Interactive Proof Construction at the Task Level

Malte Hübner, Christoph Benzmüller, Serge Autexier and Andreas Meier

Fachbereich Informatik, Universität des Saarlandes
D-66041 Saarbrücken, Germany
{huebner|chris|serge|ameier}@ags.uni-sb.de

Abstract. Interactive theorem proving systems for mathematics require user interfaces which can present proof states in a human understandable way. Often the underlying calculi of interactive theorem proving systems are problematic for comprehensible presentations since they are not optimally suited for practical, human oriented reasoning in mathematical domains. The recently developed CORE theorem proving framework [Aut03] is an improvement of traditional calculi and facilitates flexible reasoning at the assertion level. We make use of COREs reasoning power and develop a communication layer on top of it, called the *task layer*. For this layer we define a set of manipulation rules that are implemented via COREs calculus rules. We thereby obtain a human oriented interaction layer that improves and combines ideas underlying the window inference technique [RS93], the proof by pointing approach [BKT94], and the focus windows of [PB02].

1 Introduction

In interactive theorem proving (ITP) it is important that information about a proof state is presented in an intuitive way in order to help the user in the selection of a next proof step. Ideally, the knowledge (assertions) available to prove a focused goal (i.e. its logical *context*) is presented in a suggestive manner supporting a uniform handling of assertion selection and application. Consider for example a situation where a goal formula B has to be derived from some formulas $Ax_1, \ldots, Ax_n$ (we say that $Ax_1, \ldots, Ax_n$ are in the context of B). If we further assume that the formula $(\forall x.Q[x]) \Rightarrow (B \wedge C)$ is amongst the Ax_i then we could present this situation in the form shown in Figure 1(a). Using this representation the formulas in square brackets represent all the information that is available to infer the goal B. In the situation above it seems promising to apply the assertion $(\forall x.Q[x]) \Rightarrow (B \wedge C)$ to transform B into $\forall x.Q[x]$ as depicted in Figure 1(b).

In most current ITPs such a style of presentation and reasoning is difficult. The reason for this is that these ITPs are typically based on calculi that have not been developed for practical reasoning but for proof-theoretic considerations. These calculi often represent a proof state as sets of sequents or clauses and sometimes even require formulas to be in a certain normal form which makes a presentation like the above difficult. Furthermore, assertions can often not be

$$a)\left[\begin{array}{c} Ax_1 \\ \vdots \\ Q[x^\gamma] \Rightarrow (B \wedge C) \\ \cdot \\ Ax_n \\ A \end{array}\right] \qquad b)\left[\begin{array}{c} Ax_1 \\ \vdots \\ Q[x^\gamma] \Rightarrow (B \wedge C) \\ \cdot \\ Ax_n \\ A \end{array}\right]$$

$$B \qquad\qquad\qquad \forall x.Q[x]$$

Fig. 1. Intuitive presentation of proof tasks: goals appear together with the assertions available in the context.

applied directly to a subgoal and have to be explicitly decomposed through the application of calculus rules before they can be used in a proof. The need for explicit decomposition is one reason why natural deduction style calculi and sequent style calculi are far less intuitive and human oriented as we actually wish.

Autexier [Aut01,Aut03] recently developed the CORE theorem proving system as a means for the integration of multiple proof paradigms into one framework. The system supports a proof style that is based on *contextual rewriting* and it furthermore exploits ideas of the *window inference technique* [RS93]. The novelty of the system lies in the fact that it does not make use of a fixed set of calculus rules that are defined over the syntactic structure of formulas; rather, it allows the user to freely focus the reasoning process on subformulas of the overall goal formula while the system internally updates the logical context of these formulas while avoiding explicit decomposition. The context of a subformula is then made available to the user in form of transformation rules, so called *replacement rules*, which can be used to directly transform the subformula in the current focus. One of the advantages of this reasoning style is that it supports the application of assertions in a more intuitive way, while formula decomposition is treated implicitly. In fact, the stepwise unwrapping of subgoals and assertions is replaced by flexible focus relocations and replacement rule applications. However, although CORE is a potentially well suited basis for ITP it is not yet free of problems. A major drawback is that, until now, it is only possible to display the context of a formula as a usually long and unstructured list of replacement rules.

In this paper we develop a uniform communication layer — called *task structure* — on top of the CORE reasoning framework. This communication layer is intended to serve as the exclusive interface between CORE and the user and between CORE and automated proof procedures, such as the proof planner MULTI [Mei03] of the *Ω*MEGA mathematical assistant system [SBF+03]. Task structures allow to display the formulas in the context of a formula in the style used in Figure 1 and also provide a uniform framework for application of these formulas. The overall idea is that reasoning on the task structure reflects flexible reasoning with assertions while all logic level aspects (including decomposition) are realized via CORE and thus hidden from the user. We thereby gain a system that improves and combines the ideas of the window inference technique [RS93], the proof by pointing approach [BKT94], and the focus windows of [PB02]. By mapping the proof steps performed at the task level into sequences of CORE calculus rule applications we can furthermore guarantee the soundness of the steps taken at task level.

The remainder of the paper is outlined as follows. In Section 2 we introduce the CORE system before we describe the task structure in Section 3. Section 4 demonstrates the benefits of our approach at hand of a small example.

2 The CORE System

CORE supports flexible contextual rewriting: When focusing on a subformula F' of a goal F for manipulation the CORE system determines the logical context of F' and makes it available to the user as a set of replacement rules R. The replacement rules in R can be directly employed for manipulation of F'.

This supports a natural way of reasoning which treats formula decomposition implicitly and which resembles Huang's [Hua94] reasoning on the assertion level. We illustrate this at hand of the following theorem:

$$(M \wedge N) \wedge (M \wedge N \Rightarrow P) \Rightarrow P \tag{1}$$

In CORE we can use the implication $M \wedge N \Rightarrow P$ in the context of P to replace P by $M \wedge N$. This is realized with the help of the replacement rule $P \rightarrow< M \wedge N >$ which is generated from the respective implication. Application of this replacement rule to P thus yields

$$(M \wedge N) \wedge (M \wedge N \Rightarrow P) \Rightarrow M \wedge N \tag{2}$$

The newly introduced $M \wedge N$ can now be replaced by *true* through application of the replacement rule $(M \wedge N) \rightarrow< \mathit{true} >$ which we obtain from the subformula $(M \wedge N)$ on the left hand side of the implication. Thus, we have

$$(M \wedge N) \wedge (M \wedge N \Rightarrow P) \Rightarrow \mathit{true} \tag{3}$$

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 Proof Theoretic Annotations

CORE is based on the notions of *signed formulas* and *indexed formula trees* (IFTs) which go back to Smullyan [Smu68] and Wallen [Wal90]. We only briefly

α	α_1	α_2
$(A \wedge B)^-$	A^-	B^-
$(A \vee B)^+$	A^+	B^+
$(A \Rightarrow B)^+$	A^-	B^+
$(\neg A)^+$	A^-	
$(\neg A)^-$	A^+	

β	β_1	β_2
$(A \wedge B)^+$	A^+	B^+
$(A \vee B)^-$	A^-	B^-
$(A \Rightarrow B)^-$	A^+	B^-

Fig. 2. Uniform Notation (Propositional Types)

δ	δ_0
$(\forall x.F[x])^+$	$F[x/c]^+$
$(\exists x.F[x])^-$	$F[x/c]^-$

γ	γ_0
$(\forall x.F[x])^-$	$F[x/c]^-$
$(\exists x.F[x])^+$	$F[x/c]^+$

Fig. 3. Uniform Notation (Quantifiers); respective variable conditions are introduced and maintained by CORE for these cases

sketch them here and refer to [Aut03] for a more detailed introduction. CORE supports a variety of different logics, including higher-order and modal logics, however, for sake of simplicity, we only consider a classical first-order system in this paper.

Definition 1 *(Signed Formulas) A* signed formula *is a pair* (A, p)*, where* A *is a formula and* $p \in \{+, -, 0\}$ *the* polarity *of* A*. We usually write* A^p *instead of* (A, p)*.*

Signed formulas are assigned a uniform type to distinguish between disjunctive formulas (type α), conjunctive formulas (type β), universal formulas (type γ) and existential formulas (type δ). The tables in Figures 2-4 define the types of signed formulas and also determine how type information is propagated to the major subformulas of a signed formula.

In the remainder we denote by $\alpha(\alpha_1^{p_1}, \alpha_2^{p_2})^p$ a signed formula F of type α and polarity p. The $\alpha_i^{p_i}$ are the major subformulas of F with polarities p_i. Formulas of type β, γ and δ are denoted in a similar way. To indicate the type of a formula we frequently attach the type information to the topmost connective of a formula, e.g. $(A \wedge^\beta B)^+$. We furthermore abbreviate $\alpha(F_1, \alpha(F_2, \ldots, \alpha(F_{n-1}, F_n)))$ as $\alpha(F_1, \ldots, F_n)$ and use a respective notation for β.

Intuitively, the information encoded in annotated signed formulas describes the "behavior" of these formulas in sequent calculus derivations. The polarity $(+/-)$ of a signed formula is just another representation of the succedent/antecedent distinction used by Gentzen [Gen35] in the sequent calculus. Wallen [Wal90] interpretation is: Instead of distinguishing between formulas in the antecedent Γ and succedent Δ by using the sequent symbol $\vdash$ (i.e. $\Gamma \vdash \Delta$), he simply annotates all formulas that would occur in the antecedent Γ of a sequent (after application of the appropriate sequent calculus rules) with a negative polarity $(-)$, and formulas that would occur in the succedent of a sequent with a positive $(+)$ polarity. The uniform type of a formula describes whether the subformulas into

ϵ	ϵ_1	ϵ_2
$(A \Leftrightarrow B)^-$	A^0	B^0
$(s = t)^-$	s^0	t^0

ζ	ζ_1	ζ_2
$(A \Leftrightarrow B)^+$	A^0	B^0
$(s = t)^+$	s^0	t^0

Fig. 4. Uniform Notation (Equivalences and Equations); the major subformulas of equivalences and equations are of polarity 0 (the undefined polarity)

which the formula would be decomposed after application of the corresponding sequent calculus rule occur together in the same sequent (i.e. in the same branch of the proof) or will be parts of different sequents (i.e. different proof branches). In the former case the formula is assigned the uniform type α while in the latter case it has uniform type β.

Furthermore, the types δ and γ are assigned to formulas with a quantifier as topmost logical connective (i.e. formulas of the form $Qx.F[x]$ with $Q \in \{\forall, \exists\}$). Type δ is assigned to formulas which topmost quantifier binds a variable with an *Eigenvariable condition* and γ refers to freely instantiable variables. For more details on the respective variable conditions introduced and maintained by CORE for these cases we refer to [Aut03].

Negative equations and equivalences are of type ϵ and their constituents have undefined polarity. ζ and ϵ formulas are displayed in Figure 4.

Signed formulas F can be represented as trees where each node is labeled with a signed subformula of F. For each node n with label N in such a tree we require that the child nodes of n are labeled with the major subformulas of N.[1] This leads to the notion of an *indexed formula tree* (IFT). The IFT for formula (1) is given in Figure 5. For a formal definition of an IFT we refer to [Aut03] and [Wal90].

We say that two nodes of an IFT are β-related (respectively α-related) to each other if they have a common father (the first common predecessor) that is labeled with a signed formula of type β (respectively α).

We distinguish between dependent and independent nodes in an IFT.

Definition 2 *(Dependent Occurrence) Let a be a node in an* IFT *R. We say that a is* dependent *in tree R if there is a node b in R that is β–related to a. Otherwise we call a an* independent occurrence *in R.*

It it useful to introduce a partial ordering $\prec$ on the set of nodes of IFTs: $n_1 \prec n_2$ if and only if n_1 is an ancestor of n_2.

Indexed formula trees are used in CORE to represent the quantifier dependencies of a proof state in the following way: When we load a goal formula G together with axioms $Ax_1, \ldots, Ax_n$ from which we want to derive G the system creates an IFT for the signed formula $(Ax_1 \wedge \ldots \wedge Ax_n \Rightarrow^\alpha G)^+$. The system also creates a *free variable indexed formula tree* (FVIFT) for the same formula. FVIFT and IFT together represent a CORE proof state where proof search manipulates the FVIFT while the IFT is used to maintain the dependencies between quantifiers. The FVIFT can be easily obtained from an IFT by simply removing

[1] We define F and G to be the major subformulas of $F \wedge G, F \vee G, F \Rightarrow G, \neg F$. Any $Q[x/t]$ is a major subformula of the formulas $\forall x.Q[x]$ and $\exists x.Q[x]$.

$$\begin{tikzcd}
 & & & & (((M \wedge N) \wedge (M \wedge N \Rightarrow P_1)) \Rightarrow^{\alpha} P_2)^{+} \arrow[dll] \arrow[drr] & & & \\
 & & ((M \wedge N) \wedge^{\alpha} (M \wedge N \Rightarrow P_1))^{-} \arrow[dl] \arrow[drr] & & & & P_2^{+} & \\
 & (M \wedge^{\alpha} N)^{-} \arrow[dl] \arrow[dr] & & & ((M \wedge N) \Rightarrow^{\beta} P_1)^{-} \arrow[dl] \arrow[dr] & & & \\
M^{-} & & N^{-} & (M \wedge^{\alpha} N)^{+} \arrow[dl] \arrow[dr] & & P_1^{-} & & \\
 & & M^{+} & & N^{+} & & &
\end{tikzcd}$$

Fig. 5. Indexed formula tree for $(((M \wedge N) \wedge (M \wedge N \Rightarrow P)) \Rightarrow^{\alpha} P)^{+}$. Subscripts of literals are added for later reference.

all nodes of type γ and δ. Note that when we remove a node of type γ we must α-relate its children. For our propositional logic example formula (1) the FVIFT is identical to its IFT displayed in Figure 5.

2.2 Logical Context and Replacement Rules

We have already referred to the concept of a context of a subformula inside a larger formula. This concept is essential for the understanding of CORE. Making use of annotations of signed formulas we are now able to give a more precise definition of the *logical context* of a formula: two formulas F and G belong to the same context if they are α-related.

Intuitively, all formulas that would occur together in a sequent calculus derivation of a goal G belong to the same context. Note that the information available in IFTs and FVIFTs is sufficient to statically determine the logical context of any given subformula.

In CORE we can use the formulas available in the context of a subformula directly as replacement rules in a proof. We formally define the notion of replacement rules as follows:

Definition 3 *(Replacement Rules) Let a be a node with polarity p in some* FVIFT *T. Then $i \rightarrow< v_1, \dots v_n >$ is a* replacement rule *for a, if*

1. i is α-related to a by the first common ancestor c, and
2. (a) $\{v_1, \dots, v_n\}$ contains exactly those occurrences that are β-related to i and occur below c (i.e. $c \prec v_i$), or
   (b) v_1 and i are left- and right-hand side of a negative equation or equivalence and $\{v_2, \dots, v_n\}$ are all occurrences that are β-related to v_1 and i and occur below c.

A positive (negative) occurrence M^{+} (M^{-}) without any β-related nodes generates $M^{-} \rightarrow true^{+}$ ($M^{+} \rightarrow false^{-}$) as a replacement rule.

CORE provides a set of 12 "calculus rules" which can be used to manipulate IFTs. The rule that is most important for us is the one that applies a replacement rule to a subformula. We do not formally introduce the complete set of calculus rules here and instead refer to [Aut03]. For the realization of the task structure another aspect of CORE is more important which is known as *window inference.*

2.3 Window Inference

On top of the calculus rules described above, CORE supports reasoning with a so called *window inference technique* which is based on ideas of [RS93]. In this section we describe this communication layer which makes it possible to focus the reasoning process to subformulas of the overall goal. This can be done by placing a *window* (focus) on a subtree of the Fvift. As a result, the surroundings of this window are hidden from the user. However, this operation does not alter the proof state but only restricts the view on it to a particular subtree. The context of the active window is then made available to the user as a list of replacement rules.

The window inference mechanism of CORE consists of rules to focus and unfocus certain subformulas as well as window versions of each of the CORE calculus rules. It is worth pointing out that these window inference rules do not extend the reasoning capabilities of CORE. Rather, the window versions of the calculus rules are internally realized purely based upon the CORE calculus rules and the rules for opening and closing windows.

Opening and Closing Windows The two basic window rules are those for opening and closing windows on subformulas (subtrees). By applying these operations to a Fvift we impose a *window structure* on this tree. As a window structure for a Fvift R we understand a partial function f_R from an enumerable set W of unique window identifiers into the set of subtrees of R (i.e. their roots). Intuitively, when opening a new window w on a subtree r in R then we add the tuple (w, r) to f_R. Closing a window is exactly the inverse operation which removes a tuple from f_R. We will depict a window structure f_R on a Fvift R by drawing boxes around subtrees r for which a window w exists (i.e. if $f_R(w) = r$). Figure 6 (left) shows the Fvift for $P(x) \Rightarrow (Q(x) \wedge R(x))^+$ after opening windows on $P(x) \Rightarrow (Q(x) \wedge R(x))^+, P(x)^-$, and $R(x)^+$ respectively. Figure 6 (right) shows the same tree with a new window structure where the window on $R(x)$ is closed.

Windows w that correspond to subtrees $f(w)$ that are maximal with respect $\prec$ are called *active windows.* These are the windows that can be manipulated by application of CORE calculus rules.

Definition 4 *(Subwindows) If w is a window in a* Fvift *R then the set of all* subwindows *of w is defined as*

$$Subwindows(w) = \{w' | f(w) \prec f(w') \text{ and } w' \text{ is a window in } R\}$$

Windows can be closed, provided that they do not contain any subwindows. Note that opening and closing of windows never affects the Fvift, but rather the "view" the user takes on the problem represented by the tree.

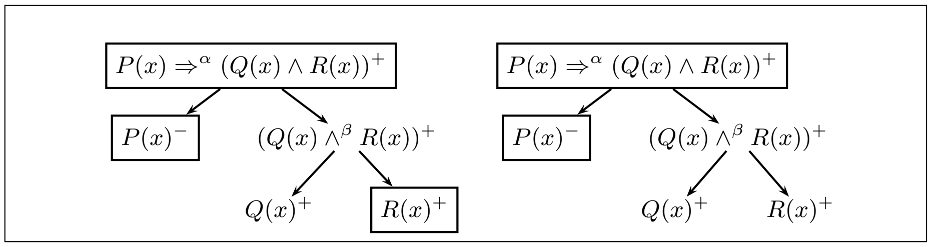

$\boxed{P(x) \Rightarrow^{\alpha} (Q(x) \wedge R(x))^{+}}$
$\boxed{P(x)^{-}}$
$(Q(x) \wedge^{\beta} R(x))^{+}$
$Q(x)^{+}$
$\boxed{R(x)^{+}}$
$\boxed{P(x) \Rightarrow^{\alpha} (Q(x) \wedge R(x))^{+}}$
$\boxed{P(x)^{-}}$
$(Q(x) \wedge^{\beta} R(x))^{+}$
$Q(x)^{+}$
$R(x)^{+}$

Fig. 6. Sample window structures for the FVIFT for $P(x) \Rightarrow (Q(x) \wedge R(x))^{+}$

2.4 Interactive Theorem Proving with CORE

Within CORE the user currently works with the window inference mechanism. This means that when the user 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 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[2]:

1. *Focus choice*: The selection of a subformula in the active window on which the user wants to focus the proof search.
2. *Rule choice*: The selection of a replacement rule.

While COREs inference mechanism is in principle well suited for interactive proof construction, 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. This is not what we had in mind when we described the intuitive presentation of a proof in Section 1. 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.

This motivates our *task structure* that allows for presenting the context of a subformula in an appropriate way, that is, as a list of assertions. The task structure also supports a uniform application mechanism for the replacement rules associated with these assertions.

3 Tasks – Organizing Proof Search

Informally, a task is an active window together with the assertions in its context. In the following, we formally define tasks and also introduce a set of rules operating on tasks. Thes rules realize the transformations needed for intuitive

[2] CORE also provides a cut-rule which we do not discuss here.

interactive proof construction with CORE (see the previous Section). These rules are in turn implemented via COREs window inference rules. Thus tasks introduce a new interaction layer on top of COREs window inference mechanism.

The concept of tasks was originally developed in the context of automated proof construction with ΩMEGAs [SBF+03] proof planner MULTI [Mei03]. The task structure presented here is an extension and adaptation of this work for the CORE system. An important new idea is to employ tasks as a common and uniform interface for automated and interactive reasoning processes simultaneously.

3.1 A Calculus for Tasks

Intuitively, tasks denote subgoals together with all the formulas that can be used to close this subgoal (all formulas that are α-related to the subgoal). Accordingly a task will simply be 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 5 *(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 node in R that is labeled with F is dependent in R. Otherwise we call w* unconditional in *R.*

Definition 6 *(Task) Let R be the* FVIFT *of the current proof state. A* task *T is a set of annotated windows $T = \{w_1, \ldots, w_n, g\}$ 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:*

1. *the subtrees denoted by the $w_1, \ldots, w_n$ are α-related between each other,*
2. *all support windows of T are unconditional in R'.*

We denote tasks $T = \{w_1, \ldots, w_n, g\}$ with goal window g as $w_1, \ldots, w_n \rhd g$. This notation is a horizontal version of the vertical representation of Figure 1.

Selecting one window as the goal window plays a role when reasoning interactively. The idea is that in an interactive proof only the goal window can be manipulated and therefore we will introduce a *shift*-rule for the explicit exchange of the goal window of a task.

Here we find an important difference to the sequents in the sequent calculus. In the sequent calculus it is relevant on which side of $\vdash$ a formula occurs. In CORE this information is already encoded in the polarities of each formula. We can therefore freely exchange the order of windows in a task. This also motivates the decision to define a task as a *set* of windows. Also note that 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 constraint that no support window of a task must be conditional is important because the content of support windows will be presented to the user as some 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; that is, the β-related subtrees would represent implicit "knowledge" which will be introduced in form of new proof obligations. We assume that this is less suited for interactive proof construction as the user might want to be able to see whether certain formulas in the context are dependent on further formulas, before applying them in the form of a replacement rule.

As an example consider the formula $(D^+ \Rightarrow^\beta (s = t)^-)^- \Rightarrow^\alpha B[s]^+$. Without the requirement that support windows must be unconditional we could generate the following task for the above formula: $(s = t)^- \rhd B[s]^+$. However, although this task gives the impression that the equation $s = t$ could be used directly to transform $B[s]$ to $B[t]$, this is not the case, as $s = t$ is dependent on D^+ and hence the replacement rule $s \rightarrow < t, D^+ >$, instead of $s \rightarrow t$ has to be used to carry out the transformation.

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

Definition 7 *(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 $Ax_1, \dots, Ax_n$ is represented by the system as a FVIFT for the signed formula $(Ax_1 \wedge \dots \wedge Ax_n \Rightarrow^\alpha G)^+$. The initial task then contains a window for the goal formula G as the goal formula and one window for each of the axioms as supports. This motivates the following definition of an *initial task*.

Definition 8 *(Initial Task) Let G be a formula and $Ax_1, \dots, Ax_n$ formulas that represent axioms from which G should be derived. Let further R be a* FVIFT *for $(Ax_1 \wedge \dots \wedge Ax_n \Rightarrow^\alpha G)^+$, w_i a window on Ax_i and g a window on G, then $w_1, \dots, w_n \rhd g$ is the* initial task *for G.*

From now on we will not distinguish anymore between a window and the formula it contains when we represent tasks. Hence, the initial window for G with axioms Ax_i will be represented as $Ax_1, \dots, Ax_n \rhd G$. Note that this implies that we can encounter tasks of the form $\Sigma, A^p, A^p \rhd G$. In this case the A^p are syntactically equal formulas that occur in different windows. However, because we are dealing with windows, rather than formulas we have to treat the A^ps as different entities.

A goal window of type β can be split into two tasks by decomposition of the goal formula. We always keep track of all tasks that are created during a proof attempt with the help of an *agenda*.

Definition 9 *(Agenda) An* agenda *is a set of tasks. An* initial agenda *is an agenda that contains only the initial task for a goal G.*

Tasks in the agenda can be manipulated by decomposition of the goal window, application of a replacement rule to the goal formula, closing of the window on

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

$$\frac{\Sigma \rhd \beta(A^{p_A}, B^{p_B})}{\Sigma \rhd A^{p_A} \qquad \Sigma \rhd B^{p_B}}\,\beta \qquad \frac{\Sigma \rhd \overset{\delta}{\gamma}\,((\Pi x.F[x])^p) \quad \Pi \in \{\forall, \exists\}}{\Sigma \rhd F[x]^p}\,\gamma\delta$$

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

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

$$\frac{\Sigma, F \rhd G}{\Sigma, G \rhd F}\,\mathit{Shift} \qquad \frac{\Sigma \rhd \diamond}{\emptyset}\,\mathit{Close}$$

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

$$\frac{\Sigma \rhd G}{\Sigma, A^- \rhd G \qquad \Sigma \rhd A^+}\,\mathit{Cut}(A)$$

Fig. 7. The task manipulation rules. The rules α_L, α_R, $\alpha_\neg$, and β are subsumed by the rule Focus.

the goal formula, or selection of a different goal window (*Shift*). In Figure 7 we introduce a set of rules that allow us to perform exactly these manipulations. The rules are of type $\mathit{rule} : \mathit{TASKS} \to 2^{\mathit{TASKS}}$, that is, by application of a rule to a task this task is replaced in the agenda by zero or more tasks. Hence, these rules can only be applied in forward direction. We now investigate each of these rules in turn.

α-decomposition Decomposition of a goal formula of type α is the simplest of the task manipulation rules. It realizes the decomposition of a task with a disjunctive goal formula. In order to decompose a formula $\alpha(A^{p_A}, B^{p_B})$ new windows on the A^{p_A} and B^{p_B} have to be opened and one new window has to be chosen as the new goal window. To be able to select either subwindow as new goal window we define three rules: α_L, α_R, and $\alpha_\neg$.

β-decomposition Decomposition of a task with a goal formula $\beta(A^{p_A}, B^{p_B})$ will lead to a split of the task into two tasks with goal formulas A^{p_A} and B^{p_B} respectively. This is realized through rule β.

β-decomposition requires a little more effort than α-decomposition. The reason is that there must be no conditional support windows in a task. That is, after decomposition of a goal formula $\beta(G_1, G_2)$ the constituents G_1 and G_2 can only become support windows if we make them unconditional when applying

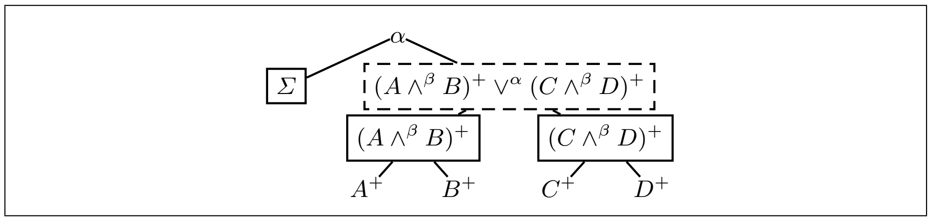


Fig. 8. FVIFT for the task $\Sigma, (A \wedge B)^+ \rhd (C \wedge D)^+$. Dashed lines indicate existing but inactive windows.

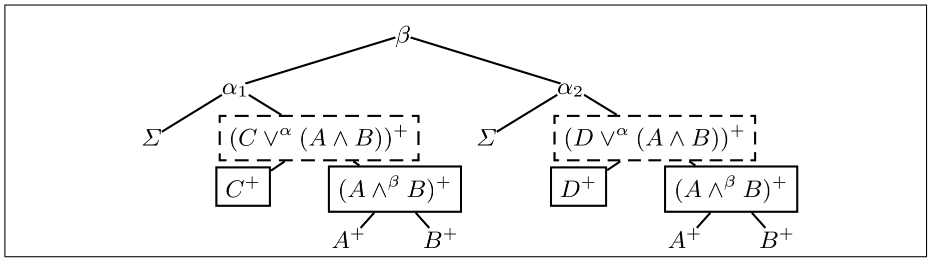


Fig. 9. A logically equivalent FVIFT to the one in Figure 8. This tree would result from the decomposition of the tree in Figure 8 if we realize the β-decomposition with the help of the *Schütte-rule*. The left subtree below α_1 corresponds to task $\Sigma, (A \wedge B)^+ \rhd C^+$ while the subtree below α_2 represents task $\Sigma, (A \wedge B)^+ \rhd D^+$.

β-decomposition. 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 [Sch77] and [Aut03]. Autexier [Aut03] shows that this *Schütte-rule* is admissible in the CORE calculus.

To see why we need the *Schütte-rule*, consider a task $\Sigma, (A \wedge B)^+ \rhd (C \wedge D)^+$. A FVIFT for this task is shown in Figure 8. If we implement the β-decomposition-rule with the help of the *Schütte-rule* we not only change the window structure of the FVIFT as we do with the α-decomposition-rule, but we also change the FVIFT itself. For instance, if we β-decompose the goal window $(C \wedge D)^+$ we obtain the new FVIFT in Figure 9.

We see that the task $\Sigma, (A \wedge B)^+ \rhd C^+$ corresponds to the minimal FVIFT R_1 with root-node α_1 and task $\Sigma, (A \wedge B)^+ \rhd D^+$ is represented by the minimal FVIFT R_2 below α_2 (cf. Figure 9). It is important to note that C^+ and D^+ are unconditional in the respective trees R_1 and R_2. Hence, we can easily make them support windows of a task. This will be important below when we define the *Shift*-rule.

$\gamma-$ and δ-decomposition Focusing on the major subformula of a γ- or δ-formula does not change the set of assertions in the context of this formula. It only has an effect on the variable conditions maintained by CORE.

Compound Decomposition Steps The single decomposition rules above can be combined in a macro-rule that allows us to focus directly on a particular subformula A^p inside the goal window of a task. 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 macro-rule *Focus* therefore provides great freedom in the selection of subwindows and simultaneously keeps track about the generated subgoals $\Sigma_2 \rhd G_1, \ldots, \Sigma_n \rhd G_n$ that appear as new tasks in the agenda. Accordingly, the *Focus*-rule can replace the rules $\alpha_R, \alpha_L, \alpha_\neg$, β, and $\gamma\delta$ since they are now subsumed.

FocusClose It is often necessary to undo a decomposition of the current goal window of a task, which has to be realized by closing the focus of the goal window G and with it, all other foci below the parent of G. The *FocusClose*-rule provides exactly this functionality. Note that the *FocusClose*-rule allows us to undo decomposition steps. Closing windows below a node of type α is easy. To close children below a β-node we need to use a reversed *Schütte-rule*.

The Shift*-rule* The *Shift*-rule changes the goal formula of a task. This is particularly important because the rules defined here only allow for manipulation of goal windows. Consider for instance a situation where we have a window on a formula A^p and a window for $(A \Leftrightarrow B)^-$ amongst the support windows for a goal G; that is, $\Sigma, (A \Leftrightarrow B)^-, A^p \rhd G$. If we now want to apply $A \Leftrightarrow B$ in a forward step to A^p we first have to make A^p the goal window. This can be done with the *Shift*-rule.

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

Closing tasks Tasks are removed from the agenda in case they are closed. This is done by the *Close*-rule which deletes any task $\Sigma \rhd \diamond$ from the agenda, where $\diamond \in \{true^+, false^-\}$.

Replacement Rule Application The way tasks are defined, all replacement rules for a goal window can be generated from the support windows of a task. In the rule *Apply* which applies a replacement rule to a task we to ensure that no "information" gets lost. To see what is meant consider a task of the form

$$\Sigma, G^+, (A^+ \Rightarrow^\beta B^-)^- \rhd A^-$$

The window on $(A^+ \Rightarrow^\beta B^-)^-$ justifies the replacement rule $A^- \rightarrow B^-$. Intuitively, application of the replacement rule to A^- should yield an additional window on B^- such that the task that results from application of this rule is $\Sigma, A^-, G^+, (A^+ \Rightarrow^\beta B^-)^- \rhd B^-$; i.e application of the rule should not remove A^- from the task. However, merely applying this rule to A^- would replace A^- by B^-, which would yield $\Sigma, G^+, (A^+ \Rightarrow^\beta B^-)^- \rhd B^-$. This is not quite what we want since we might need to make use of A^- again.

The solution to this problem lies in the application of the contraction rule to the goal window; that is, we copy the goal window before the application of the replacement rule.

The way the rule is defined here 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 subterm with rule *Focus* first. Alternatively the rule can easily be extended to operate on subformulas of i as well.

Cut-rule The introduction and speculation of new lemmata plays an important role for interactive theorem proving. For this we introduce the rule *Cut*. This rule allows to introduce a lemma A into the context of a goal G. As a consequence, a new task with the goal to prove that lemma is generated. Implementation of this rule in CORE is quite straightforward as CORE already provides a *cut*-rule which we can use directly at the task level when we update the window and task structure.

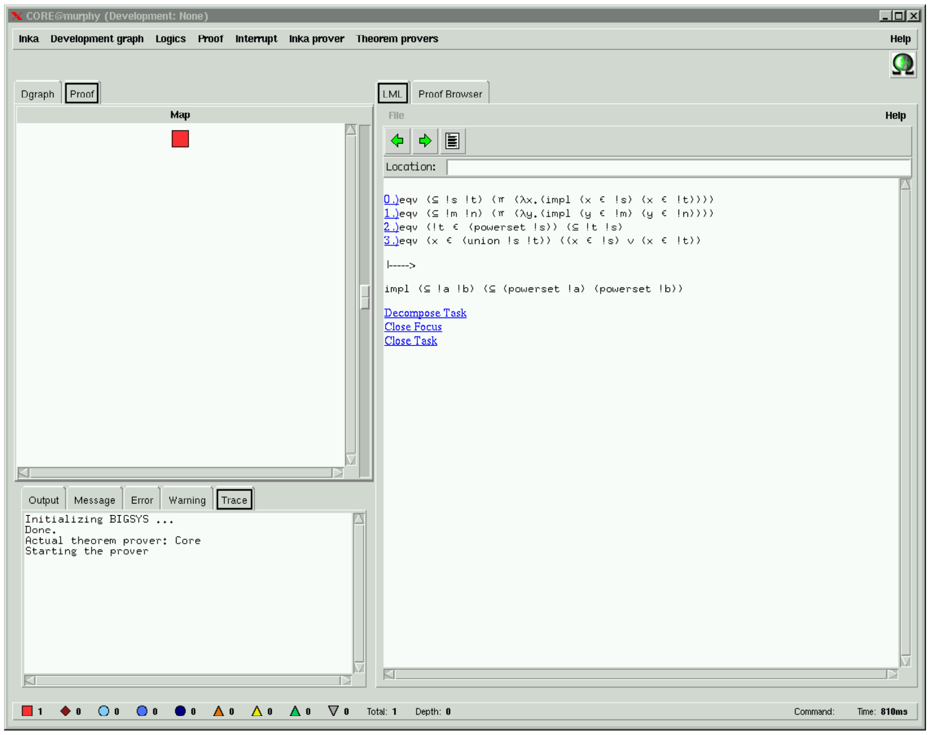
3.2 Tasks in Interactive Proof Search

Tasks enable a compact presentation of a proof situation. Instead of listing all replacement rules that can be generated from the context, the support windows give us access to the assertions which we can list line by line on top of the GUI screen, followed by the goal window at the bottom. Such a presentation of a task is meant to resemble the presentation of proofs in mathematical textbooks, where available assertions (axioms and definitions, etc) are mentioned before the theorem is stated[3].

In a system state where the user is shown the current task he has the following options to continue the proof. He can either focus on parts of the goal formula (this includes simple as well as compound decompositions) or apply any other of the task manipulation rules. Application of one of the assertions that are represented by the support windows are realized with the *Apply*-rule by simply clicking on the respective formula. The system then presents all replacement rules that can be generated from the chosen support window. It is then the choice of the user to identify the replacement rule that encodes the preferred application direction of the assertion. This is already an improvement to the situation where the user was presented with all replacement rules for the current window; that is, the replacement rules generated from *all* formulas in the context.

However, it is possible to do even better. For many of the rules that can be generated from a single support window in a task, one can see in advance that they are not applicable in the goal window. For other rules it is likely that they do not represent a suitable application of the formula from which they were generated. As a consequence, the system currently uses a heuristic to generate only those rules that are likely to represent the intended application direction of the assertion (see Section 4).

[3] Of course, in mathematical texts this is often done implicitly or indirectly in the sense that the available mathematical knowledge is referenced by stating the theory or domain context of a proof problem.



```
0.)eqv (⊆ !s !t) (π (λx.(impl (x ∈ !s) (x ∈ !t))))
1.)eqv (⊆ !m !n) (π (λy.(impl (y ∈ !m) (y ∈ !n))))
2.)eqv (!t ∈ (powerset !s)) (⊆ !t !s)
3.)eqv (x ∈ (union !s !t)) ((x ∈ !s) v (x ∈ !t))

|----->

impl (⊆ !a !b) (⊆ (powerset !a) (powerset !b))

Decompose Task
Close Focus
Close Task
```

```
Initializing BIGSYS ...
Done.
Actual theorem prover: Core
Starting the prover
```

Fig. 10. GUI of the CORE system. The initial task for the example from Section 4 is shown in the window on the right-hand side.

Hübner[Hüb03] 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 [SBF+03] theorem proving system and has now been adapted to this new environment which will provide the basis for the next generation of ΩMEGA.

4 Tasks - A Worked Example

In this section we illustrate interaction at the task layer at hand of an example. The proof problem is given as[4]

Theorem 1 *For sets A and B with $A \subseteq B$ holds that $2^A \subseteq 2^B$.*
This statement is encoded as:

$$\forall A, B. A \subseteq B \Rightarrow 2^A \subseteq 2^B \qquad (G)$$

We assume that the following axioms defining $\subseteq$ and 2^M are given:

$$\forall M, N. M \subseteq N \Leftrightarrow (\forall z. z \in M \Rightarrow z \in N) \qquad (Ax1)$$
$$\forall X, M. X \in 2^M \Leftrightarrow X \subseteq M \qquad (Ax2)$$

[4] We use 2^X to denote the powerset of a set X.

4.1 A Natural Deduction Proof

We first sketch a natural deduction proof of the above proof problem which will then be compared to the proof we construct at the task layer. Our natural deduction proof can be divided into 6 different parts as indicated by the boxes B1–B6 below. The conceptual steps addressed in these boxes are indicated by their headlines. Some subproofs employ derived rules (tactics), such as Transitivity of $\subseteq$ or subsequent applications of $\forall_I$ with $\forall_I^*$. Hence, they expand to even larger proofs in a pure natural deduction calculus.

$\boxed{\text{B1}}$ We start with goal G and focus on the right hand side of the implication.

$$\dfrac{\dfrac{\begin{array}{c}[a \subseteq b]^1\\ \vdots\\ 2^a \subseteq 2^b\end{array}}{a \subseteq b \Rightarrow 2^a \subseteq 2^b}\Rightarrow_I^1}{G}\forall_I^*$$

$\boxed{\text{B2}}$ Then we apply the definition of $\subseteq$ as given in $Ax1$.

$$\dfrac{\dfrac{\dfrac{Ax1}{2^a \subseteq 2^b \Leftrightarrow (\forall z.z \in 2^a \Rightarrow z \in 2^b)}\forall_E^*}{2^a \subseteq 2^b \Leftarrow (\forall z.z \in 2^a \Rightarrow z \in 2^b)}\Leftrightarrow_{E_l} \qquad \begin{array}{c}[a \subseteq b]^1\\ \vdots\\ \forall z.z \in 2^a \Rightarrow z \in 2^b\end{array}}{2^a \subseteq 2^b}\Rightarrow_E$$

$\boxed{\text{B3}}$ We focus on the right hand side of the implication of the current subgoal.

$$\dfrac{\dfrac{\begin{array}{c}[a \subseteq b]^1\\ [c \in 2^a]^2\\ \vdots\\ c \in 2^b\end{array}}{c \in 2^a \Rightarrow c \in 2^b}\Rightarrow_I^2}{\forall z.z \in 2^a \Rightarrow z \in 2^b}\forall_I$$

$\boxed{\text{B4}}$ We apply the definition of 2^M as given in $Ax2$.

$$\dfrac{\dfrac{\dfrac{Ax2}{c \in 2^b \Leftrightarrow c \subseteq b}\forall_E^*}{c \in 2^b \Leftarrow c \subseteq b}\Leftrightarrow_{E_l} \qquad \begin{array}{c}[a \subseteq b]^1\\ [c \in 2^a]^2\\ \vdots\\ c \subseteq b\end{array}}{c \in 2^b}\Rightarrow_E$$

$\boxed{\text{B5}}$ We apply transitivity of $\subseteq$.

$$\dfrac{\begin{array}{c}[a \subseteq b]^1\\ [c \in 2^a]^2\\ \vdots\\ c \subseteq a\end{array} \qquad [a \subseteq b]^1}{c \subseteq b}\text{Transitivity-of-}\subseteq$$

$\boxed{\text{B6}}$ We finish the proof by applying the definition of 2^M once more.

$$\dfrac{\dfrac{\dfrac{Ax2}{c \in 2^a \Leftrightarrow c \subseteq a}\,\forall^*_E}{c \in 2^a \Rightarrow c \subseteq a}\,\Leftrightarrow_{E_r} \qquad [c \in 2^a]^2}{c \subseteq a}\,\Rightarrow_E$$

4.2 A Proof at Task Layer

At task layer the natural deduction derivations described in B1-B6 collapse into single interactions. We present tasks as in Figure 1, that is, each support window is displayed in one line and the content of the goal window is shown below the supports.

In Figure 10 we can see that the presentation of tasks we use here is very similar to the way they are actually presented in the current GUI of our system. In the GUI we are experimenting with different colored formulas to indicate the polarities of formulas in order to improve readability.

The initial task of our proof problem is:

$$\begin{bmatrix} (Ax1) & (M \subseteq N \Leftrightarrow (z \in M \Rightarrow z \in N))^- \\ (Ax2) & (X \in 2^M \Leftrightarrow X \subseteq M)^- \end{bmatrix} \\ (A \subseteq B \Rightarrow 2^A \subseteq 2^B)^+ \tag{4}$$

The first step is straightforward: we focus on the right hand side of the implication and obtain

$$\begin{bmatrix} (Ax1) & (M \subseteq N \Leftrightarrow (z \in M \Rightarrow z \in N))^- \\ (Ax2) & (X \in 2^M \Leftrightarrow X \subseteq M)^- \\ (3) & (A \subseteq B)^- \end{bmatrix} \\ (2^A \subseteq 2^B)^+ \tag{5}$$

where $(A \subseteq B)^-$ occurs now as a new support window. Next we apply the definition of $\subseteq$ to the goal formula $(2^A \subseteq 2^B)^+$. In the GUI this is done by clicking on the assertion formula $Ax1$. In general, the problem of determining the applicable rules involves higher-order unification and is hence not decidable. We are therefore employing a heuristic approach that computes for the selected formula a set of replacement rules that are "most likely" appropriate, that is, we currently employ an imperfect filter. From the suggested rules the user then has to select one rule for application. For the given situation the heuristic computes the single applicable rule (i.e. $(M \subseteq N)^+ \rightarrow < (z \in M \Rightarrow z \in N)^+ >$). The application of this rule results in the new task:

$$\begin{bmatrix} (Ax1) & (M' \subseteq N' \Leftrightarrow (y \in M' \Rightarrow y \in N'))^- \\ (Ax2) & (X \in 2^M \Leftrightarrow X \subseteq M)^- \\ (3) & (A \subseteq B)^- \end{bmatrix} \\ (z \in 2^A \Rightarrow z \in 2^B)^+ \tag{6}$$

Note that although the application of the replacement rule generated from $Ax1$ has instantiated the variables M and N, we now have an uninstantiated version of $Ax1$ with fresh variable copies M' and N' in the supports. From now on we always assume that after application of an assertion we have an uninstantiated version in the supports, without indicating this explicitely. In the system these fresh copies of applied and thus instantiated assertions are automatically generated by CORE.

In a next step, we apply $Ax2$ to the subformulas $z \in 2^A$ and $z \in 2^B$ respectively. After clicking on the assertion $Ax2$ the system suggests to apply the rule $X \in 2^M \rightarrow < X \subseteq M >$ where we merely have to determine the application position, that is, $(z \in 2^A)$ or $(z \in 2^B)$. We apply it first to the former and then repeat the step and apply to the latter subformula. We obtain the new task:

$$\begin{bmatrix} (Ax1) & (M' \subseteq N' \Leftrightarrow (y \in M' \Rightarrow y \in N'))^- \\ (Ax2) & (X \in 2^M \Leftrightarrow X \subseteq M)^- \\ (3) & (A \subseteq B)^- \end{bmatrix} \qquad (7)$$

$$(z \subseteq A \Rightarrow z \subseteq B)^+$$

We now focus on the right hand side of the implication in the new subgoal and obtain:

$$\begin{bmatrix} (Ax1) & (M' \subseteq N' \Leftrightarrow (y \in M' \Rightarrow y \in N'))^- \\ (Ax2) & (X \in 2^M \Leftrightarrow X \subseteq M)^- \\ (3) & (A \subseteq B)^- \\ (4) & (z \subseteq A)^- \end{bmatrix} \qquad (8)$$

$$(z \subseteq B)^+$$

The rest of the proof consists now in showing (or, if available, applying a lemma for) the transitivity of $\subseteq$.

4.3 Discussion

The previous example illustrates that whole proof blocks in natural calculus are replaced by single and far more intuitive steps at task layer (which are probably in a different order). In summary, we can identify the following three main advantages of the task layer for interactive proof construction:

- It allows to conduct proofs in a way that resembles natural deduction style proofs while at the same time it hides many distracting details of the natural deduction level. Assertion formulas can now be applied to the goal window in a uniform way with the help of simple mouse clicks in the GUI.
- Proofs at the task layer are much shorter and fewer interactions are required. For instance, we do not need to apply quantifier elimination rules because this is done implicitly by CORE. Focusing onto subformulas is realized as single steps, which are ideally realized again in the GUI by simple mouse clicks.

– Tasks allow us to present assertions in an intuitive way to the user. This is a prerequisite for fruitfully supporting the above aspects. In combination with the fact that many branchings of the proof are avoided (cf. backward application the $\Rightarrow_E$-rule) this supports a convenient way to construct proofs.

Further aspects are:

– Since the task level is simply an additional, independent layer on top of CORE soundness of our approach is guaranteed by the underlying CORE system (see [Aut03]).
– Proof methods and automated theorem provers can be easily defined and added to the system to support automated proof construction (see [Hüb03]). This means that tasks can not only be manipulated by application of decomposition and basic rewriting steps but also through application of macro-steps that are described as proof methods.

Work in the direction of intuitive proof presentation and manipulation has been described by Bertot et al. [BKT94]; they show how to focus in a sequent calculus proof to certain subformulas. However, although the approach goes into the right direction it is still very tightly connected to the sequent calculus.

5 Conclusion and Future Work

We have introduced task structures as an intuitive front end for interactive theorem proving based on the CORE system. Our approach is not committed to a particular underlying calculus and it supports very flexible proof construction. This is different to the proof by pointing approach which is strongly connected to the sequent calculus. In Section 4 we have illustrated how explicit decomposition of assertions as required in sequent and natural deduction style calculi is replaced by a uniform application mechanism that avoids decompositions.

In a next step we will now evaluate how well the task transformation rules are suited to model human proofs in the domain of naive set theory. Benzmüller et al. [BFG$^+$03a,BFG$^+$03b] have collected in an experiment on tutorial dialogues with a mathematical assistant system a first corpus of proofs in this domain against which we want to evaluate the naturalness of the task manipulation rules. We also want to investigate whether assertion application at the task layer can be further improved by stronger and better filter; see for instance [VBA03] for ongoing work.

References

[Aut01] Serge 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.

[Aut03] Serge Autexier. *Hierarchical Contextual Reasoning.* PhD thesis, University of the Saarland, 2003. to appear.

[BFG$^+$03a] 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.

[BFG$^+$03b] 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.

[BKT94] Yves Bertot, Gilles Kahn, and Laurent Thery. Proof by pointing. In *Theoretical Aspects of Computer Science (TACS)*, 1994.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39:572–595, 1935.

[Hua94] Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction*, pages 738–752. Springer-Verlag, 1994.

[Hüb03] Malte Hübner. Interactive theorem proving with indexed formulas. Master's thesis, Fachbereich Informatik, University of the Saarland, 2003.

[Mei03] Andreas Meier. *Proof-Planning with multiple strategies*. PhD thesis, University of the Saarland, 2003. forthcoming.

[PB02] Florina Piroi and Bruno Buchberger. Focus windows: A new technique for proof presentation. In Jaques Calmet, Belaid Benhamou, Olga Caprotti, Laurent Henocque, and Volker Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Computation*, number 2385 in LNAI, pages 337–341. Springer, 2002.

[RS93] Peter J. Robinson and John Staples. Formalizing a hierarchical structure of practical mathematical reasoning. *Journal of Logic Computation*, 3(1):47–61, 1993.

[SBF$^+$03] 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.

[Sch77] K. Schütte. *Proof Theory*. Springer Verlag, 1977.

[Smu68] Raymond R. Smullyan. *First Order Logic*. Springer, 1968.

[VBA03] Quoc Bao Vo, Christoph Benzmüller, and Serge 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).

[Wal90] Lincoln 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.