Interactive Theorem Proving with Tasks

M. Hübner[1] S. Autexier[2] C. Benzmüller[2] A. Meier[2]

[1] *Max-Planck-Institut für Informatik*
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

[2] *Department of Computer Science, Saarland University,*
and DFKI Saarbrücken,
66041 Saarbrücken, Germany

Abstract

Interactive theorem proving systems for mathematics require user interfaces which allow for user interaction that is as natural as possible. However, this interaction is often limited by the traditional calculi underlying most theorem proving systems. This is particularly problematic with respect to the application of assertions and intuitive presentation of proof states. In this paper we show how a more flexible user interaction can be realized when traditional calculi for classical logic are replaced by a less restrictive reasoning engine, the recently developed CORE [2] system. We describe the *task level* which is built on top of the CORE system and combines the Proof by Pointing approach [5] with a flexible mechanism for the application of assertions that avoids decomposition and abstracts from the syntactical form of an assertion. We demonstrate how proof steps that are difficult to implement in other systems, like forward application of assertions, are quite naturally supported by the underlying CORE system and are therefore straightforward to realize at the task level.

Key words: Interactive Theorem Proving, Tasks, Proof by Pointing, Assertion Application

1 Introduction

Interactive theorem provers (ITPs) are used in a variety of domains, ranging from proof assistants for formal methods to mathematical assistant systems for tutoring mathematics. Successful application of ITPs in these domains requires to support a human oriented reasoning style. Most ITPs are therefore based on natural deduction (ND) or sequent calculus (SK) reasoning engines

[1] Email:huebner@mpi-sb.mpg.de
[2] Email:chris@ags.uni-sb.de,{autexier|ameier}@dfki.de

which match the human reasoning style better than calculi developed for automated reasoning (e.g. the resolution principle). However, because these calculi have been invented for proof-theoretic purposes there are still many problems inherent to interactive proof frameworks based on these calculi. Two particularly prominent problems with respect to interactive theorem proving are the *presentation* of proofs and the *application of assertions*

These problems become immediately apparent when we look at a sample sequent calculus proof (Fig. 1). We see that the original goal formula becomes

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \Rightarrow Q, Q \Rightarrow R, P \vdash P \quad P \Rightarrow Q, Q \Rightarrow R, P, Q \vdash Q}{P \Rightarrow Q, Q \Rightarrow R, P \vdash Q} \qquad P \Rightarrow Q, Q \Rightarrow R, P, R \vdash R}{P \Rightarrow Q, Q \Rightarrow R, P \vdash R}}{P \Rightarrow Q, Q \Rightarrow R \vdash P \Rightarrow R}}{(P \Rightarrow Q) \wedge (Q \Rightarrow R) \vdash P \Rightarrow R}}{\vdash (P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow P \Rightarrow R}$$

Fig. 1. A sequent calculus proof.

decomposed throughout the proof which makes proof presentation difficult. For the same reason it is difficult to support the application of assertions (formulas on the left hand side of a sequent) in an intuitive manner. In particular, the application of an assertion depends on its syntactical structure. For example, the steps necessary to apply $P \Rightarrow Q$ and $\neg P \vee Q$ are different in a sequent calculus although $P \Rightarrow Q$ and $\neg P \vee Q$ are logically equivalent. Similar problems arise when proofs are constructed in the ND calculus.

To improve user interaction with systems relying on the SK or ND calculus much work was undertaken to present proofs in a more user friendly way and to hide the calculus as much as possible from the user.

Bornat [6] describes a system in which linearized versions of sequent calculus proofs are presented in a Box-and-Line form to the user. Proof steps can be expressed at the Box-Line level and are then mapped into a sequence of SK rule applications. However, in this approach, the disadvantages of the underlying SK show trough to the level of proof presentation. For example, to enable the user to carry out forward steps every such step has to be painstakingly mapped into the SK calculus via the application of a cut-rule[3].

The *Proof by Pointing* approach by Bertot et al. [5] also tries to hide the inconveniences of the SK from the user. They use an annotated set of SK rules to enable the user to select a subformula of a sequent. The system is then able to extract this subformula by automatic application of the appropriate SK rules. However, in this approach, manipulating a subformula still requires the decomposition of the overall formula.

It can be seen that user interaction in the above approaches is severely restricted by the calculus on which they are based. In this paper we show how

[3] In Bornat's system cuts are introduced by a tactic.

a more flexible user interaction can be realized when the underlying ND or SK calculus is replaced by a less restrictive reasoning engine.

We use the recently developed CORE [2] system as a logical basis. CORE already supports flexible reasoning at the assertion level in a *contextual rewriting* proof style. In this paper we develop a communication layer, the so called *task layer*, on top of the system. This layer exploits COREs strength in the application of assertions and provides an additional mechanism for the structuring of proofs. The task layer consists of a datastructure to reference subgoals, together with a set of inference rules defined over this datastructure. In addition the layer provides a *proof datastructure* which represents the proof history.

The task level serves as a common interface between various proof construction components (e.g. an interactive user or automated proof procedures such as the proof planner Multi [14] of the Ωmega mathematical assistant system [17]) and the CORE system. This common interface makes it possible to interleave automated and interactive proof planning. For example, currently inactive subproblems can be tackled by an automated proof planner, while the planning methods are also available to the interactive user who tries to close an active subgoal [4]

Here we focus on the user interaction aspect of the task layer. We show that user-interaction at the task layer is related to the Box-Line approach or the focus windows of [15] but also provides a uniform mechanism for the application of assertions that abstracts from the logical details like the syntactical form of an assertion.

The paper is structured as follows. First, we briefly introduce the CORE system before we describe the task layer in Sec. 2. Sec. 4 shows how the task layer is used in practice.

2 The CORE System

The main characteristic of the CORE system is the *contextual rewriting* technique in which proofs are constructed. This means that the system is able to determine the logical *context* of a subformula F inside a goal formula $G[F]$. The logical context gives then rise to a number of *replacement rules* which can be used to rewrite the formula F to some formula F', yielding the new goal $G[F']$. By making use of the contextual rewriting technique the system supports a natural way of reasoning in which formula decomposition is dealt with implicitly and which operationalizes Huang's [9] reasoning on the assertion level. We illustrate this at hand of the following theorem (see also Fig. 1):

$$(P \Rightarrow Q) \wedge \underline{(Q \Rightarrow R)} \Rightarrow (P \Rightarrow R) \tag{1}$$

[4] This is easy in case of independent subgoals, i.e. subgoals that do not share variables. The problem of how to deal with subgoal that share variables is still an open research problem.

In CORE we can use the implication $Q \Rightarrow R$ (underlined in (1)) in the context of the rightmost R to replace R by Q. This is realized with the help of the replacement rule $R \rightarrow< Q >$ which is generated from the respective implication. Application of this replacement rule to R thus yields

$$\underline{(P \Rightarrow Q)} \wedge (Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \tag{2}$$

In a similar manner, it is possible to rewrite the newly obtained Q to P by applying the replacement rule $Q \rightarrow< P >$ which is justified by the implication $(P \Rightarrow Q)$ in the context of Q. Thus, we have

$$(P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow (P \Rightarrow P) \tag{3}$$

which CORE simplifies to True.

This small example illustrates COREs key characteristics: Reasoning at the assertion level is made possible through the generation of replacement rules from the assertions in the context of a subformula. Furthermore, the proof problem is always represented and maintained in its entirety instead of being decomposed into smaller pieces as in sequent- and natural deduction style calculi.

2.1 *Technical Description*

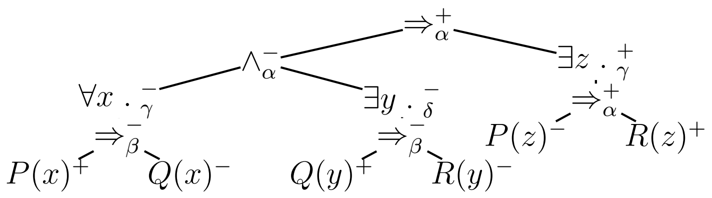
The key technique underlying the CORE-calculus is to view formulas as trees and to annotate each subtree with proof theoretic information. The proof theoretic information are polarities and uniform types introduced by Smullyan [19]: intuitively the polarity of a formula is positive, if the formula is a goal, i.e. occurs in the succedent of a sequent in some sequent-style calculus, and otherwise negative. As a result we obtain *signed formulas*. Types can be assigned to signed formulas, α for disjunctive formulas, β for conjunctive formulas, γ for universally quantified formulas and δ for existentially quantified formulas with an *Eigenvariable* condition. For instance, indicating polarities in superscripts and uniform types as subscripts, the positively signed formula

$$(\forall x\ .\ P(x) \Rightarrow Q(x) \wedge \exists y\ .\ Q(y) \Rightarrow R(y)) \Rightarrow \exists z\ .\ P(z) \Rightarrow R(z)$$

is canonically annotated as follows:

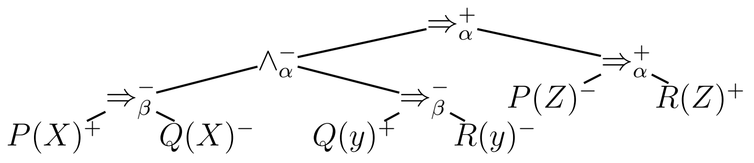
$$\begin{aligned}&(((\forall x\ .\ (P(x)^{+} \Rightarrow Q(x)^{-})^{-}_{\beta})^{-}_{\gamma} \wedge (\exists y\ .\ (Q(y)^{+} \Rightarrow R(y)^{-})^{-}_{\beta})^{-}_{\delta})^{-}_{\alpha} \\ &\Rightarrow (\exists z\ .\ (P(z)^{-} \Rightarrow R(z)^{+})^{+}_{\alpha})^{+}_{\gamma})^{+}_{\alpha}\end{aligned} \tag{4}$$

Viewing that formula as a tree we obtain:


$\Rightarrow^{+}_{\alpha}$
$\wedge^{-}_{\alpha}$
$\exists z\ .\ ^{+}_{\gamma}$
$\forall x\ .\ ^{-}_{\gamma}$
$\exists y\ .\ ^{-}_{\delta}$
$\Rightarrow^{+}_{\alpha}$
$\Rightarrow^{-}_{\beta}$
$\Rightarrow^{-}_{\beta}$
$P(z)^{-}$
$R(z)^{+}$
$P(x)^{+}$
$Q(x)^{-}$
$Q(y)^{+}$
$R(y)^{-}$

We denote this representation as an *indexed formula tree (*Ift*)* and observe that it represents a proof state in matrix calculi [21]. Thus, we can reuse the rules from these calculi to construct a matrix proof. In particular, it provides us with an efficient representation of variable dependencies.

In order to enable the assertion level reasoning style, we first derive a *free variable* representation from such an (initial) Ift which we denote by a *free variable indexed formula tree (*Fvift*)* to obtain:



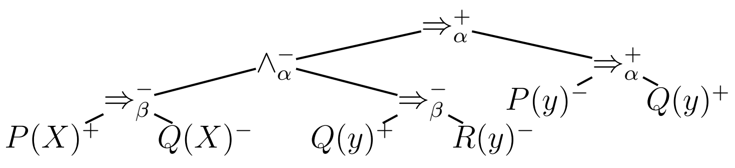
Thereby we introduce variables in capital letters if they are bound at γ-type positions in the Ift, and lower-case letters for those from δ-type positions. The assertion-style reasoning is enabled by exploiting the preserved polarities and uniform types as follows:

- First, the uniform types allow to statically determine the *logical context* for any subformula, i.e. those formulas that can be applied on that subformula. The sufficient criterion for this is that the minimal subtree containing both subformulas is of uniform type α, i.e. they are *α-related.* For instance, the signed subformulas $(P(Z)^- \Rightarrow R(Z)^+)^+_\alpha$ and $R(y)^-$ are α-related by the top-level connective $\Rightarrow^+_\alpha$.
- For each subformula in the logical context all rules operationalizing the possible applications of that formula can be derived by (1) fixing the left-hand side of the rule and (2) determining the subgoals by collecting all β-related subformulas. For instance, fixing $R(y)^-$ as the left-hand side of a rule, the β-related formulas form the singleton list $\langle Q(y)^+\rangle$, which we denote by

$$R(y)^- \longrightarrow \langle Q(y)^+\rangle$$

Note that we agree to denote an occurrence which has at least one β-related subtree as a *dependent occurrence.*

The above so-called *replacement rule* encodes the fact that we can replace the occurrence of a resolution partner for $R(y)^-$ by $Q(y)^+$. This allows for instance to apply that rule to the occurrence $R(Z)^+$ by applying the substitution $\{y/Z\}$ to rewrite the whole Fvift to



Note that we use the original Ift in order to check the admissibility of the substitution using the matrix-calculi techniques developed for this purpose.

Formally, this resolution style concept of a replacement rule is defined as follows[5]:

Definition 2.1 (Admissible Resolution Replacement Rules) Let R_0, R be nodes in some FVIFT. Then $R_0 \longrightarrow \langle R_1, \ldots, R_n \rangle$ is an *admissible resolution replacement rule* for R, if, and only if

(i) R_0 and R have opposite polarities and are α-related by a node c,

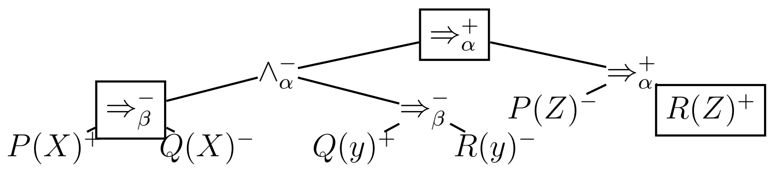
(ii) and $(R_1, \ldots, R_n)$ are the subtrees that are below c and β-related to R_0.$\Box$

In short, the CORE-calculus relies on proof states consisting of an IFT representing quantifier and substitution dependencies and a FVIFT, which is a kind of working copy that is actively manipulated by replacement rule applications. The calculus consist of 12 rules, including a cut rule, that transform a proof state into exactly one derived proof state. A proof state is proved if the FVIFT is a propositionally trivially valid formula, such as, for instance, True^+. The calculus is sound and complete for a variety of logics.

The framework provided by CORE so far is that the complete status of a proof is always represented as a single formula, which was one of the design goals. The system is augmented by the possibility to focus the reasoning process on arbitrary subformulas, without actually enforcing the decomposition of the formula, i.e. the FVIFT. To this end we add the possibility to introduce *windows* on arbitrary subformulas, which are explicit representations of the focus. For instance in the example above, without windows the complete formula is visible, i.e. the focus of the reasoning process is on the entire formula

$$(((P(X)^+ \Rightarrow Q(X)^-)^-_\beta \wedge (Q(y)^+ \Rightarrow R(y)^-)^-_\beta)^-_\alpha \Rightarrow (P(Z)^- \Rightarrow R(Z)^+)^+_\alpha)^+_\alpha$$

To support focusing the reasoning process, we annotate nodes of the FVIFT by windows, as for instance in



The *content* of a window is the (signed) subformula contained in the subtree rooted at the node denoted by the window. For instance, the content of the window $\boxed{R(Z)^+}$ is $R(Z)^+$ and the content of $\boxed{\Rightarrow^-_\beta}$ is $(P(X)^+ \Rightarrow Q(X)^-)^-_\beta$. Note that we allow windows to occur below other windows. Windows below which there are no more windows are so-called *active windows*. The intuition of focusing is that the content of active windows are those objects that can be manipulated by rule applications. To this end the pure CORE-calculus is

[5] The general concept of replacement rules encompasses the treatment of primitive equality and equivalences giving rise to so-called *rewriting replacement rules*. However, for the purpose of this paper the resolution style replacement rules are sufficient. For more details we refer the interested reader to [2].

extended to Fvift with windows to obtain a reasoning mechanism similar to window inference [16].

2.2 Interactive Theorem Proving with CORE

Within CORE the user currently works with the window inference mechanism. This means that when he invokes CORE in interactive mode on a goal G with axioms $Ax_1, \ldots, Ax_n$ it assembles an Ift for the formula $(Ax_1 \wedge \ldots \wedge Ax_n \Rightarrow^\alpha G)^+$ and creates a Fvift for it together with an initial window on G which is presented to the user. The content of this window can then be altered by applying the window versions of COREs calculus rules. Typically this will be an application of a replacement rule. Proof search in CORE is therefore characterized by two major kinds of choices:

(i) *Focus choice*: The selection of a subformula in the active window on which the user wants to focus the proof search.

(ii) *Rule choice*: The selection of a replacement rule.

Because of its strength in the application of assertions CORE is well suited for interactive proof construction. However, optimal support for focus and rule choice is still challenging. One challenge is related to rule choice since the context of a formula is currently available only as a usually long and unstructured list of replacement rules. To make things even worse, there are already dozens of replacement rules even for rather trivial problems. In particular, the number of replacement rules that can be generated from a subtree of a Fvift is exponential in the number of nodes in that tree. Although this is an effective sign of the flexibility in proof construction provided by CORE this flexibility must be controlled during automatic and semi-automatic proof construction. A further difficulty for automatic proof search is the choice of the focus. Because the focus of a proof can be changed arbitrarily it is hard to search for a proof in a systematic and goal directed way. In the following section we describe the task layer which supports the user in the structuring of the proof.

3 Tasks – Organizing Proof Search

Fvifts as introduced in Sec. 2 represent all conjunctive and disjunctive subgoals of a proof state simultaneously. However, while individual subgoals can be highlighted with the window inference mechanism the system does not support the user in systematically splitting the proof into smaller subgoals. Furthermore, a Fvift can grow significantly during a proof which makes the presentation of a proof state difficult.

It is therefore a challenge to present a proof state to the user as a set of subgoals (*tasks*) while maintaining the advantages that arise from the fact that a proof state is represented as a single formula. This is what is addressed by our task layer. At this layer, we use the window structure to reference parallel subgoals within an Fvift. These subgoals (tasks) structure the proof at the task layer where additional rules are provided to manipulate these subgoals. Reasoning steps at the task level are mapped into reasoning steps in the CORE system which automatically guarantees soundness.

To describe the task layer we proceed as follows. First, we give a formal definition of the task data structure. Then, we provide a set of implemented task manipulation rules. The inferences realized by these rules comprise simple decompositions of compound formulas, compound steps such as applications of assertions, and human-oriented steps such as lemma introduction.

3.1 Tasks

Intuitively, a task denotes a subgoal together with all the formulas (assertions) that can be used to derive this subgoal (all formulas that are α-related to the subgoal). Accordingly, a task is defined as a list of windows that all occur in the same context. However, before we make this intuition formal, we transfer the notion of dependent occurrences to windows.

Definition 3.1 (Conditional Window) Let w be a window on a subformula F in a Fvift R. We say that the window w is *conditional in* R iff the occurrence F is dependent in R. Otherwise we call w *unconditional in* R.

Definition 3.2 (Task) Let R be the Fvift of the current proof state. A *task* T is a set of windows $T = w_1, \ldots, w_n \rhd g$ for R with exactly one *goal window* g and *support windows* $w_1, \ldots, w_n$. Additionally we require that the following holds: if R' is the smallest subtree in R that contains all windows in T:

(i) the subtrees denoted by the $w_1, \ldots, w_n$ are α-related to each other,

(ii) all support windows of T are unconditional in R'.

We distinguish between goals and support windows to explicitly represent the focus of attention within a task. However, technically there is no difference between the support windows and the goal window. In particular, since in CORE the information on the polarity of each formula is explicitly held, the windows of a task can be freely exchanged. Thus, task manipulations (see the *Shift* rule in Sec. 3.2) can exchange the goal window of a task. This is a notable difference to the sequent calculus, in which formulas occurring left and right of $\vdash$ cannot be freely exchanged and have to be treated differently. In particular, a task $\Sigma, a^p \rhd a^q$ does not necessarily correspond to an initial sequent in the sequent calculus because the a's might have the same polarity (i.e. $p = q$).

The decision to forbid conditional support windows in a task was also driven by the interactive reasoning we envision. The content of support windows will be presented to the user as directly available knowledge that can be used to derive the formula in the goal window of the task. If the content of the support windows would be β-related to subtrees that lie outside the respective window, then these trees would automatically become conditions for any replacement rule that is generated from this window (compare to Def. 2.1). That is, the β-related subtrees would represent implicit "knowledge", which would be introduced in form of new proof obligations. We assume this to be less suited for interactive proof construction as it would result in subgoals whose origins are not directly obvious for a user.

Because tasks are basically representations of subgoals, we next define when a task is closed.

Definition 3.3 (Closed task) A task $\Sigma \rhd G$ is *closed* iff there exists a $w \in \Sigma \cup \{G\}$ such that w denotes a proved subtree; that is, w is either True^+ or False^-.

The initial problem to derive a goal G from axioms $Ax_1, \ldots, Ax_n$ is represented in the system as a FVIFT for the signed formula $(Ax_1 \wedge \ldots \wedge Ax_n \Rightarrow^\alpha G)^+$. The corresponding initial task consists of a goal window for G and support windows for each of the axioms:

Definition 3.4 (Initial Task) Let G be a formula and $Ax_1, \ldots, Ax_n$ formulas that represent axioms from which G should be derived. Let further R be an FVIFT for $(Ax_1 \wedge \ldots \wedge Ax_n \Rightarrow^\alpha G)^+$, w_i a window on Ax_i and g a window on G, then $w_1, \ldots, w_n \rhd g$ is the *initial task* for G.

Henceforth, when presenting tasks, we will not distinguish between a window in the task and the formula it contains. For instance, instead of $w_1, \ldots, w_n \rhd g$ in the definition above we will write $Ax_1, \ldots, Ax_n \rhd G$. [6]

3.2 Task Manipulation Rules

Tasks describe goals that have to be achieved during a proof process. The current tasks are stored in a so-called *agenda*. The proof process starts with an agenda that contains only the initial task. We can refine a task on the agenda to simpler tasks by the application of task manipulation rules. The proof process terminates, when all tasks in the agenda are closed.

Task manipulation rules may vary from low-level, basic rules that perform simple logic manipulations, to complex rules and speculative rules. In the following, we shall give examples of three different kinds of task manipulation rules and their realization in CORE: (1) Simple rules to split tasks for a conjunctive goal into subtasks, or to decompose disjunctive goals (Sec. 3.2.1). (2) Complex rules to apply assertions via the replacement rules and rules, which provide a functionality similar to the Proof by Pointing approach (Sec. 3.2.2). (3) Speculative rules, such as lemma introduction (*Cut*), which can be used to model human reasoning steps more closely (Sec. 3.2.3). In fact, these rules are motivated by the data that Benzmüller et al. [4] have collected to examine how undergraduate students carry out proofs in the domain of naive set

[6] Nevertheless, we can encounter tasks of the form $\Sigma, A^p, A^p \rhd G$. Although both A^p are syntactically equal formulas, they are different entities since they occur in different windows.

theory. Although these data have so far only been analyzed with respect to linguistic phenomena, a preliminary analysis of emerging proof patterns shows that students frequently apply lemmata that are not explicitly represented in the context but can be derived in a few steps from the available assumptions.

3.2.1 Decomposition Rules

Fig. 3.2.1 shows the rules for the decomposition of tasks with a compound goal formula. These rules resemble the usual rules of the ND and SK calculus. However, by making use of the α- and β-annotation of formulas, we can represent the rules in a compact manner. To distinguish between SK rules and the task manipulation rules, we give the rules in a top-down manner. That is, the rules read as follows: Tasks below the line replace the task above the line on the agenda.

$$\frac{\Sigma \rhd \alpha(A^{p_A}, B^{p_B})^p}{\Sigma, B^{p_B} \rhd A^{p_A}}\,\alpha_L \qquad \frac{\Sigma \rhd \alpha(A^{p_A}, B^{p_B})^p}{\Sigma, A^{p_A} \rhd B^{p_B}}\,\alpha_R \qquad \frac{\Sigma \rhd \alpha(A^{-p})^p}{\Sigma \rhd A^{-p}}\,\alpha_\neg$$

$$\frac{\Sigma \rhd \beta(A^{p_A}, B^{p_B})^p}{\Sigma \rhd A^{p_A} \qquad \Sigma \rhd B^{p_B}}\,\beta \qquad \frac{\Sigma \rhd \zeta(A^o, B^o)^+}{\Sigma, B^- \rhd A^+ \qquad \Sigma, A^- \rhd B^+}\,\Leftrightarrow$$

Fig. 2. The basic decomposition rules for tasks.

We distinguish between tasks with a conjunctive goal window (type β) and tasks with a disjunctive goal window (type α).

α-Decomposition

Disjunctive goal windows can be decomposed with the rules α_R, α_L and $\alpha_\neg$, where the rule α_R corresponds to the $\Rightarrow_E$ rule in the ND calculus. It is crucial that (at least for the α-rules) decomposition at the task level is *simulated* in CORE by a change of the window structure of the corresponding Fvift. More precisely, the decomposition of a formula $\alpha(A^{p_A}, B^{p_B})$ is simulated in CORE by the opening of new subwindows on the A^{p_A} and B^{p_B}. In the case of the α_R rule the new window on A^{p_A} is added to the support windows of the corresponding task, while the B^{p_B} replaces the goal window. The other α-decomposition rules are realized accordingly. This has the advantage that decomposition steps can easily be undone (see the *FocusClose* rule).

β-Decomposition

The β-decomposition rule reduces a task with a conjunctive goal formula $\beta(A^{p_A}, B^{p_B})$ to new subtasks for the goals A^{p_A} and B^{p_B} respectively. However, mapping the application of the β-decomposition rule into the CORE system requires a little more effort than was the case for α-decomposition. The reason is that according to Definition 3.2 there must be no conditional support windows in a task. Hence, if the β-rule would be mapped into the CORE-system

$$\frac{\Sigma \rhd G[A^{p_A}]}{\Sigma_1 \rhd A^{p_A} \;\; \Sigma_2 \rhd G_1, \ldots, \Sigma_n \rhd G_n}\; \mathit{Focus}$$

$$\frac{\Sigma \rhd G \quad G' = \mathit{Parent}(G)}{\Sigma \rhd G' \quad \mathit{Subwindows}(G') = \emptyset}\; \mathit{FocusClose}$$

$$\frac{\Sigma, F \rhd G}{\Sigma, G \rhd F}\; \mathit{Shift} \qquad \frac{\Sigma \rhd \diamond}{\emptyset}\; \mathit{Close\ where}\ \diamond \in \{\mathsf{True}^+, \mathsf{False}^-\}$$

$$\frac{\Sigma \rhd i}{\Sigma, i \rhd v_1 \ldots \Sigma, i \rhd v_n}\; \mathit{Apply}(i \rightarrow < v_1, \ldots, v_n >)$$

Fig. 3. Assertion application and focusing rules.

similar to the α-decomposition rule, the goal windows of the tasks introduced by this rule would be conditional and consequently, it would not be permitted to apply the *Shift*-rule to these tasks to change the goal window. To avoid these restrictions, the formulas A^{p_A} and B^{p_B} in $\beta(A^{p_A}, B^{p_B})$ are made unconditional when the β-rule is applied.

This can be done by splitting up the goal formula $\beta(G_1, G_2)$ while retaining the context φ around it. We can achieve this by applying a rule of the form $\varphi(\beta(A, B)) \rightarrow \beta(\varphi(A), \varphi(B))$ which is described in [18] and [2]. Autexier [2] shows that this *Schütte-rule* is admissible in the CORE calculus.

Positive Equivalence Expansion

Finally, we introduce a rule $\Leftrightarrow$ to expand the definition of equivalences $A \Leftrightarrow$ into $A \Rightarrow B$ and $B \Rightarrow A$ for tasks where the goal window contains a positive equivalence formula.

3.2.2 Assertion Application and Proof by Pointing

As opposed to the rules introduced in the preceding section, we shall next introduce manipulation rules that extend the inferences at the task layer beyond the usual scope of the ND and the SK calculus. Figure 3 shows the five rules that – among others – makes CORE's flexibility available for the applications of assertions and also enable the Proof by Pointing approach at the task layer.

Compound Decomposition Steps with *Focus*

The rule *Focus* combines the decomposition rules from Sec. 3.2.1. By applying *Focus* the user can directly focus on a particular subformula A^p inside a goal window G. The uniform types of the nodes in a Fvift that occur on a path between the selected subformula A^p and the root $G[A^p]$ of the goal window uniquely define a sequence of decomposition steps that need to be applied in order to obtain the chosen formula as a goal window in a single step. The *Focus* rule carries out these decomposition steps and keeps track of the subgoals $\Sigma_1 \rhd G_1, \ldots, \Sigma_n \rhd G_n$ that are generated whenever a β-decompositions is carried out. Essentially, this is the Proof by Pointing idea, which is integrated by the *Focus* rule into the task layer. We call the *Focus* rule a *macro*-rule because it applies a sequence of basic decomposition rules.

FocusClose

The *FocusClose*-rule is the inverse rule to the decompositions described by the α, β and the *Focus* rule. It closes the window on the goal formula and simultaneously all other foci below the parent of G. Therefore, decomposition steps can be undone one after another.

Applying Assertions with *Apply*

To apply replacement rules at the task layer we introduce the *Apply*-rule. The idea is that in order to apply an assertion (i.e. a formula in a support window) the user merely needs to select an assertion by clicking at it at the user interface. The system then creates a list of all replacement rules that are justified by this assertion. In case there is only a unique applicable rule it is applied automatically via the *Apply*-rule. In the more likely case that there is more then one rule applicable the user has to select the replacement rule that describes the appropriate application direction of the assertion. At this point heuristics will be used to narrow down the choice for the user.

Hübner[10] describes how replacement rule selection can be supported by the agent-based suggestion mechanism ΩANTS which was until now only used in conjunction with the ΩMEGA [17] theorem proving system. [7]

Closing tasks with *Close*

Tasks are removed from the agenda in case they are closed. This is done by the *Close* rule, which deletes any task $\Sigma \rhd \diamond$ such that $\diamond \in \{\mathsf{True}^+, \mathsf{False}^-\}$ from the agenda.

Shifting the goal of a task with *Shift*

The *Shift*-rule changes the goal window of a task. This is particularly important because the other manipulation rules work only on the goal window of a task. For instance, consider that a window with formula A^p and a window with $(A \Leftrightarrow B)^-$ are amongst the support windows for a goal G; that is, $\Sigma, (A \Leftrightarrow B)^-, A^p \rhd G$.

Note that we realized β-decomposition with the *Schütte-rule*. Therefore, we can ensure that goal windows are always unconditional, which is a prerequisite for the above definition of the *Shift*-rule.

[7] Note that in the way the *Apply* rule is defined here it can only be applied to the topmost occurrence i in a goal window and not to subformulas of i. However, this is no problem since we can always focus on a subformula with the rule *Focus* first.

$$\frac{\Sigma \rhd G^p}{\Sigma \rhd H \quad \Sigma, G^{-p} \rhd H^p}\ LemBW \qquad \frac{\Sigma \rhd G}{\Sigma, A^- \rhd G \quad \Sigma \rhd A^+}\ LemFW$$

$$\frac{\Sigma \rhd G}{\Sigma, A^- \rhd G \quad \Sigma, A^+ \rhd G}\ Case$$

Fig. 4. Assertion application and focusing rules.

3.2.3 Speculative Proof-Steps

The introduction of new lemmata or case splits are speculative proof steps that are often crucial to accomplish a proof. It is not surprising that such steps play also an important role in both automated – and interactive theorem proving. The application of speculative steps is difficult to control in automated theorem proving they open a Pandora's box (the possible lemmata and case splits are a priori not restricted). Bundy and Ireland [12] as well as Meier [14] describe the exploitation of failures in automated proof processes in order to guide the introduction of lemmata and case splits.

To be able to model this reasoning at the task-level we augment the inference rules by the rules *LemFW*, *LemBW*, and *Case* shown in Fig. 4.

The *LemFW* rule corresponds to the forward application of a lemma and replaces a goal formula G by another formula H. This then leads to the generation of an additional task which encodes the obligation to show that the proof step described by the replacement was actually valid. The related *LemBW* rule allows to insert a new lemma A into the supports of a task which can then be applied via the *Apply* rule.

The *Case* rule reduces a task with goal G to two new tasks with goal G. One of the new tasks has a new support window A^- whereas the other new task has the support A^+.

3.2.4 Tasks and Proof Presentation

So far, we have described the inferences that can be carried out at the task layer. These inferences enable the structured application and exploitation of CORE's functionalities and flexibility. Another motivation for the invention of the task layer on top of CORE was proof presentation. With the proof structure provided by the tasks, subgoals can be displayed together with the available assertions in a suggestive manner. In fact, using the structure provided by tasks, we can display subgoals in the Box-Line representation, which is well known from introductions to the ND calculus (e.g., see [11]) and which has also been suggested as an appropriate visualization for interactive proofs by Piroi and Buchberger [15].

In the Box-Line representation, a goal formula is presented below the available assertions, which are displayed in a Box (see Sec. 4). We realize this presentation style in the user interface by presenting the support windows of a task on top of the goal window. This facilitates the application of assertions, which can be applied in a uniform way by making use of the *Apply* rule. The Box-Line presentation enables the user to apply an assertion by clicking on the corresponding support window. The replacement rules corresponding to the appropriate application direction of the assertion can then be applied with the help of the *Apply* rule.

4 Tasks - A Worked Example

In this section, we prove the theorem from Fig. 1 at the task layer. This permits us to illustrate the communication between user and the system as well as to compare our approach to the one of Bornat and Bertot[8]. We use a syntactically different, but logically equivalent formula to point out the advantages of our approach.
The initial task of our proof problem is:

$$\left[\ \right] \qquad (1)$$

$$((P \Rightarrow Q) \wedge (\neg Q \vee R) \Rightarrow (P \Rightarrow R))^{+}$$

The application of the *focus* to extract the rightmost R results in the following situation:

$$\left[\begin{array}{c} (P \Rightarrow Q)^{-} \\ (\neg Q \vee R)^{-} \\ P^{-} \end{array}\right] \qquad (2)$$

$$R^{+}$$

Then, we can reason backwards by applying the assertion $(\neg Q \vee R)^{-}$ with the *Apply* rule to the goal window. Internally, *Apply* carries out this proof step by the application of the replacement rule $(R^{+} \rightarrow < Q^{+} >)$ to the goal. Similarly, we could reason forward by applying the assertion $(P \Rightarrow Q)^{-}$ (i.e., the replacement rule $(P^{-} \rightarrow < R^{-} >)$) to the support P^{-}. The application of

[8] For a more detailed example we refer to [10].

the backward step results in the task:

$$(3) \qquad \left[\begin{array}{c} (P \Rightarrow Q)^- \\ (\neg Q \vee R)^- \\ P^- \\ R^+ \end{array}\right]$$

$$Q^+$$

Similarly, we can continue to reasoning backwards by applying the assertion $(P \Rightarrow Q)^-$ to the goal window of the task or reason forward by applying the same assertion to the support P^-. This time we decide in favor of the forward step, which we realize on the task-level by the aforementioned application of the *Apply* rule with the replacement rule $(P^- \rightarrow < R^- >)$. The resulting task is

$$(4) \qquad \left[\begin{array}{c} (P \Rightarrow Q)^- \\ (\neg Q \vee R)^- \\ P^- \\ R^+ \\ Q^- \end{array}\right]$$

$$Q^+$$

which can be closed by the application of the *Apply* rule with the replacement rule $(Q^+ \rightarrow < \mathsf{True}^+ >)$ to the goal.

Discussion

By building the task layer on top of the CORE system both concepts benefit mutually. Tasks allow to structure and display a CORE proof state in a user-oriented way (Box-and-Lines) while the underlying CORE systems allows for flexible reasoning at the task-level. This flexibility manifests itself in the integration of the Proof by Pointing approach with a uniform mechanism for the application of assertions. Both concepts can be improved at the task layer. Forward and backward application of assertions can be naturally carried out at the task layer and do not have to be mapped into a sequence of cut-steps.

Furthermore, at the task layer, assertions can be applied independently from their syntactic structure. By looking at the sample proof it can be seen that the interactions necessary for the application of $P \Rightarrow Q$ and $\neg Q \vee R$ are the same. In the pure SK and ND calculus the corresponding steps would be different for both assertions.

The user interaction as well as the presentation of tasks at the user interface we presented here is similar to the the *TkWinHOL*-system ([13]). However, the approaches differ in the underlying reasoning system. While the *TkWinHOL*-system is built on top of Grundy's [8] window inference system. In this system rewriting steps on the content of the active window result in a local lemma that needs to be tackled afterwards. This also holds for the window inference approach of Robinson and Staples [16]. In the CORE system underlying the task layer introduced here transformations that are realized by application of replacement rules are guaranteed to be sound and hence do not introduce new subgoals.

5 Conclusion and Future Work

We have introduced a task layer for the CORE system and described the implications of this layer for interactive proof construction. We were able to point out how the task layer helps to structure and display CORE proofs. Furthermore, by exploiting the contextual reasoning paradigm of the CORE system we were able to combine and extend existing concepts like the Proof by Pointing approach in a single interaction layer.

In this paper, we gave a bottom-up description of the task layer serving as interface to CORE. Such a bottom-up approach to enrich the logic layer by more abstract level reasoning tools is the standard approach in many proof assistants to support abstract level proof development. A drawback of the bottom-up approach, however, is that it usually causes an unnecessarily strong dependence of the abstract layer upon the logic layer. Therefore, we are actually developing the task layer as independent as possible from the underlying logic layer, such that, in the ideal case, the logic layer becomes exchangeable. However, we do not propose proof assistants lacking a sound logical basis. Instead, our aim is to distinguish better between abstract level reasoning and expansion into verifiable proofs at the logic layer.

At the task layer we are interested to allow for mathematics-oriented, intuitive proof development steps. Thus, the task layer will be enriched by further task manipulation operators that can be domain-specific and have to be acquired and designed in a knowledge engineering process as known from tactical theorem proving and proof planning. Alternatively, the human may embody the role of an operator by declaratively describing the refinement of a task into successor tasks; such *oracle steps* adapt the idea of interactive island proofs as introduced in [17]. Thus, the concretely available task manipulation operators may vary from rather "safe" ones (those who directly guarantee logical soundness at the logic layer, e.g., traditional tactics constructed on top of the logic layer) to highly "unsafe" ones (those with non-sharp application constraints causing frequent failing logic layer expansions, e.g., human constructed oracle steps). However, in order to classify the task-level proof as sound, *all* proof steps at the task-level have to be successfully expandable and verifiable at the logic layer.

How "expensive" the expansion of a task manipulation step to the logic layer is depends on the concrete steps used at the task layer as well as on the chosen logic layer. In the Ωmega proof assistant [17] the logic layer consists of a higher-order natural deduction calculus and the distance from abstract proof plans via expansion to this particular basic logic is in many case studies very huge (e.g., see [17]). We are currently beginning with a re-implementation of Ωmega with CORE as the logic layer and the task layer as separated layer for abstract proof development. Due to this move from higher-order natural deduction calculus to CORE we are able to drastically reduce the expansion distance from the abstract layer to the basic logic layer in ΩMEGA^{CORE}.

The task manipulation rules we present in this paper are all "safe", i.e., they directly guarantee logical soundness in CORE. [9] They are very "close" to the chosen logic level and serve as the interface rules to work directly on top of CORE. We tested them at hand of relatively simple logical problems such as the one described in Fig. 1. The ΩMEGA^{CORE} system is envisaged as platform for a mathematical assistant system as well as for a tutoring tool for mathematics. As opposed to our current applications and tests, these applications require the specification of domain-dependent and "unsafe" operators at the task layer as well as their expansion to CORE.

In this paper, we gave a bottom-up description of the task layer serving as interface layer to the user on top of CORE. Such a bottom-up approach to enrich the logic layer by more abstract level reasoning tools is the standard approach in many proof assistants to enable (ideally) abstract level proof development. A drawback of the bottom-up approach, however, is that it usually causes an unnecessarily strong dependence of the abstract layer upon the logic layer. Therefore, we are actually developing the task layer in a top-down approach to keep it as independent as possible from the underlying logic layer, such that, in the ideal case, the logic layer becomes exchangeable. However, we do not propose proof assistants lacking a sound logical basis. Instead, our aim is to distinguish better between abstract level reasoning and expansion into verifiable proofs at the logic layer.

At the task layer we are interested to allow for mathematics-oriented, intuitive proof development steps. Thus, concrete task manipulation operators can be domain-specific and have to be acquired and designed in a knowledge engineering process as known from tactical theorem proving and proof planning. Alternatively, the human may embody the role of an operator by declaratively describing the refinement of a task into successor tasks; such *oracle steps* adapt the idea of interactive island proofs as introduced in [17]. Thus, the concretely available task manipulation operators may vary from rather "safe" ones (those who directly guarantee logical soundness at the logic layer, e.g., traditional tactics constructed on top of the logic layer) to highly "unsafe" ones (those with non-sharp application constraints causing frequent

[9] Speculative steps such as case splits and lemmas may not result in proof objects at the task level. However, their expansion into CORE is straightforward.

failing logic layer expansions, e.g., human constructed oracle steps). However, in order to classify the task-level proof as sound, *all* proof steps at the task-level have to be successfully expandable and verifiable at the logic layer.

How "expensive" the expansion of a task manipulation step to the logic layer is depends on the concrete steps used at the task layer as well as on the chosen logic layer. In the ΩMEGA proof assistant [17] the logic layer consists of a higher-order natural deduction calculus and the distance from abstract proof plans via expansion to this particular basic logic is in many case studies very huge (e.g., see [17]). We are currently beginning with a re-implementation of ΩMEGA with CORE as the logic layer and the task layer as separated layer for abstract proof development. Due to this move from higher-order natural deduction calculus to CORE we are able to drastically reduce the expansion distance from the abstract layer to the basic logic layer in ΩMEGA^{CORE}.

The task manipulation rules we present in this paper are all "safe", i.e., they directly guarantee logical soundness in CORE. [10] They are very "close" to the chosen logic level and serve as the interface rules to work directly on top of CORE. We tested them at hand of relatively simple logical problems such as the one described in Fig. 1. The ΩMEGA^{CORE} system is envisaged as platform for a mathematical assistant system as well as for a tutoring tool for mathematics. As opposed to our current applications and tests, these applications will need the specification of domain-independent and "unsafe" operators at the task layer as well as their expansion to CORE.

References

[1] S. Autexier. A proof-planning framework with explicit abstractions based on indexed formulas. In Maria Paola Bonacina and Bernhard Gramlich, editors, *Electronic Notes in Theoretical Computer Science*, volume 58. Elsevier Science Publishers, 2001.

[2] S. Autexier. *Hierarchical Contextual Reasoning*. PhD thesis, University of the Saarland, 2003. to appear.

[3] C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Quoc Vo, and M. Wolska. Discourse phenomena in tutorial dialogs on mathematical proofs. In *In Proceedings of AI in Education (AIED 2003) Workshop on Tutorial Dialogue Systems: With a View Towards the Classroom*, Sydney, Australia, 2003.

[4] C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Quoc Vo, and M. Wolska. Tutorial dialogs on mathematical proofs. In *In Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, Acapulco, Mexico, 2003.

[10] Speculative steps such as case splits and lemmas may not result in proof objects at the task level. However, their expansion into CORE is straightforward.

[5] Y. Bertot, G. Kahn, and L. Thery. Proof by pointing. In *Theoretical Aspects of Computer Science (TACS)*, 1994.

[6] R. Bornat. Interactive Disproof: Rendering Tree Proofs in Box Form In D. Aspinal & C. Lüth, Proceedings of UITP03, Rome, Italy, September, 2003.

[7] G. Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39:572–595, 1935.

[8] Grundy, J., *Window inference in the hol system*, in: *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications* (1991), pp. 177–190.

[9] X. Huang. Reconstructing proofs at the assertion level. In A. Bundy, editor, *Proc. 12th Conference on Automated Deduction*, pages 738–752. Springer-Verlag, 1994.

[10] Hübner, M., *Interactive Theorem Proving with Indexed Formulas*, SEKI Technical Report SR-03-04, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany (2003).

[11] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge Univ. Press, 2000.

[12] A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

[13] Långbacka, T., R. Ruksenas and J. v. Wright, *Tkwinhol: A Tool for Window Interference in HOL*, , **971** (1995), pp. 245–260.

[14] A. Meier. *Proof-Planning with multiple strategies.* PhD thesis, University of the Saarland, 2003. forthcoming.

[15] 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 Computation*, number 2385 in LNAI, pages 337–341. Springer, 2002.

[16] P. J. Robinson and J. Staples. Formalizing a hierarchical structure of practical mathematical reasoning. *Journal of Logic Computation*, 3(1):47–61, 1993.

[17] J. Siekmann, C. Benzmüller, A. Fiedler, A Meier, I. Normann, and M. Pollet. Proof development in OMEGA: The irrationality of square root of 2. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, Kluwer Applied Logic series. Kluwer Academic Publishers, July 2003.

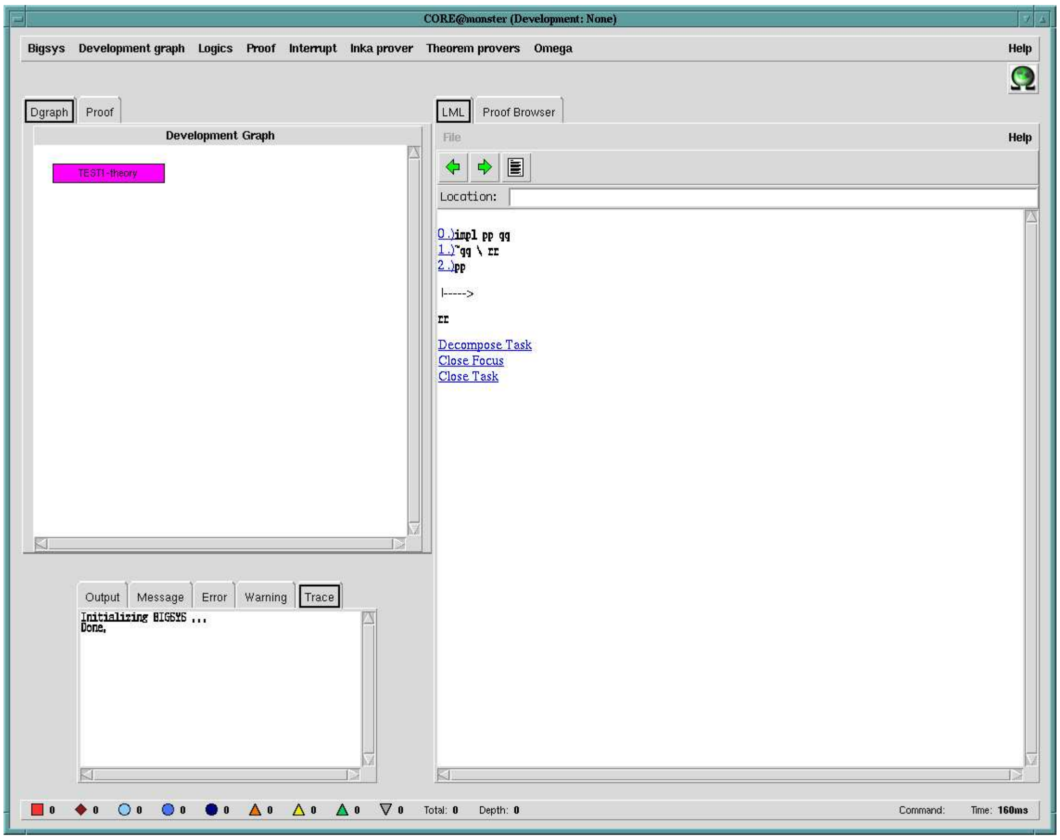
[18] K. Schütte. *Proof Theory.* Springer Verlag, 1977.

[19] R. R. Smullyan. *First Order Logic.* Springer, 1968.

[20] Q. B. Vo, C. Benzmüller, and S. Autexier. Assertion application in theorem proving and proof planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. (poster description).

[21] L. A. Wallen. *Automated Proof Search in Non-Classical Logics. Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* MIT Press, Cambridge, Massachusetts;London, England, 1990.

A Screenshots



```
0.)impl pp qq
1.)~qq \ rr
2.)pp

|---->

rr

Decompose Task
Close Focus
Close Task
```

```
Initializing BIGSYS ...
Done.
```

Fig. A.1. GUI of the CORE system. The right window displays the current task. Here we show the task corresponding to (2) on page 14.