are grouped by common features like rules that operate on the goal, rules that terminate the proof, rules operating on the assumptions, rules operating on both goals and assumptions, and so on. (At this stage of the implementation this collection is available only for the PredicateProver and for the PropositionalProver, but we are working on extensions. See the ***Future Work*** chapter.)

The inspection has as many steps as the proof has. One can follow the proof steps in the Focus Window and one should choose from the list of inference rules the one used in the current step. The Focus Window presentation shows two phases for each proof step. The 'Attention Window' shows the user the formulae that have been utilized by the applied rule. The 'Transformation Window' adds to this any formulae that have been inferred by the inference rule applied.

Independent of the two focus windows phases, the rule inspector can be used in two modes: free - the default one - and strict. By default, a user can freely navigate through the proof, without necessarily choosing an inference rule in the Inference Rule Window. In the strict mode reaching a situation in a Transformation Window the user cannot continue navigating through the proof until the inference rule actually used is picked from the inference rule window. In the strict inspection mode the navigation buttons are deactivated.

4.2 The Rough Structure of RuleInspector

The **RuleInspector** is implemented like a *Theorema* package. Here is a proof call with the basic structure:

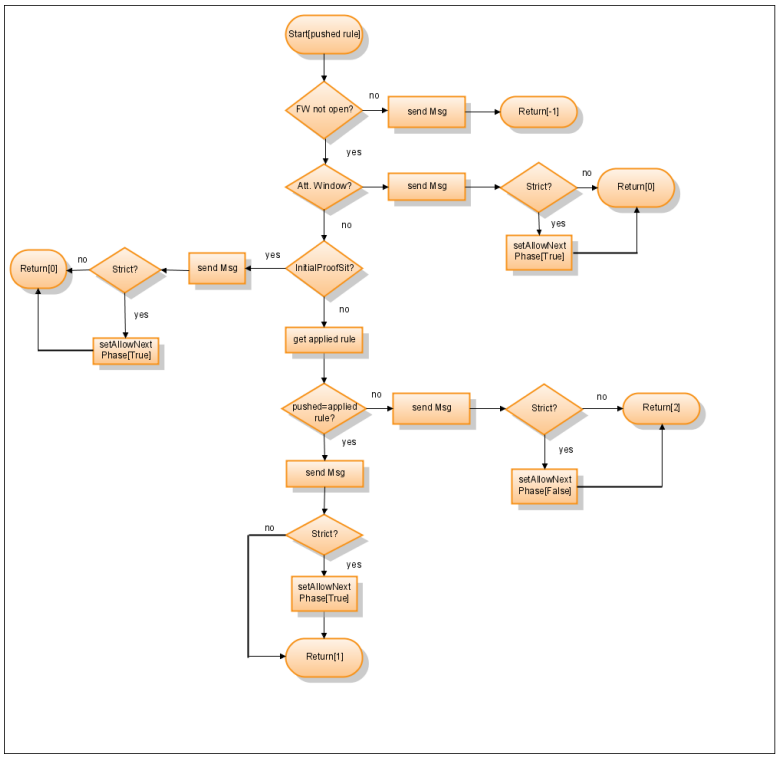
```
Prove[ ..., by → ..., showBy → RuleInspector, ...]
```

It will load the package in which the **RuleInspector** routines are implemented and use them for proof presentation when the *Theorema* proof search routines are done. The window with the collection of the inference rules of the currently called prover will be created and opened, and shown in the Focus Window presentation style. (With other words, the RuleInspector package collects the implementations of the RuleInspector presentation method.)

The input parameters of the **RuleInspector** function are the proof object and the options (free or strict). If no option is provided, then the default one is used. The RuleInspector function looks at the general proof context created by *Theorema* to see which prover was used to search for a proof.

Each proof step is an inspecting scene with a checking function in the background. The input of this function is an inference rule name and is picked (for each step) by the user from the collection of the possible inference rules. (The check function is called for each click on a rule name, while the RuleInspector function is called only once.) The output of this checking function is the answer yes/no (correct/not correct), with the restriction that if the proof was called in strict mode it sets the 'continue' flag to true, so that the user can go on with the proof inspection.

The work flow of the checking function is as follows:



where

- "FW" stands for abbreviating Focus Window
- "Strict?" stands for checking whether the strict option was used by the proof call or not (namely `ShowOptions -> {Check -> "Strict"}`),
- "send Msg" stands for the `NotificationDlg["..."]`, filled with different text depending on the kind of the focus window, and the result of the matching (if there was matching in the current step),
- "get applied rule" stands for searching for the proof rule applied in the current node of the proof tree,
- "pushed=applied rule?" stands for do the matching.

Let' s take the following proof call as an example

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver,
  transformBy → ProofSimplifier, TransformerOptions → {steps → Useful},
  showBy → RuleInspector, ShowOptions → {Check → "Strict"}]
```

The output of the *Theorema* provers are proof objects that contain all the information on the sequence of inference rules used in the proofs and the formulae involved in each inference step. The *Theorema* post processor for proof objects produces a *Mathematica* notebook from the final proof object that shows the entire proof object together with some explanatory text in natural language (e.g. English) at each proof step. However, many times, the user does not want to see the complete information on the proofs generated. Often, the user is only interested in the successful branch in a proof search and does not want to see the information on failing or superfluous branches. Also, when a proof has been found by a prover, it may well be the case that afterwards, by inspecting the proof object, one may be able to re structure the proof in such a way that it becomes more concise and easier to understand. That is why we use the `ProofSimplifier` and we are interested only in the useful steps.

During the matching, the funciton needs only to find the actual place in the proof and all of the desired information are stored there. One can see that the matching happens basically in the Transformation phase. In the Initial Proof Situation there is nothing to check, because there is no inference applied, therefore the user has to press next. In the Attention phase, one can only study the currently given proof situation, while the guess of the rule used has to be made in the Transformation phase. (In the future, we will extend the inspection for advanced users where their guess can be make in every phase.)

4.3 RuleInspector Program Code in *Mathematica* with Comments

```
Clear[fun];
fun::usage := "The function which compares
        the proof rule applied with the proof rule chosen by the user.";

fun[buttonRule_] := Module[{ndType},
 (* do some checking *)
 (* is the Focus window open while pushing the button in the inference window? *)
 If[Head[$FocusWnd] =!= NotebookObject,
 (* if it is not.. *)
 NotificationDlg["The Focus window is not available. please call the proof again."];
 Return[-1]];
 (* if the FW is open... *)
 If[MemberQ[Cases[Options[$FocusWnd],
          (WindowTitle -> x___) -> x], "Attention Window"],
 NotificationDlg["Please press Next."];
 (* if the current FW is an
        "Attention Window" one should go furter to the next phase *)
 If[$TmaSChecker === "Strict", SetAllowNextPhase[True]];
 Return[0]];

 (* get the node type - the applied rule from the actual node of the proof tree*)
 ndType = GetProofStepParamsAtPos[
        $FWProofObject, GetFocusPosition[][[1]], "InternalPosition"];
 (* some debug information *)
 PrintDebug[$DbgRuleInspector,
        "RuleInspector package:: inside fun: ", ndType, ndType[[1]]];

 If[ndType[[1]] === "InitialProofSituation",
 (* if we are in the initial proof situation, go further *)
 NotificationDlg["Please press Next."];
       If[$TmaSChecker === "Strict", SetAllowNextPhase[True]];
 Return[0]];
 (* here comes the matching... we are in a "transformation window" *)
 Function[{guess},
  If[guess === ndType[[1]],
  (* if they are the same... *)
  NotificationDlg["Congrat! You did it. It was the right rule. Keep it up!"];
  If[$TmaSChecker === "Strict", SetAllowNextPhase[True]];
   Return[1],
  (* if they do not match... *)
  NotificationDlg["Wrong answer. Try to find the rule applied by the prover."];
  (* in the strict mode one can not go further
            until the button of the applied rule was pushed by the user *)
  If[$TmaSChecker === "Strict", SetAllowNextPhase[False]];
  Return[2]]][buttonRule];
];
```

Chapter 5

Applications

In this chapter, we are going to illustrate the work of **RuleInspector**. To do this, three examples will be given, each one of them will be supported with screenshots and their explanation.

The first example, "*case analysis*" is from propositional logic. For inspecting this proof, we used the PropositionalProver without any restriction regarding the guessing. The second one is the main example of the thesis, the "*ForAll Distributivity*". We inspected the proof generated by the PredicateProver in a "strict" way. Finally, the last example is from elementary analysis, for which (domain) we have built a complex knowledge base. The example is aimed to emphasize the difference in the flow of the generated proof, as a consequence of adjusting the (special) settings for the Prove command.

5.1 Description of the windows

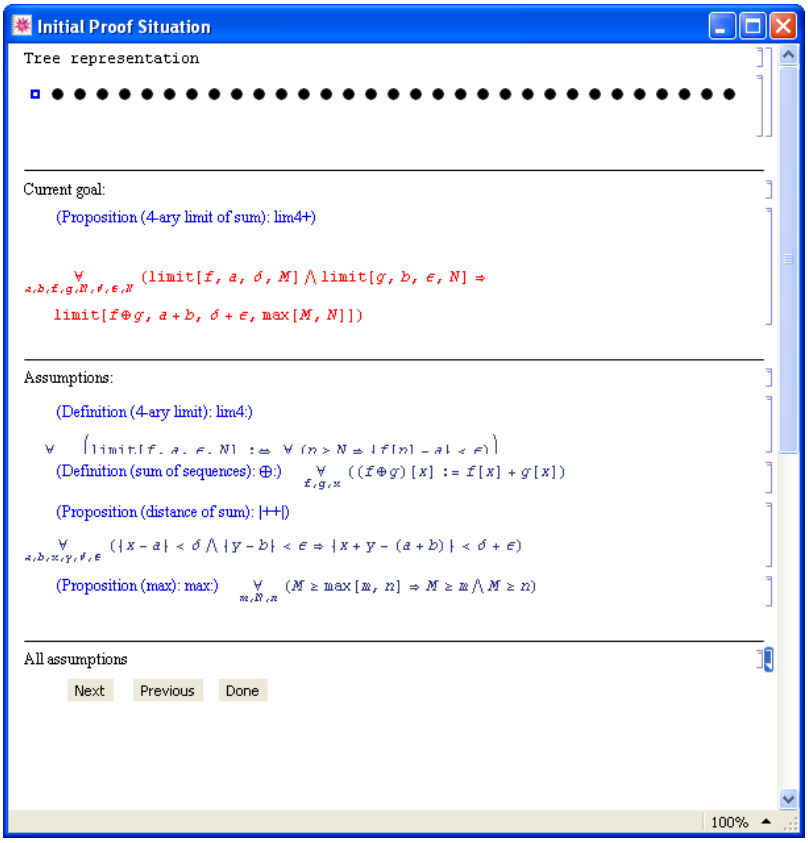
During an inspection, one will meet three groups of windows belonging to the **RuleInspector**.

5.1.1 The Focus Windows

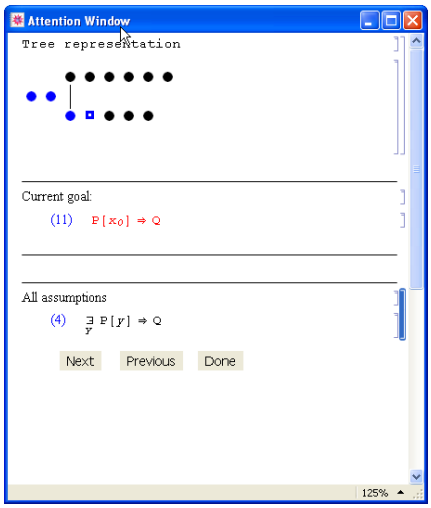
The focus windows have all the same basic structure, on top of the window there is the graphic representation of the proof, the "Tree representation". This cell is closed by default. One can follow the state of the proof in this tree. The black bullets are for the proof steps that are not discovered yet, the blue ones are for the already visited ones and the blue square shows the currently active step. Navigating between the steps is possible by clicking on the bullets (if one is allowed to go further in the proof).

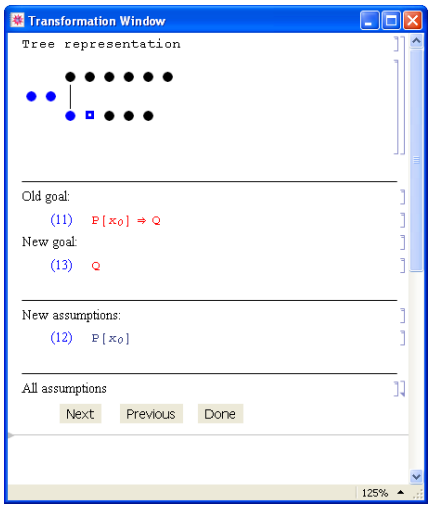
The relevant formuale for the current step are collected in the next field, between the horizontal lines. And under them one can find the (by default closed cell) of "All assumptions". By the bottom of the window there are three buttons - 'Next', 'Previous' and 'Done' - dedicated to navigate between the proof phases. (It is important when leaving the Focus Window that you do not close it with the cross

button in the upper right corner. Rather, push 'done' in order to finish/leave the proof.)



The InitialProofSituation - the first window, it appears only at the beginning of a proof.





The Attention Window - it shows the pre-inference situation.

The Transformation Window - it shows the post-inference situation.

5.1.2 The Window of the Inference Rules

The content of this window depends on the global prover. All of the possible inferences are there, grouped in a nicely open/closed way.

The PropositionalProver has six different groups of inference rules, namely:

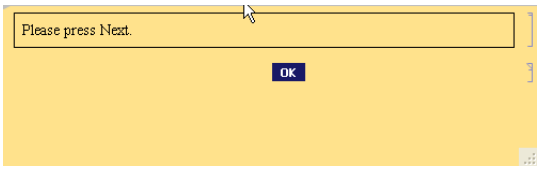
- Terminal Cases (Trivial, Handling Truth),
- Simplification,
- Resolution, Modus Ponens and Modus Tollens,
- Simplifying/Reducing the Goal,
- BackChaining Rules,
- Case Distinction

The PredicateProver has seven different groups of inference rules, as

- Trivial Cases,
- Simplification,
- Resolution,
- Expansion of Quantified Assumptions,
- Reduction of Proof Situations,
- Quantified Conclusions and Back Chaining,
- Case Distinction.

5.1.3 The Message Windows

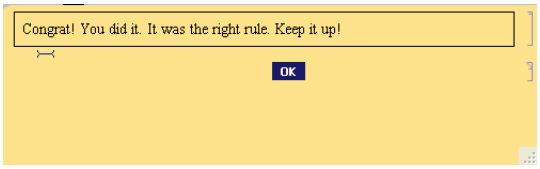
Navigating between the proof steps one can get three basic types of messages:



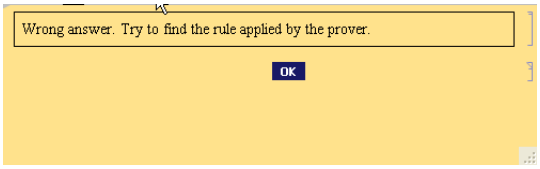
This message appears by clicking on a button in the Inference Rule Library while the focus window is

- the Initial Proof Situation (because there is no inference rule to be applied),
- an Attention Window (in the current state of implementation one can guess only in the transformation phase)

In the transformation phase (after the inference rule was applied; the Focus becomes a Transformation Window) depending on the correctness of the picked rule the two following answer is possible:



The previous one in case of the right answer and if the answer is not correct the next one.



In the free mode they are all the possible message windows.

In the "Strict" mode there is one more message:

This one is a warning that one cannot go further in the proof until the good rule was picked from the Inference Rule Library.

5.2 A Propositional Logic Example

The propositional logic is the basis of the predicate logic. Let' s start the demo with an example from this basic domain. We choose case analysis as example.

Before starting the proof, check whether the *Theorema* is running, then set the global prover for the propositional one.

```
SetGlobals[Prover → PropositionalProver]
```

Input the example:

```
Proposition["case analysis",
  (((A ⋁ B) ⟹ C) ⟺ ((A ⟹ C) ⋀ (B ⟹ C)))]
```

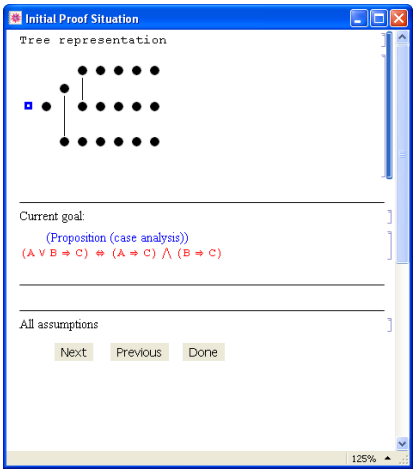
If the global prover is set and one would like to use this one for proving a statement, one can leave out the "by → ..." parameter from the proof call. In this example we are going to use the default settings of the ***RuleInspector***, the inspection is going in a free way.

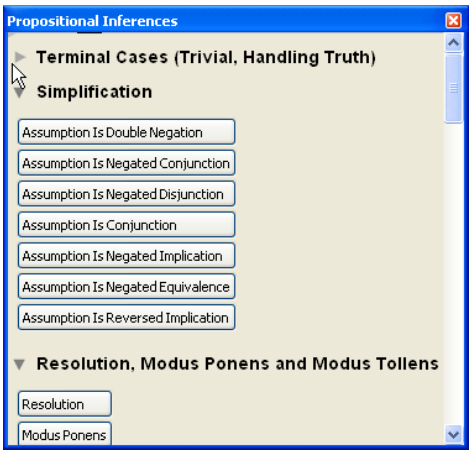
Input the following proof call :

```
Prove[Proposition["case analysis"], transformBy –> ProofSimplifier,
  TransformerOptions –> {steps –> Useful}, showBy –> RuleInspector]
```

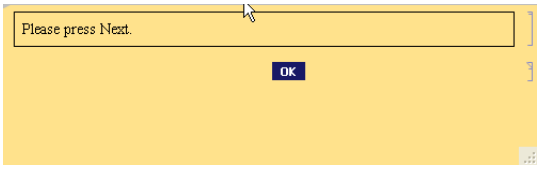
```
- ProofObject -
```

The following two windows will appear besides the already opened windows:

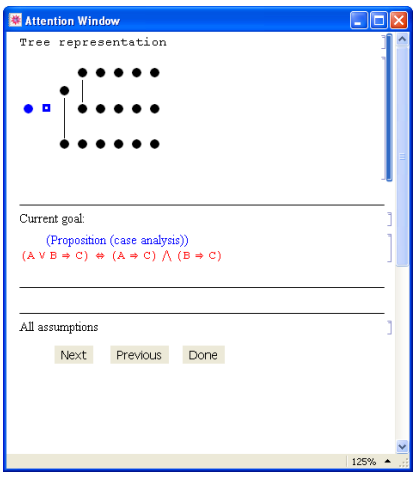


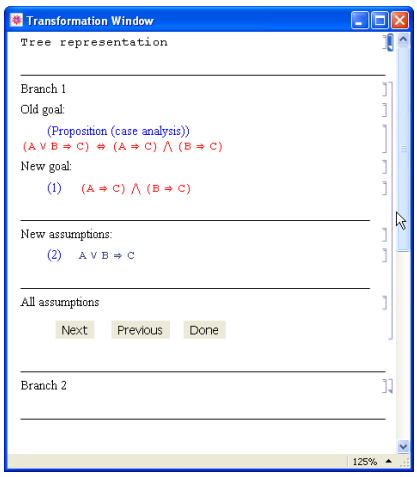


The Initial Proof Situation and the window of the Propositional Inferences. One can choose where to click. Any click on a button in the 'Propositional Inferences' window causes the following message:



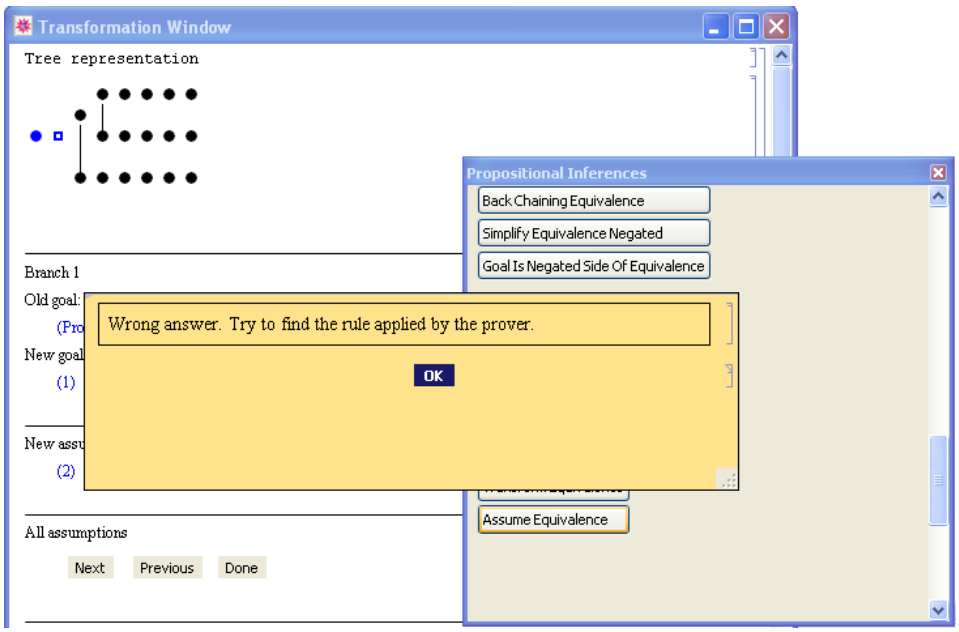
(Note, that in the Initial Proof situation there is no inference rule to apply.) For other options for clicking, there are the buttons in the 'Initial Proof Situation' window. The suggestion is to press this 'Next'. Pressing 'Done' ends the proof inspection, pressing 'Previous' -in this special case - causes the same result, like pressing 'Next', the Initial Proof situation is transformed to 'Attention Window', displaying the first inference step before the (inference) rule was applied. Regarding that the inspection can be done only in the 'Transformation Window' phase, clicking on any of the buttons in the Propositional Inferences Window causes the previously shown message, "Please, press Next".





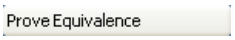
After pressing 'Next', the 'Attention Window' turns into 'Transformation Window'. This is the time to make a guess, which one of the inference rules was used in the

current proof step. In this particular step, we have two branches after the inference rule was applied on the formula shown in the 'Attention Window'. One can open each of the branches in order to see the generated formulae. Choose a rule button form the, e.g. 'Assume Equivalence' from the Case Distinction group. The result is the following:

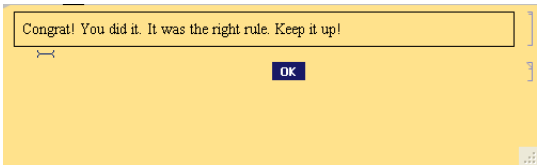


The navigation between the rule groups is easy, use the scroll bar at the right hand side of the inferences window. One can easily open or close a group of rule buttons by clicking on the triangle before the name of the group.

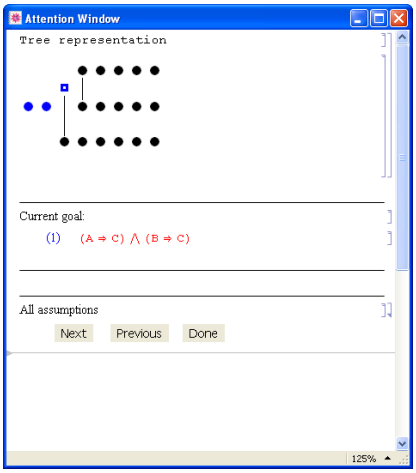
The right answer is

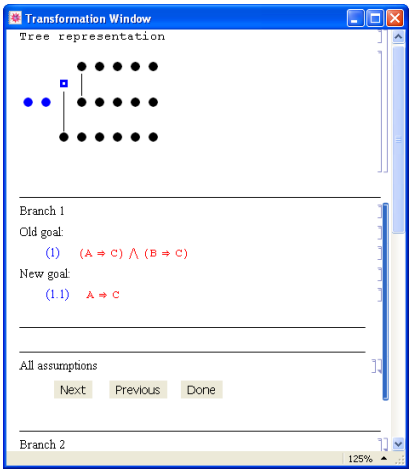


One can find this rule in the 'Simplifying/Reducing the Goal' group. After clicking on this one, we get the following message from the system:

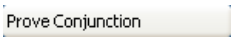


Now we can continue the proof. There are two branches, one has to decide, which one will be inspected next time. We have chosen the first one in this demo. After pressing 'Next' in the first branch, the following 'Attention Window' will be shown:



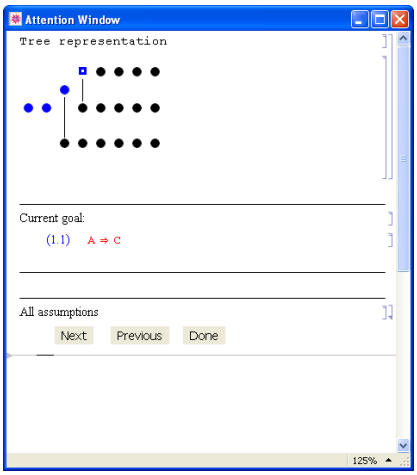


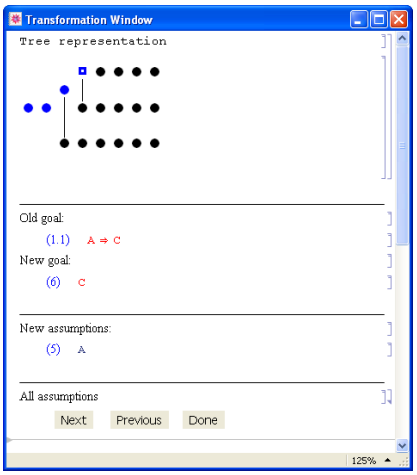
Clicking on the Next button again, we will get the 'Transformation Window' above. We are in free inspecting mode, one does not need to pick a rule in each transformation step in order to continue the proof. (The right one in this situation is



This one is also in the 'Simplifying/Reducing the Goal' group.) Note that we can really go further without pressing a button. Try it out!

We have chosen the first branch again by going further in the proof.





Now, we assume the user gets the idea and can move on further. (In the ***Appendix*** chapter, in ***A.1*** we show each proof step with the Attention and Transformation Windows, and below them the rule applied in the particular step.)

5.3 A Predicate Logic Example

After we inspected a propositional logic example, we are going to take one from predicate logic. It is based on propositional logic but there are in addition the quantifiers.

This particular example is the main example of the thesis. Before proving it, set

the global prover to the PredicateProver, in order to let the RuleInspector access this.

```
SetGlobals[Prover → PredicateProver]
```

```
Proposition["ForAll Distributivity",
    (∀x (P[x] ⟹ Q)) ⇔ ((∃y P[y]) ⟹ Q)    "∀d"]
```

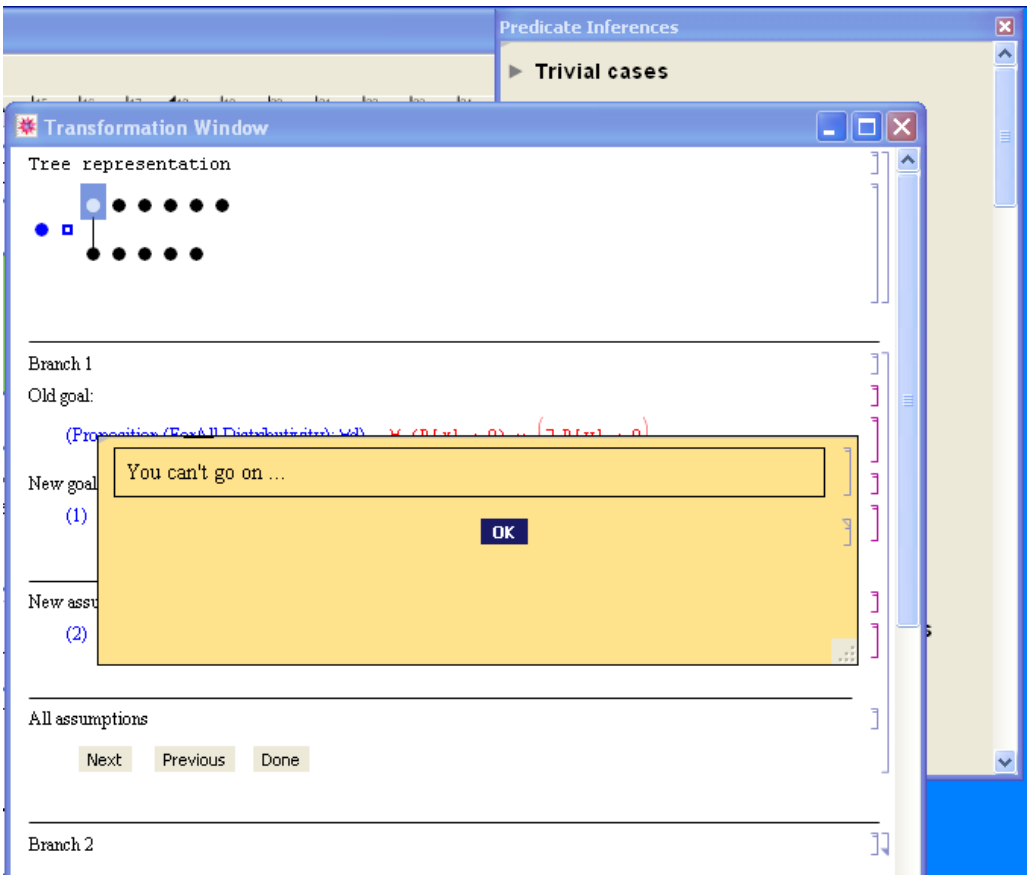
Let' s start the proof.

```
Prove[Proposition["ForAll Distributivity"], by → PredicateProver,
  transformBy → ProofSimplifier, TransformerOptions → {steps → Useful},
  showBy → RuleInspector, ShowOptions → {Check → "Strict"}]
```

- ProofObject -

This time we are going to use the restriction of the inspection, notice that `ShowOptions → {Check → "Strict"}` is given.

The difference in the proof flow is that one cannot go further until the correct rule is picked. Reaching a "Transformation Window" the system waits for the correct rule applied in the given proof step. Neither clicking in the Proof tree nor clicking on the navigation buttons causes changes except getting a warning message like on the following picture:



The user is stuck in this situation until clicking on the 'Prove Equivalence' rule.

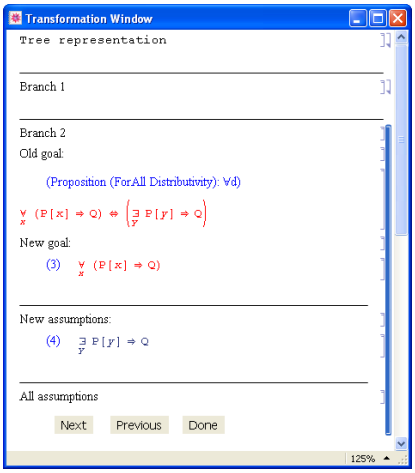
As a small hint, look at the following command.

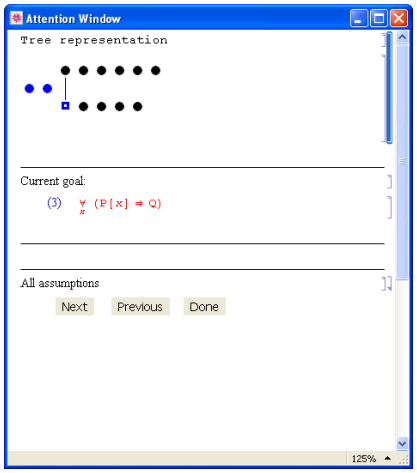
```
Cases[{$TmaProofObject},
    Theorema‘Provers‘Common‘ProofObject‘Private‘ProofInfo[kw_, __] :→ kw, ∞] // FullForm
```

```
List["InitialProofSituation", "NewGoal", "ProveEquivalence", "NewGoal", "Simplify",
  "ProveImplication", "NewGoal", "Skolem", "SimplifyImplicationLHS", "Simplify",
  "ContradictionUniversal", "NewGoal", "ArbitraryFixed", "NewGoal", "ProveImplication",
  "NewGoal", "SimplifyImplicationLHS", "Simplify", "ContradictionUniversal"]
```

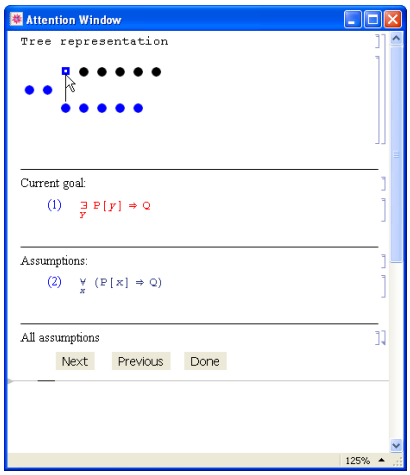
This command takes the current value of the $TmaProofObject and outputs the list of the rules applied during the proof. In this list there are some so-called blind steps, rules that are only applied and not shown in the proof, e.g 'NewGoal'.

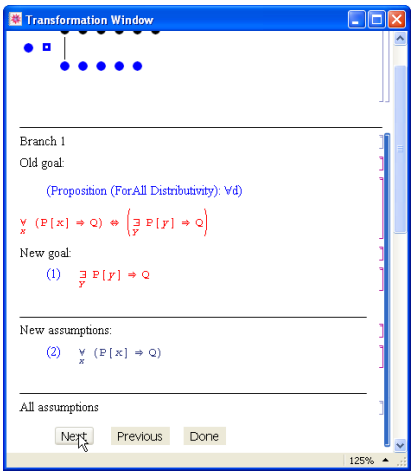
After pushing the "ProveEquivalence " button, one should choose a branch. Now we chose the second one.





Note, by reaching the last bullet in the proof tree, clicking on 'Next' will continue the proof inspection from the beginning, in an Attention Window. In order to go on the other branch, one has to possible ways, either click on the first bullet of the branch in the proof tree or push 'Next' until entry of the branch is available again.





(All proof steps are shown in ***Appendix A.2***)

5.4 An Example with Interpreted Formulae (Elementary Analysis)

5.4.1 The Limit of a Sum

By this example we would like to show some logical statement where the predicates have meaning. We build up a complex knowledge base and we present the differences between the levels depending on the quantifiers. We are proving with the PredicateProver again.

```
SetGlobals[Prover → PredicateProver]
```

5.4.1.1 Initializations

In the spirit of "structured programming", we build the definition of limit up in a hierarchy of definitions, one quantifier at a time:

```
Definition["2−ary limit",
    ∀_{f,a} ((limit[f, a]) : ⇔ (∀_ϵ limit[f, a, ϵ]))   "lim2:"]
```

```
Definition["3−ary limit",
    ∀_{f,a,ϵ} ((limit[f, a, ϵ]) : ⇔ ∃_N limit[f, a, ϵ, N])   "lim3:"]
```

```
Definition["4−ary limit",
    ∀_{f,a,ϵ,N} ((limit[f, a, ϵ, N]) : ⇔ (∀_n (n ≥ N) ⟹ (|f[n] − a| < ϵ)))   "lim4:"]
```

Auxiliary knowledge:

```
Definition["sum of sequences", any[f, g, x],
    ((f ⊕ g)[x]) := (f[x] + g[x])   "⊕:"]
```

```
Proposition["distance of sum", any[x, y, a, b, δ, ϵ],
    (|x − a| < δ ⋀ |y − b| < ϵ) ⟹ (|(x + y) − (a + b)| < δ + ϵ)   "|++|"]
```

```
Proposition["max", any[M, m, n],
    (M ≥ max[m, n]) ⟹ (M ≥ m ∧ M ≥ n)   "max:"]
```

5.4.1.2 Limit4

We are creating the knowledge base, like written in chapter 3.

```
Theory["4−ary limit",
    Definition["4−ary limit"]
    Definition["sum of sequences"]
    Proposition["distance of sum"]
    Proposition["max"]]
```

Ultimately, we want to prove

Proposition["2−ary limit of sum", any[f, a, g, b],

((limit[f, a] ⋀ limit[g, b]) "lim2+"
⟹ limit[f ⊕ g, a + b])]

The decomposition of the limit definition suggests building up the proposition (and its proof) in a hierarchy of propositions (and proofs):

Proposition["4−ary limit of sum",

$\underset{f,a,\delta,M,g,b,\epsilon,N}{\forall}$ ((limit[f, a, δ, M] ⋀ limit[g, b, ϵ, N]) "lim4+"
⟹ limit[f ⊕ g, a + b, δ + ϵ, max[M, N]])]

In case the ProofSimplifier was not set as the default transformer of the system then evaluate the following cell:

SetOptions[Transform, by → ProofSimplifier]

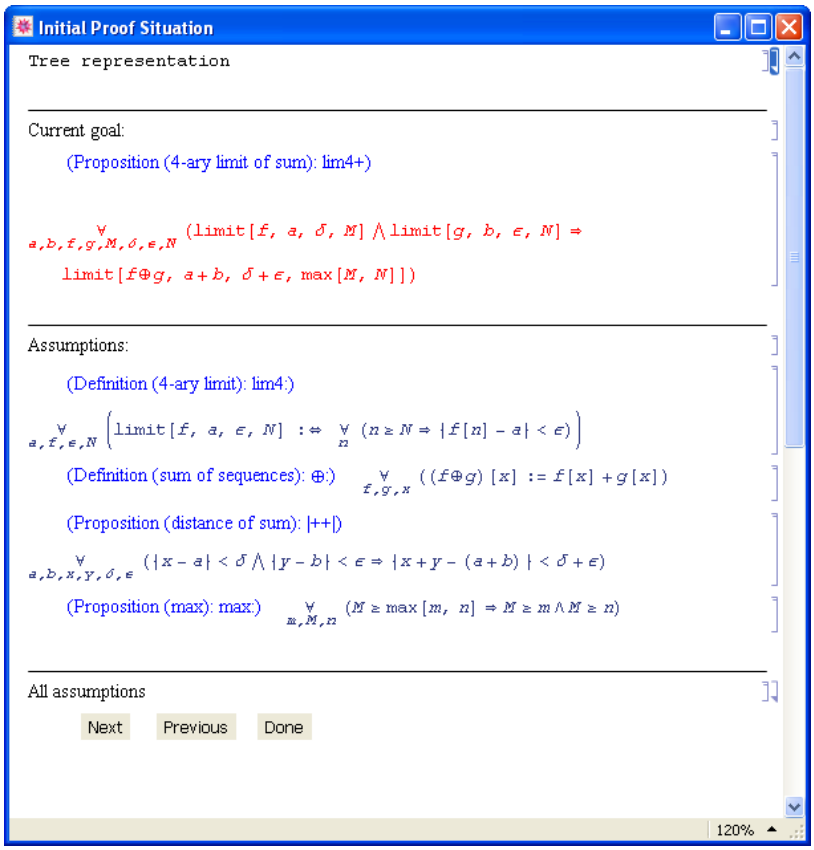
{by → ProofSimplifier, TransformerOptions → {}, showBy → StaticWriter, ShowOptions → {}}

? steps

Option of ProofSimplifier with possible values: All,
Essential, Combined, Useful, Lifted, LiftedParallel and list combinations of th

Prove[Proposition["4−ary limit of sum"], using → Theory["4−ary limit"], SearchDepth → 50,
showBy → RuleInspector, TransformBy → ProofSimplifier, steps → {Essential}]

- ProofObject **-**



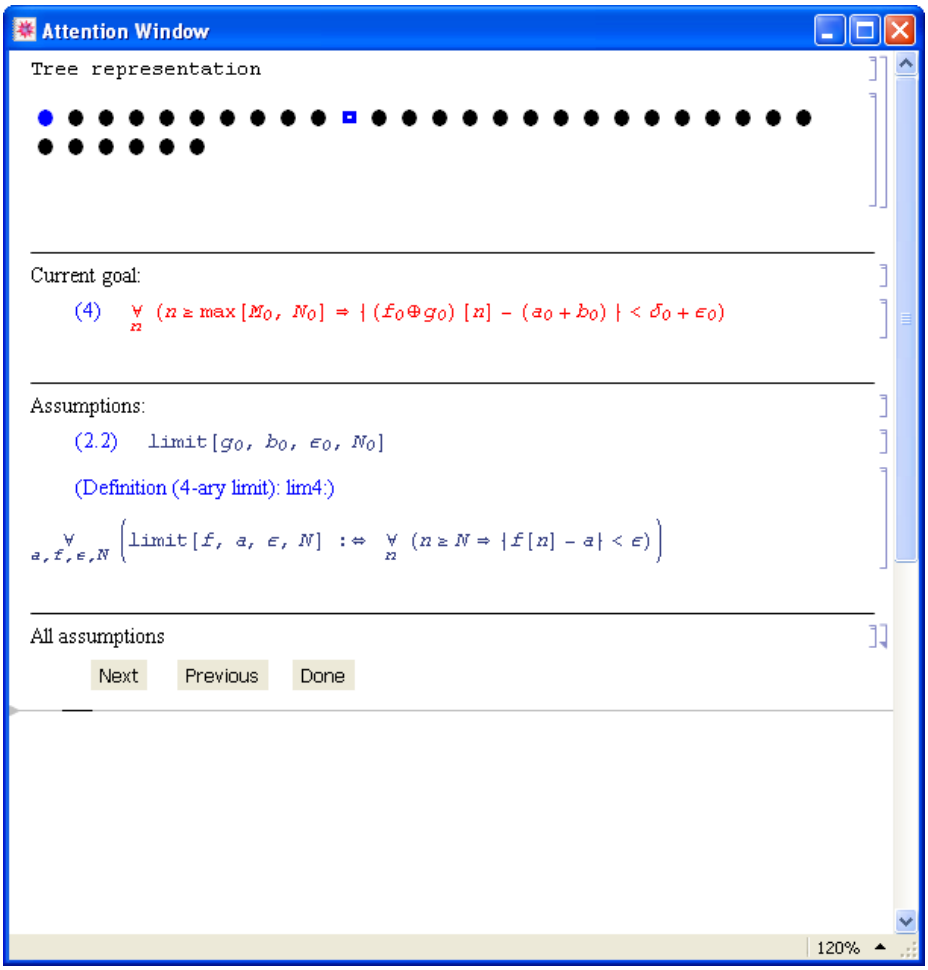
Looking at the proof tree, one can see, this proof is linear and it has 33 steps. In

order to compare the result, we collected the rules from each proof. For Proposition["4-ary limit of sum"] they are the following:

```
Cases[{$TmaProofObject},
    Theorema`Provers`Common`ProofObject`Private`ProofInfo[kw_, __] :> kw, ∞] // FullForm
```

```
List["InitialProofSituation", "NewGoal", "ArbitraryFixed", "NewGoal", "ProveImplication",
  "NewGoal", "AssumptionIsConjunction", "ExpandGoalByDef", "NewGoal",
  "ExpandAssmByDef", "ExpandAssmByDef", "ArbitraryFixed", "NewGoal", "ProveImplication",
  "NewGoal", "TransformConclusionBy", "NewGoal", "ObtainFromBy", "DoneMatching",
  "AssumptionIsConjunction", "ObtainFromBy", "ObtainFromBy", "DoneMatching",
  "ObtainFromBy", "ObtainFromBy", "DoneMatching", "ObtainFromBy", "ObtainFromBy",
  "ObtainFromBy", "ObtainFromBy", "DoneMatching", "ConclusionIsAssumption"]
```

An other one of the proofs steps:



5.4.1.3 Limit3

The next level of abstraction, 3 arguments.

```
Theory["3−ary limit",
        Definition["3−ary limit"]
        Proposition["4−ary limit of sum"]]
```

```
Proposition["3−ary limit of sum",
    ∀_{f,a,δ,g,b,ϵ} ((limit[f, a, δ] ⋀ limit[g, b, ϵ])  "lim3+"
        ⟹ limit[f ⊕ g, a + b, δ + ϵ])]
```

```
Prove[Proposition["3−ary limit of sum"], using → Theory["3−ary limit"], showBy → RuleInspector]
```

```
- ProofObject -
```

22 linear steps, with the following list of rules applied :

```
Cases[{$TmaProofObject},
    Theorema`Provers`Common`ProofObject`Private`ProofInfo[kw_, __] :→ kw, ∞] // FullForm
```

```
List["InitialProofSituation", "NewGoal", "ArbitraryFixed", "NewGoal",
  "ProveImplication", "NewGoal", "AssumptionIsConjunction", "ExpandGoalByDef",
  "NewGoal", "ExpandAssmByDef", "Skolem", "ExpandAssmByDef", "Skolem",
  "ObtainFromBy", "ObtainFromBy", "DoneMatching", "ObtainFromBy", "ObtainFromBy",
  "ObtainFromBy", "ObtainFromBy", "DoneMatching", "InstanceOfExistential"]
```

```
Timing[Prove[Proposition["3−ary limit of sum"], using → Theory["3−ary limit"]]]
```

```
{1.235, - ProofObject -}
```

5.4.1.4 Limit2

Back to the starting, desired level.

```
Proposition["3−ary limit of sum: variant",
    ∀_{f,a,g,b,ϵ} ((limit[f, a, ϵ/2] ⋀ limit[g, b, ϵ/2])  "lim3+"
        ⟹ limit[f ⊕ g, a + b, ϵ])]
```

```
Theory["2−ary limit",
    Definition["2−ary limit"]
    Proposition["3−ary limit of sum: variant"]]
```

```
Proposition["2−ary limit of sum",
    ∀_{f,a,g,b} ((limit[f, a] ⋀ limit[g, b])  "l2+"
        ⟹ limit[f ⊕ g, a + b])]
```

```
Prove[Proposition["2−ary limit of sum"], using → Theory["2−ary limit"],
  ProverOptions → {BackChaining → True}, showBy → RuleInspector]
```

```
- ProofObject -
```

22 steps again, but one can see, different rules were used during the proof.

```
Cases[{$TmaProofObject},
    Theorema`Provers`Common`ProofObject`Private`ProofInfo[kw_, __] :→ kw, ∞] // FullForm
```

```
List["InitialProofSituation", "NewGoal", "ArbitraryFixed", "NewGoal", "ProveImplication",
  "NewGoal", "AssumptionIsConjunction", "ExpandGoalByDef", "NewGoal",
  "ExpandAssmByDef", "ExpandAssmByDef", "ArbitraryFixed", "NewGoal",
  "DoneMatching", "ProofByContradiction", "NewGoal", "ObtainFromBy", "DoneMatching",
  "AssumptionIsNegatedConjunction", "UniversalResolution", "ContradictionUniversal"]
```

```
Timing[Prove[Proposition["2−ary limit of sum"],
    using → Theory["2−ary limit"], ProverOptions → {BackChaining → True}]]
```

```
{1., - ProofObject -}
```

5.5 Experiences

While studying this examples, we presented a few possible application of the **RuleInspector**. We believe, that our tool can help to have a closer experience, a more detailed image about the proof mechanism and one can inspect non-trivial cases.

Chapter 6

Future work

We have presented how the RuleInspector tool works for predicate and propositional logic. We want to extend this to other provers in *Theorema*. Currently, the inference rule palette is created manually, the next desirable step is to automatically extract the inference rules that are available to the prover which produced the proof the user inspects.

Another possibility of future work is introducing a set of logging routines to register the number of tries of a student before the correct inference name is picked from the available list. A teacher may later consult the logs, compare them in time to observe how the student' s knowledge of a studied theory improves.

We can also implement advanced inspection through which the user can choose the inference rule applied in the attention phase. This method includes pseudo-false answers, which are theoretically possible but not the ones used at the time.

Literature

[Buchberger00a] B.Buchberger. *Focus Windows Presentation: A New Approach to Presenting Mathematical Proofs (in Automated Theorem Proving Systems)*.TheoremaTechnical Report Research Institute for Symbolic Computation, January 30, 2000.

[Buchberger01] B.Buchberger.*The PCS Prover in Theorema.* In R.Moreno - Diaz, B.Buchberger, and J.L.Freire, editors, Proceedings of EUROCAST 2001 (8 th International Conference on Computer Aided Systems Theory - Formal Methods and Tools for Computer Science), Lecture Notes in Computer Science 2178, pages 469-478. Las Palmas de Gran Canaria, Copyright: Springer-Verlag Berlin, 19-23 February 2001.

[BuchbergerEtAl97] *A Survey of the Theorema project*. B.Buchberger, T.Jebelean, F.Kriftner, M.Marin, E.Tomuta, D.Vasaru. In: Proceedings of *ISSAC' 97* (*International Symposium on Symbolic and Algebraic Computation*, Maui, Hawaii, July 21-23, 1997), W.Kuechlin (ed.), ICM Press 1997, pp.384-391.ISBN 0-89791-875-4

[BuchbergerEtAl00] *The Theorema Project: A Progress Report.* B.Buchberger, C.Dupre, T.Jebelean, F.Kriftner, K.Nakagawa, D.Vasaru, W.Windsteiger. In: Symbolic Computation and Automated Reasoning (Proceedings of *CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, August 6-7, 2000, St.Andrews, Scotland), M.Kerber and M.Kohlhase (eds.), A.K.Peters, Natick, Massachusetts, pp.98-113. ISBN 1-56881-145-4.

[BuchbergerEtAl06] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. *Theorema: Towards Computer-Aided Mathematical Theory Exploration*. Journal of Applied Logic, 4(4): 470-504, 2006.

[BuchbergJeb97] B.Buchberger and T.Jebelean. *Theorema: The Predicate Logic Prover*. In First International Theorema Workshop, RISC, Hagenberg, Austria, June 9-10 1997.

[BuchbergPiroi02] F. Piroi and B. Buchberger. *Focus windows: A new technique for proof presentation*. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Compu·*

tation. Proceedings of *Joint AISC'2002 - Calculemus'2002 Conference*, volume 2385 of LNAI, pages 290-304, Marseille, France, July 2002. Springer Verlag.

[MathAgents08] Florina Piroi, Bruno Buchberger, Camelia Rosenkranz. *Mathematical Journals as Reasoning Agents: Literature Review*. Technical report no. 08-05 in RISC Report Series, University of Linz, Austria. March 2008.

[Piroi04] F. Piroi. *Tools for Using Automated Provers in Mathematical Theory Exploration*. PhD thesis, RISC, Johannes Kepler University, Linz, Austria, August 2004.

[PiroiKutsia05] Florina Piroi and Temur Kutsia. *The Theorema Environment for Interactive Proof Development*. In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*. Proceedings of the *12th International Conference, LPAR'05*, volume 3835 of Lecture Notes in Artificial Intelligence, pages 261-275. Springer Verlag, 2005. RISC2795.

[SinkaPiroi09] Zs. Sinka and F.Piroi. *Teaching Predicate Logic with Theorema*. Informal proceedings of the First International *AUTOMATHEO'09 Workshop*, Hagenberg, Austria, June 29-30, 2009. pp 1-5.

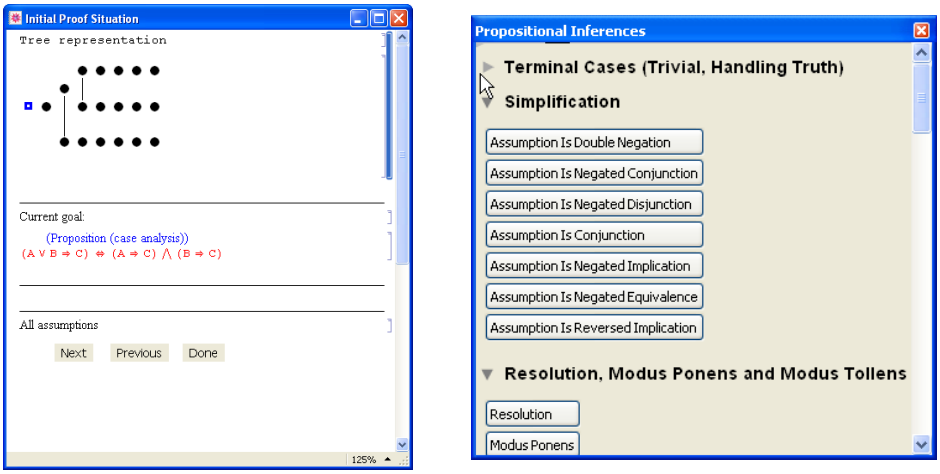
[Windsteiger06] W. Windsteiger. An Automated Prover for Zermelo-Fraenkel Set Theory in Theorema. JSC, 41(3-4): 435-470, 2006. RISC2788.

[Wolfram03] S. Wolfram. *The Mathematica Book*. Wolfram Media Inc., 5th edition, 2003.

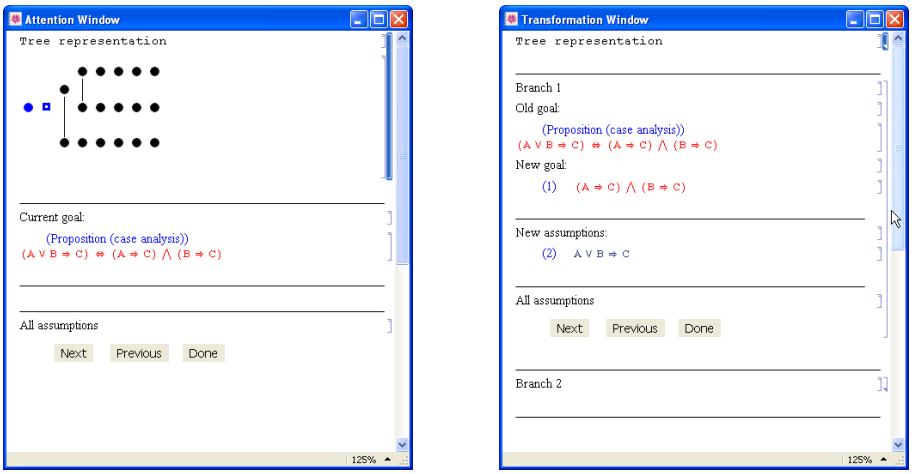
Appendix

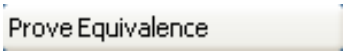
A.1 Propositional Example screenshots

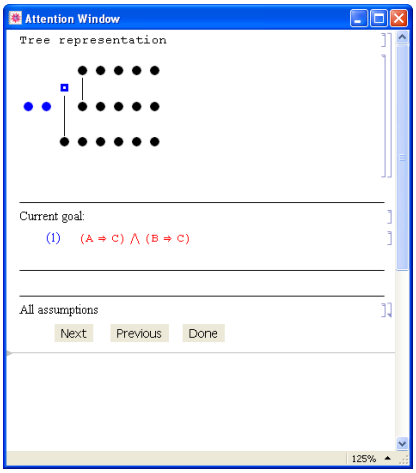
The entire proof of "*case analysis*" is shown, where each proof step is illustrated by the Attention and Transformation Windows together with the rule was applied in the particular step.

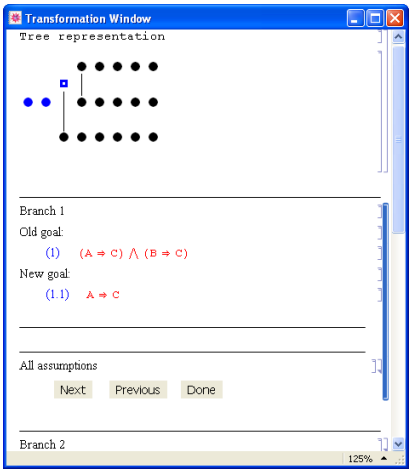


Step 1: initialization

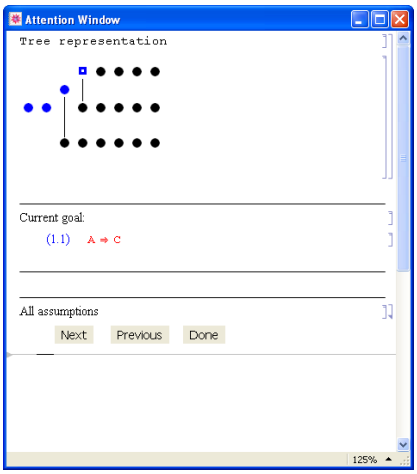


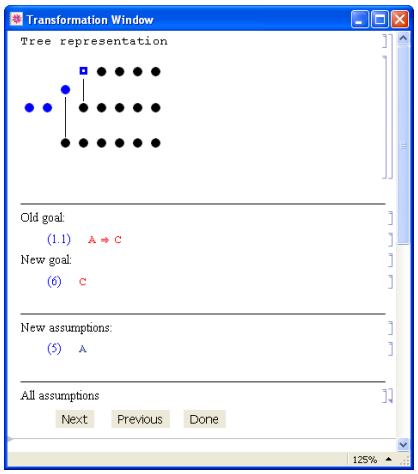
Step 2 : 



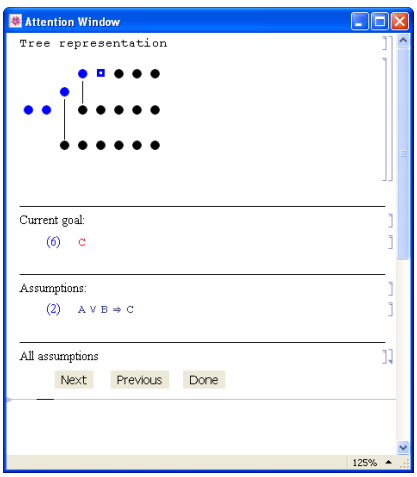


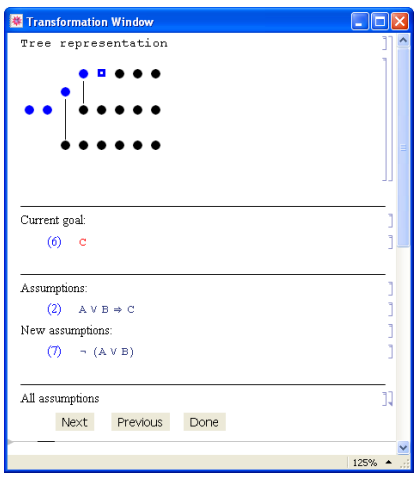
Step 3 : Prove Conjunction



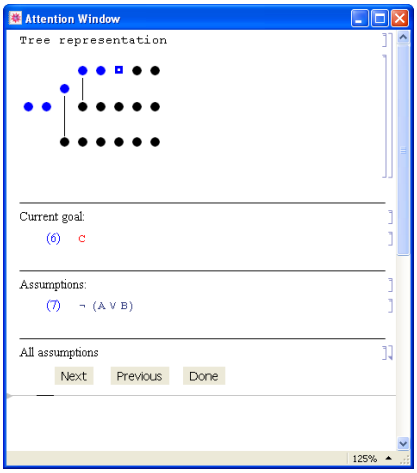


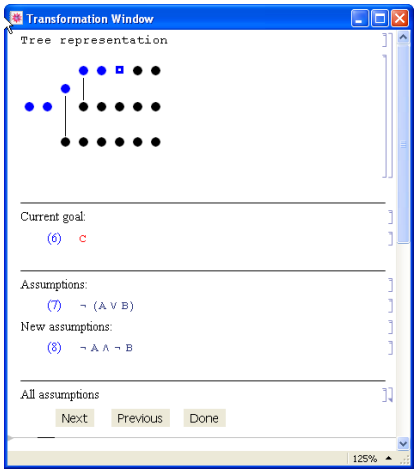
Step 4 : Prove Implication



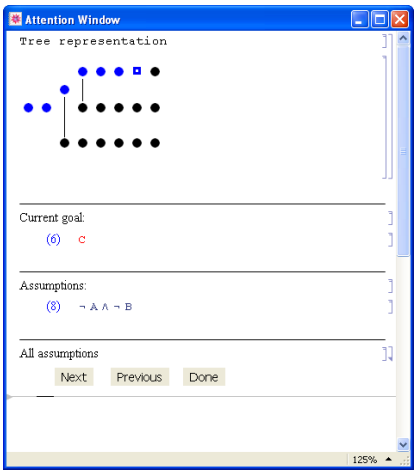


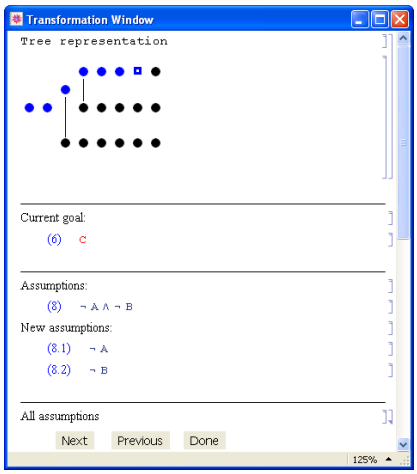
Step 5 : Simplify Implication LHS



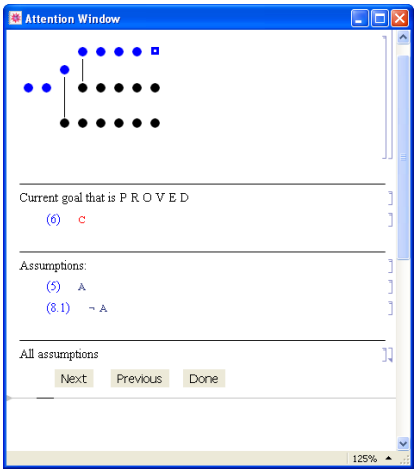


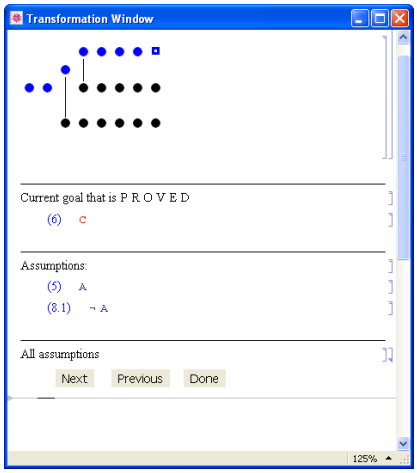
Step 6 : Assumption Is Negated Disjunction



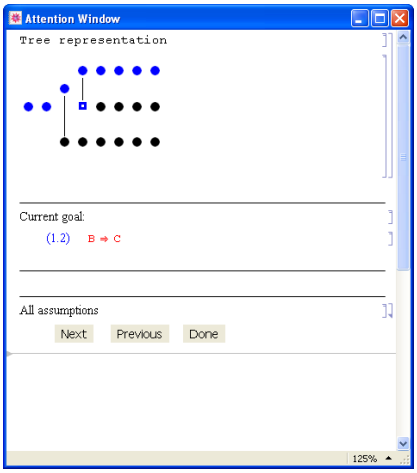


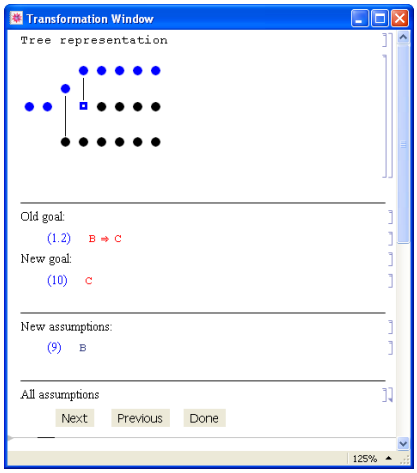
Step 7 : Assumption Is Conjunction



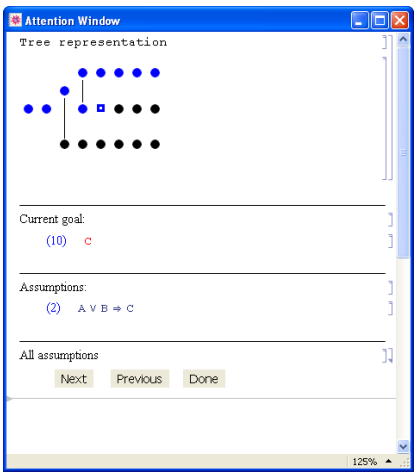


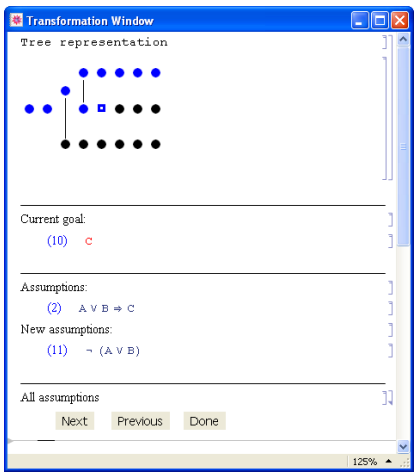
Step 8 : Contradictory Assumptions



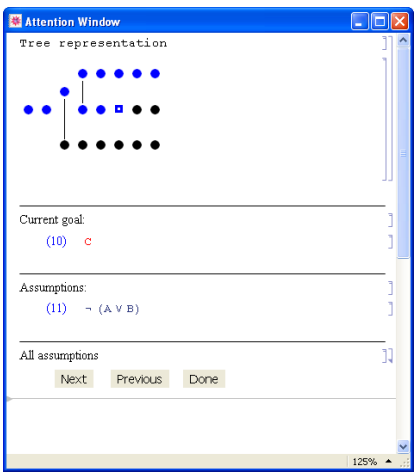


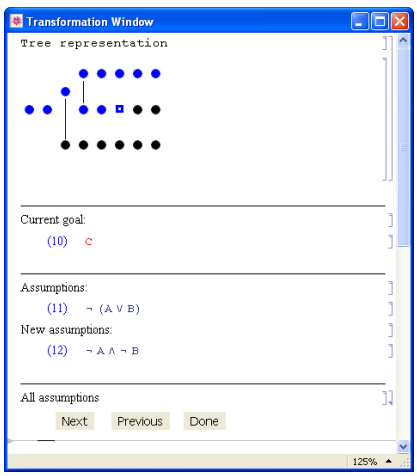
Step 9 : Prove Implication



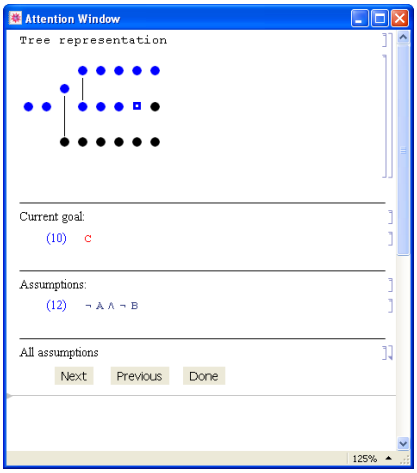


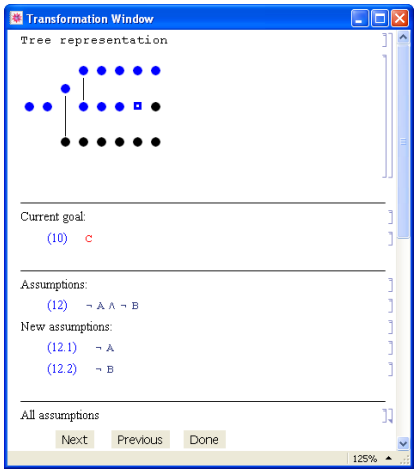
Step 10 : Simplify Implication LHS



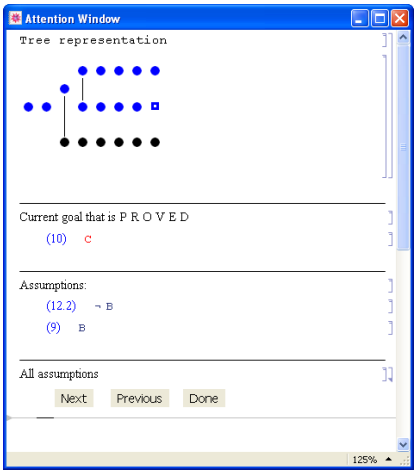


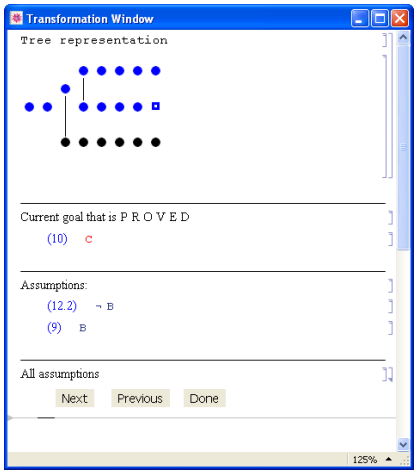
Step 11 : Assumption Is Negated Disjunction



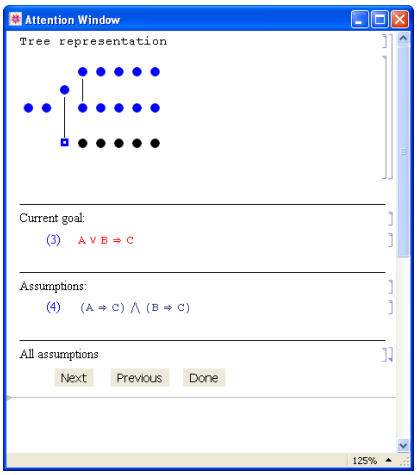


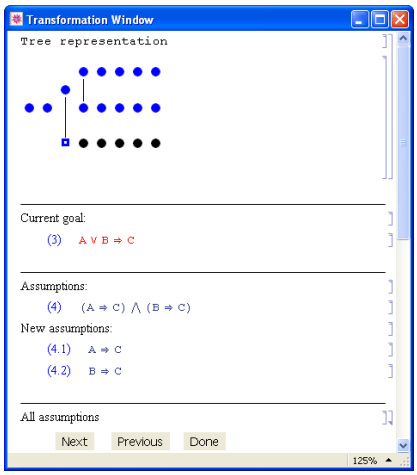
Step 12 : Assumption Is Conjunction



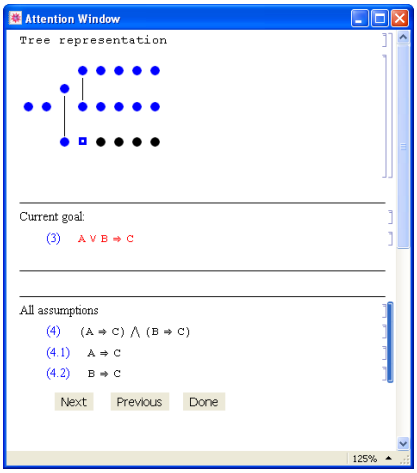


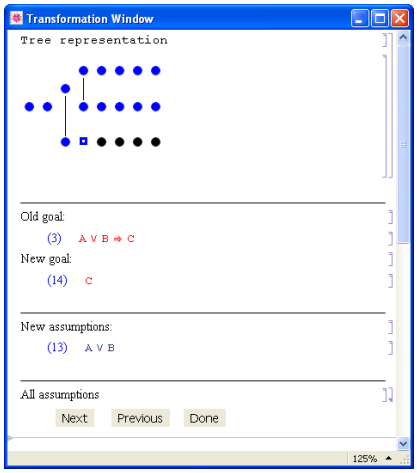
Step 13 : Contradictory Assumptions



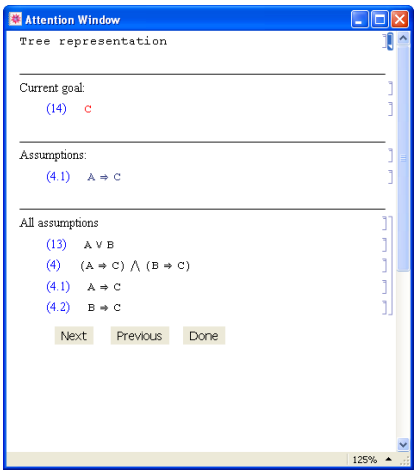


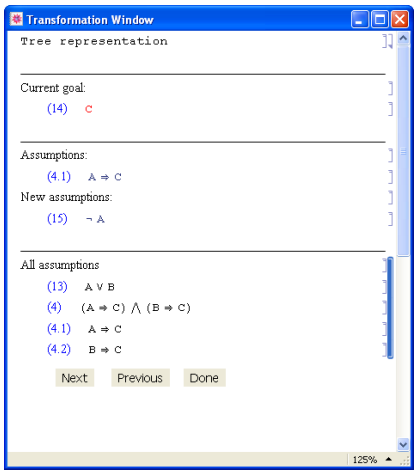
Step 14 : Assumption Is Conjunction



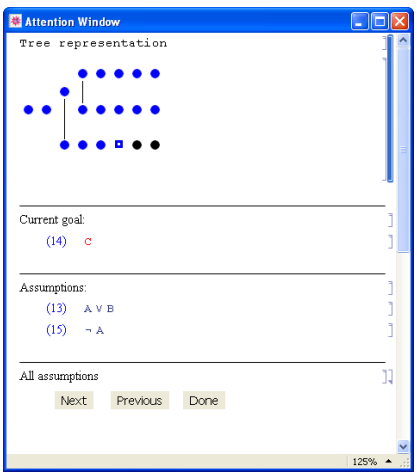


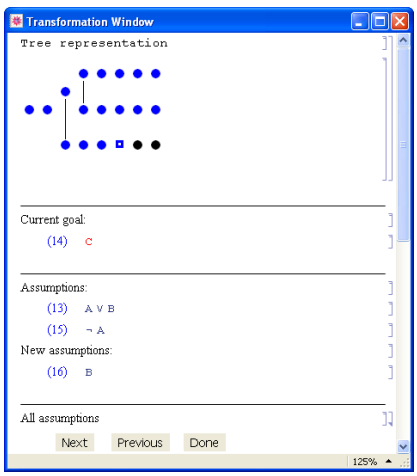
Step 15 : Prove Implication



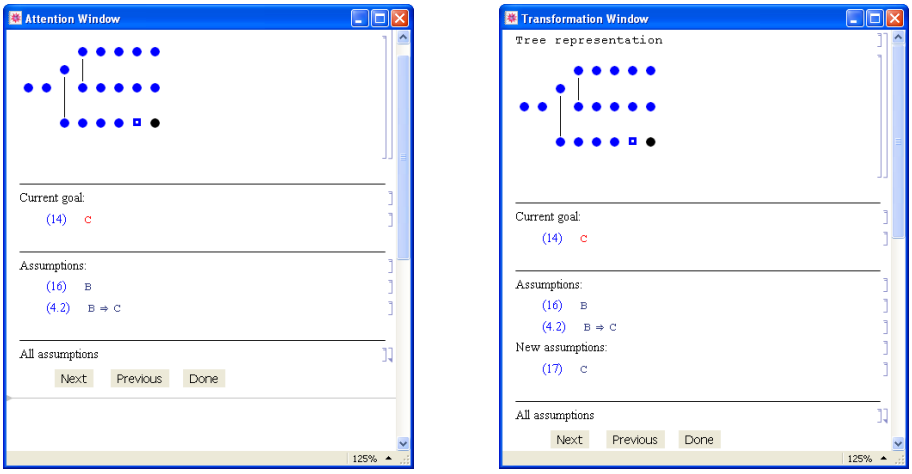


Step 16 : Simplify Implication LHS

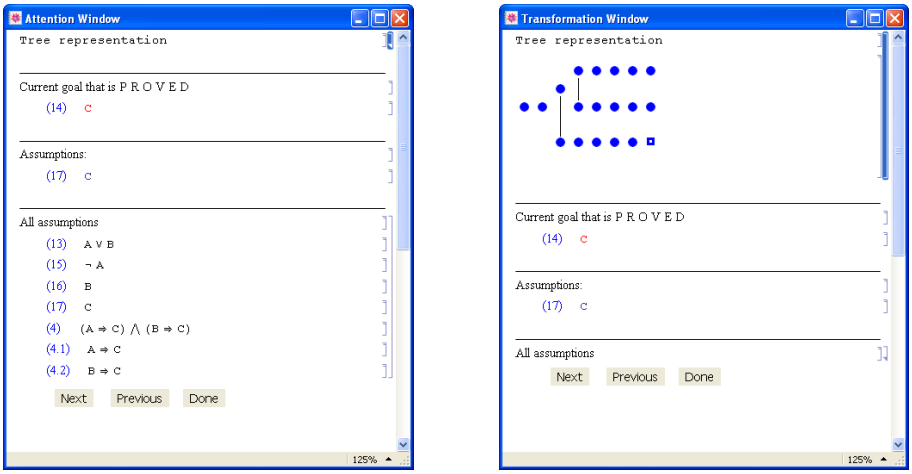




Step 17 : Resolution



Step 18 : Modus Ponens

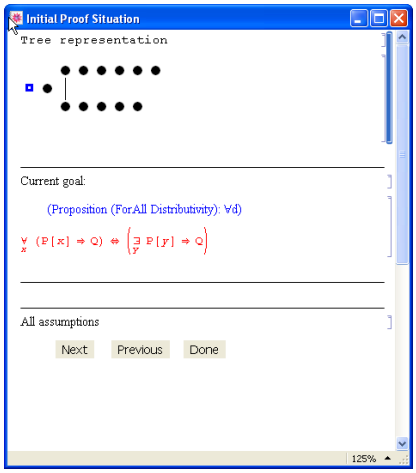


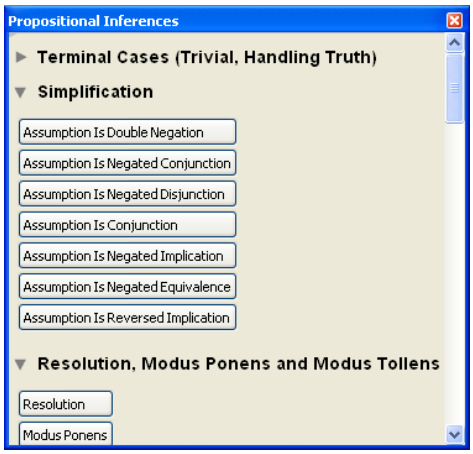
Step 19 : Conclusion Is Assumption

The End

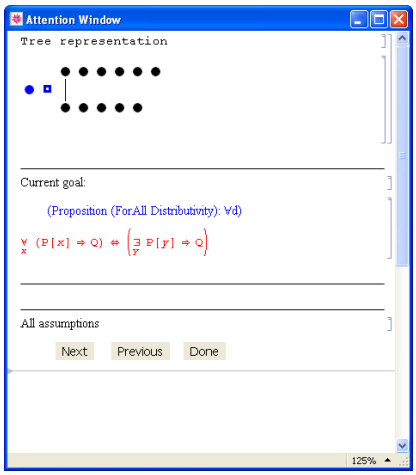
A.2 Predicate Example

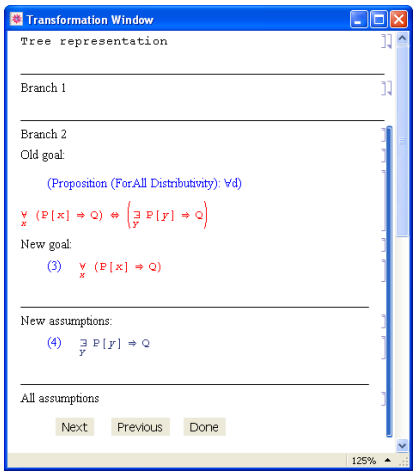
Here we show the entire proof of "*ForAll Distributivity*". Each proof step is illustrated by the Attention and Transformation Windows, and below them the rule applied in the particular step.



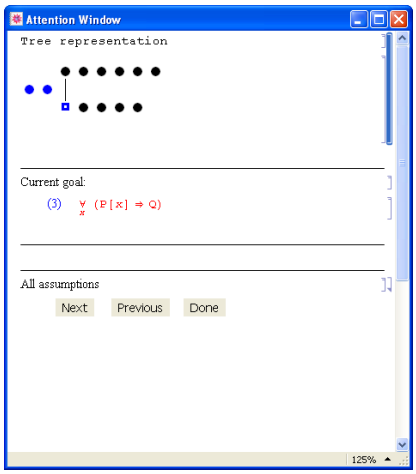


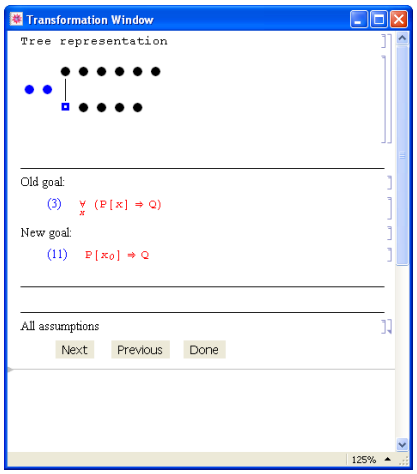
Step 1: initializations



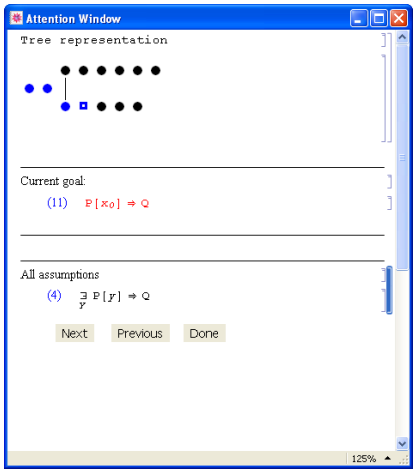


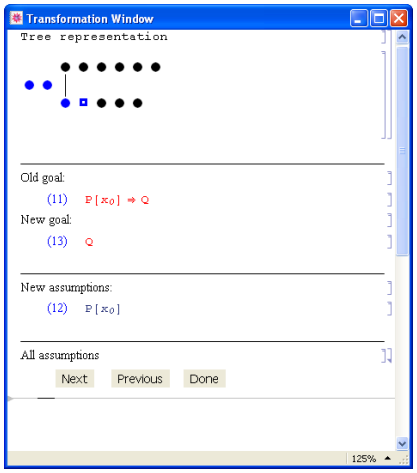
Step 2 Prove Equivalence



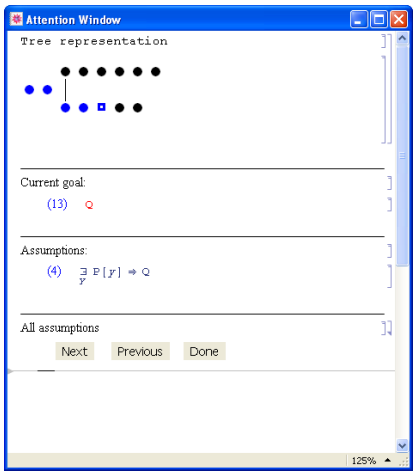


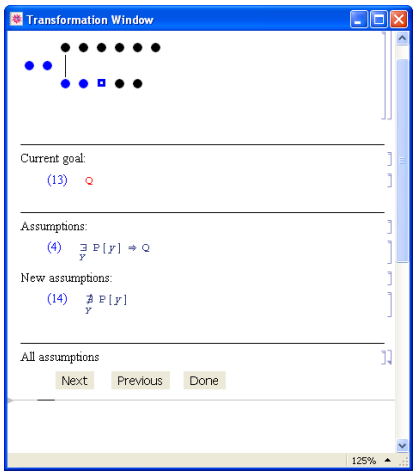
Step 3: Simplify



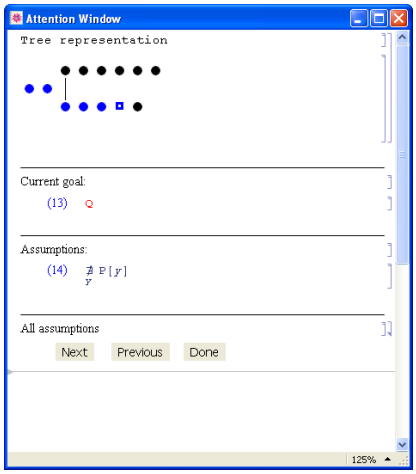


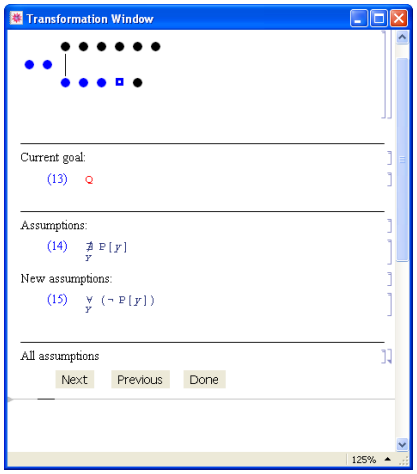
Step 4: Prove Implication



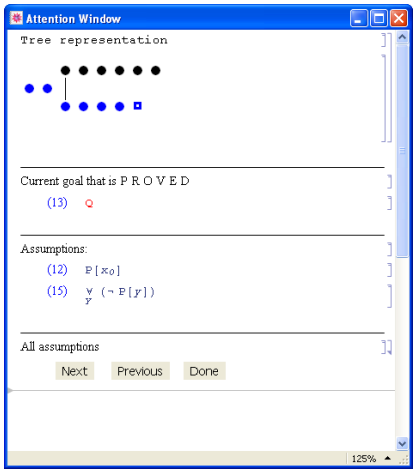


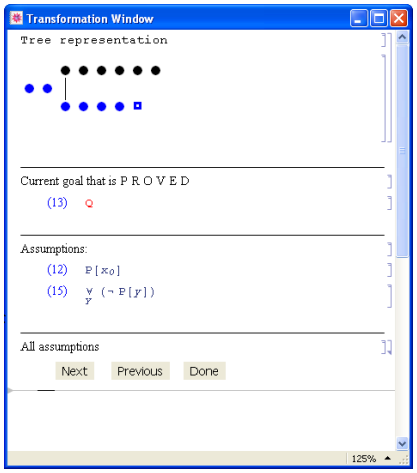
Step 5: Skolem



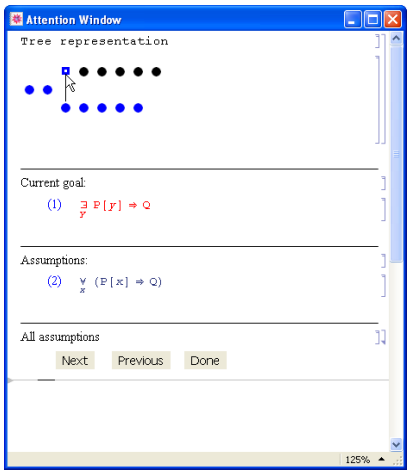


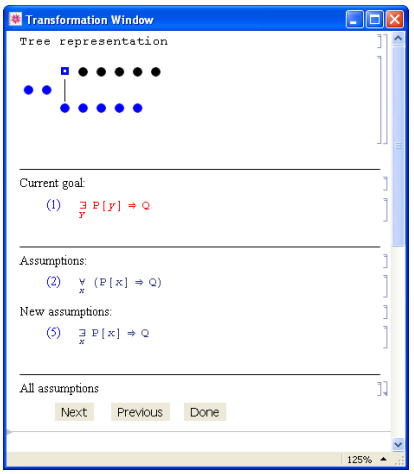
Step 6: Simplify Implication RHS



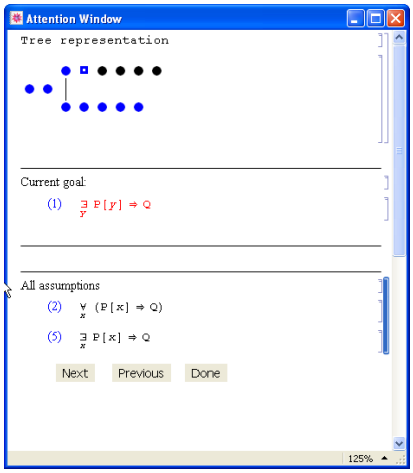


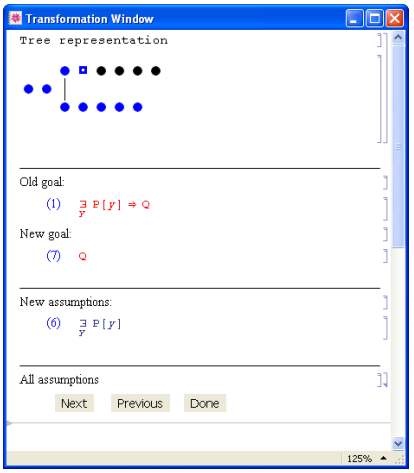
Step 7: Simplify



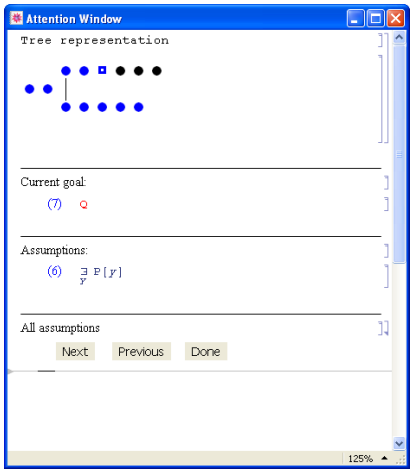


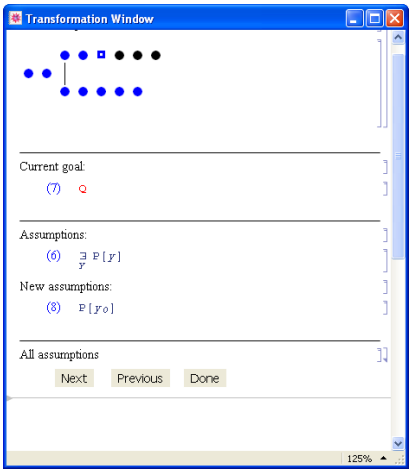
Step 8: Contradiction Universal



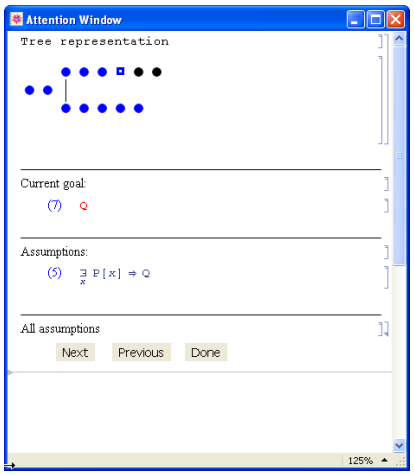


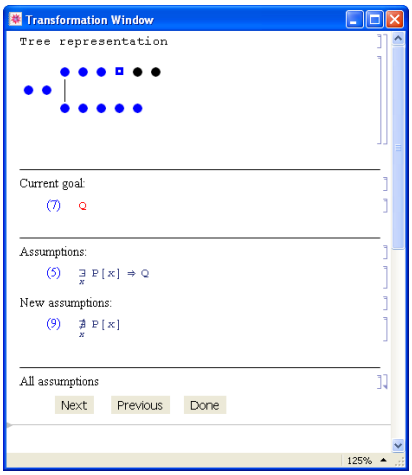
Step 9: Arbitrary Fixed



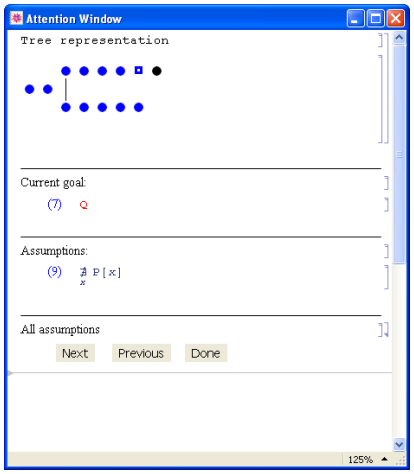


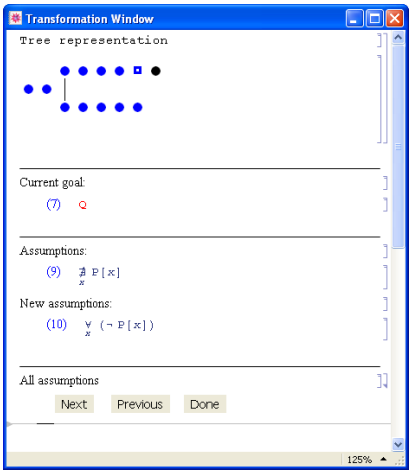
Step 10: Prove Implication



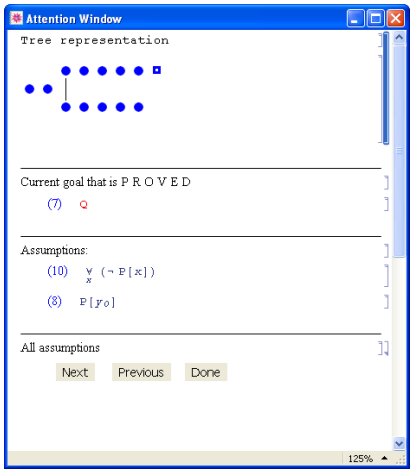


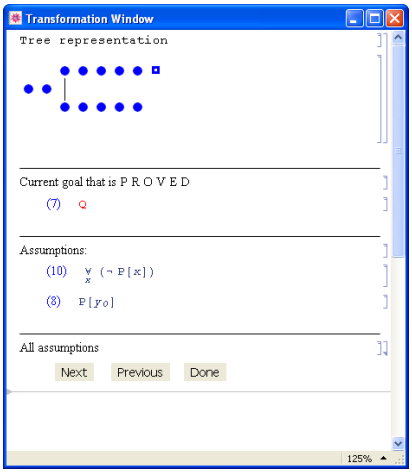
Step 11: Simplify Implication LHS





Step 12: Simplify





Step 13: Contradiction Universal

The End

Curriculum Vitae

PERSONAL DATA

Name	Zsuzsanna Sinka
Phone	+43 650 347 81 79
E-mail	zsinka@risc.uni-linz.ac.at
Date of birth	14/06/1985
Place of birth	Esztergom, Hungary
Nationality	Hungarian
Address	Pataksor Alsó Street 31, Dorog, 2510 Hungary

SCHOOLS AND REFERENCES

2008 -2009	International Master's Program for Informatics at the Johannes Kepler University (Linz)
2008 -	Student lecturer, teaching Basics of Formal Mathematical Logic (practice course)
2008 -	Educational assistant for educational vice dean
2007 -	Mentoring first year students in various subjects (e. g. Linear algebra and geometry, Introduction to Mathematics and Programming)
2005 – 2008	Students' Union at the Eötvös Loránd University: managing enrolment, arranging and paying social scholarships, organising student community events of the IT faculty at the university (for example open days, Neumann-day)
2003 -	Eötvös Loránd University (Budapest), Faculty of Informatics, Programmer Mathemetician (MSc) degree expected in 2010
1999 – 2003	Dobó Katalin Secondary Grammar School (Esztergom)

SKILLS

Spoken languages

English

Germany (intermediate state language exam, type C)

Programming languages, technologies

Logical programming languages, Prolog , Mathematica, Theorema

Formal logic (automatic thesis proving, multi-valued logic, temporary logic, Fuzzy logic, table methods), Computability theory

Predicate logic, Theorema

AI (inductive systems, knowledge based technologies and expert systems, robotics)

C# and .NET 2.0, C, C++/QT, Ada95, x86 Assembly, Oracle, pl/SQL, mySQL, PHP, HTML, XML, UML, OO design and design patterns

Fields of interests

Logic ; automated reasoning, logical verification

Publication

Zs. Sinka and F.Piroi. Teaching *Predicate Logic with Theorema*. Informal proceedings of the First International *AUTOMATHEO'09 Workshop*, Hagenberg, Austria, June 29-30, 2009. pp 1-5.