Proposing a Task Interface for Proof Assistants

Serge Autexier and **Christoph Benzmüller** and **Malte Hübner** and **Andreas Meier** and **Martin Pollet** and **Claus-Peter Wirth** [1]

Abstract. Proof assistants are usually developed bottom-up. A predominant choice thereby is that of the logical base calculus and typically this choice imposes frame constraints on the system. In many proof assistants the logic layer is enriched by more abstract level reasoning tools and ideally these tools are sufficient for 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.

In this paper we argue for a top-down development of proof assistants where the abstract layer is kept as independent as possible from the logic layer. 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.

1 Introduction

Proof assistants (PAs) are usually developed bottom-up. A predominant choice thereby is that of the logical base calculus and typically this choice imposes some frame constraints on the possible functionalities, strengths and weaknesses, and the proof-theoretic flavor of the PA.

Natural deduction or sequent calculus are widely believed to be the most natural and intuitive option for interactive proof. Hence, they are employed in many PAs as the logical base layer (BL). In many PAs the BL is then enriched by more abstract level reasoning support tools, e.g., by so called proof tactics [9], proof methods [6, 14], or by integrated external specialist reasoners [15, 17, 2, 18, 12]. Ideally interactive proof and abstract level automation, for instance, as proposed in proof planning [6, 14], only employs the reasoning tools provided at the abstract layer (AL), so that non-interesting logic layer details can be hidden first and later independently filled-up when "expanding" from the AL to the BL.

A drawback of the bottom-up approach is that it usually causes a strong interweavement of the two layers, in particular, a unnecessarily strong dependence of the AL upon the BL. Problems imposed by this are discussed, for instance, in [7] and [4]. Here we point to the following arguments:

- Modifications of the BL typically cause major adaptations of the tools and control mechanisms provided at the AL.
- The BL typically heavily influences the flavor and appeal of the PAs user interface, which is often designed to support the particular "proof style" of the BL.
- PAs supporting white-box integrations of external systems typically provide mechanism to translate the proof representation formats of the external reasoners into their base calculus. Modifications of the BL usually require non-trivial adaptations of these transformation mechanisms.
- PA users have to learn and adopt the peculiarities and the proof style of the system's BL.

In this paper we argue for a top-down development of PAs. We particularly argue for the design of an AL that is as far as possible independent of a PA's BL. Our AL proposal for the new ΩMEGA^{CORE} system is called the *task interface* (TI).

Relevant TI design issues are: What notion of proof is reflected? Is this notion mathematically and/or cognitively adequate? What level of granularity in the argumentative steps is supported? Can the TI simultaneously support interaction and automation (and an interleaving of both)? Does the TI sufficiently hide non-interesting logic level details while it reveals some crucial constraints for proof construction at the AL? How complicated is it to provide a suitable expansion mechanism capable of transforming AL proofs in the TI into verifiable BL proofs? What are the options for a concrete BL that can fruitfully support our TI?

While we propose to keep the AL as independent as possible from the BL, the latter two questions are nevertheless important: We do not propose PAs lacking a sound logical basis. Instead, we propose to distinguish in PAs better between AL reasoning and expansion into verifiable BL proofs. Lean and elegant proof expansion/contraction mechanisms that can mediate between the two layers are hence important. They in turn may gain from the "right" choice of a BL for a previously chosen AL.

Our top-down viewpoint is motivated not least by the corpus [20] of human constructed proofs gained in the DIALOG project (on *Tutorial Natural Language Dialog in Mathematics*) at Saarland University [3]. These proofs were collected in a Wizard-of-Oz study where 25 students were asked to interactively develop proofs in naive set theory (employing typed natural language / formula input). Thereby no preconditions on a particular proof style or a formal reasoning framework were assumed and in fact the corpus shows that logical peculiarities do not play a role (or only a minor role) in these human constructed proofs. Further motivation is given by the idea of interactive island proofs as described in [16]. A TI has already been proposed and employed in automated proof planning in the old ΩMEGA system [12]. However, this work still suffers from a too strong dependency of its TI upon the old ΩMEGA's BL (a higher-order natural deduction cal-

[1] Fachrichtung Informatik, Universität des Saarlandes, Saarbrücken, Germany *and* Max-Planck-Institut für Informatik, Stuhlsatzenhausweg, Saarbrücken, Germany *and* DFKI, Stuhlsatzenhausweg, Saarbrücken, Germany

culus); it is also not sufficiently integrated with interactive proof development. The TI presented in this paper improves and extends our work in [10]; the new aspects include: and/or trees supporting conjunctive and disjunctive proof branches, variable conditions, and a formal grammar for our TI.

The structure of this paper is as follows: First we illustrate "natural" reasoning at the AL with an example. Next, we present our TI for the new ΩMEGA^{CORE} PA. TI proof development is then illustrated employing our previous example. Finally, we briefly discuss the role of *CORE*, which is our concrete BL choice in ΩMEGA^{CORE}, and sketch how mediation between the two layers is realized.

2 Motivating Example

To motivate our notion of AL reasoning we look at the sample problem of showing that the two residue class structures $S_1 = (\mathbb{Z}_4, x\bar{+}y)$ and $S_2 = (\mathbb{Z}_4, (x\bar{+}y)\bar{+}\bar{1}_4)$ are isomorphic, where $\mathbb{Z}_4$ is the set of the four residue classes modulo 4, $\bar{+}$ denotes the addition on residue classes, and $\bar{1}_4$ denotes the residue class modulo 4 with residue 1 (see [13] for details on formalizing and proof planning residue class problems). Our initial goal is:

$$(Thm) \qquad isomorphic(S_1, S_2)$$

A straightforward proof for this conjecture consists of finding a mapping $h : S_1 \to S_2$ and showing that h is (i) a homomorphism and (ii) bijective. However, a closer look reveals that there are multiple alternatives for carrying out each step. In the first step we have to choose a h. For this we can employ, for instance, a model generator. Actually there are two suitable candidates for h:
$h_1(\bar{0}_4) = \bar{3}_4$, $h_1(\bar{1}_4) = \bar{0}_4$, $h_1(\bar{2}_4) = \bar{1}_4$, $h_1(\bar{3}_4) = \bar{2}_4$
$h_2(\bar{0}_4) = \bar{3}_4$, $h_2(\bar{1}_4) = \bar{2}_4$, $h_2(\bar{2}_4) = \bar{1}_4$, $h_2(\bar{3}_4) = \bar{0}_4$.

Further alternatives concern the *representation* of h_1 and h_2. We can represent h_1 as polynomial $h_1(x) = x\bar{+}\bar{3}_4$, whereas for h_2 no polynomial representation is obvious such that a pointwise representation is more appropriate. Depending on the alternative representations of h_1 and h_2 we can also perform different subproofs to show that the candidates are homomorphisms and bijective. For instance, to prove that h_1 is a homomorphism, we can reduce the homomorphism goal

$$\forall x{:}\mathbb{Z}_4, y{:}\mathbb{Z}_4 \centerdot h(x\bar{+}y) = (h(x)\bar{+}h(y))\bar{+}\bar{1}_4 \tag{1}$$

to the equation

$$(x\bar{+}y)\bar{+}\bar{3}_4 = ((x\bar{+}\bar{3}_4)\bar{+}(y\bar{+}\bar{3}_4))\bar{+}\bar{1}_4 \tag{2}$$

which holds since we can reduce the right hand side with modulo arithmetic to $x\bar{+}y\bar{+}\bar{3}_4$, i.e., to the left hand side.

For the point-wise given function h_2 we can perform a *case split* where we reduce the quantified formula in (1) in $4{*}4 = 16$ cases for each of the instantiations for x and y. This tedious approach is, of course, also possible for h_1. Similarly, there are also alternatives to prove the remaining part that the functions are bijective for h_1 and h_2.

This discussion illustrates that at AL we are typically confronted with different conceptually motivated alternatives which have a different appeal from typical logic-level alternatives. In the example above these alternatives boil down to: different possibilities to *instantiate* and *represent h*. Further alternatives concern *mathematical proof techniques*, e.g., to introduce a case split or not. Each alternative leads to a different proof attempt. A reasoning layer that is cognitively and mathematically adequate should allow the user to directly carry out the proof steps described above and to carry out different proof attempts in parallel, comprising also failing proof attempts. If, for instance, no suitable h can be found, it might be useful to experiment and learn from failing candidates. The AL therefore should

1. provide a proof data structure that can maintain *different proof attempts* simultaneously,
2. be able to handle *different instantiation of variables*,
3. support one-to-one representation of the essential proof steps as they occur in textbook proofs or in tutorial dialogs on mathematics, and render further steps unnecessary, and
4. support the integration of reasoning specialists.

3 Tasks, Task Trees and Forests

We now present a declarative language to encode as adequately as possible the style of reasoning sketched above. The language is based on the notion of a *task*, which represents one subproblem that has to be solved as part of the overall proof problem. A task is denoted by $\varphi_1^{p_1}, \ldots, \varphi_n^{p_n} \rhd \varphi_0^{p_0}$, where for all i φ_i is a formula and $p_i \in \{+, -\}$ are polarities. The polarity p_i of each *signed formula* $\varphi_i^{p_i}$ thereby indicates whether φ_i is an assumption or a goal. In order to explicitly represent the focus of attention in such a task, it allows to distinguish one signed formula, i.e. $\varphi_0^{p_0}$ in the example. Assuming formulas are defined by the non-terminal grammar symbol $\mathsf{FORMULA}$, the grammar rule for tasks is

$$\mathsf{TASK} ::= (\mathsf{FORMULA}^{\{+|-\}})^* \rhd \mathsf{FORMULA}^{\{+|-\}}$$

Based on tasks we define the notion of a *task tree (TT)* and *task forest (TF)* as a collection of task trees with unique identifiers (ID).

A *task tree* is an inductively defined structure that consists of leaf nodes denoted as *open tasks*

$$\mathsf{Open}(\varphi_1^{p_1}, \ldots, \varphi_n^{p_n} \rhd \varphi_0^{p_0})$$

and tasks that have been refined by some *action*:

$$\mathsf{Node}(\varphi_1^{p_1}, \ldots, \varphi_n^{p_n} \rhd \varphi_0^{p_0}, \mathsf{action}).$$

For task refinement we distinguish between (1) tasks that have been refined by some action A into conjunctively related subtasks (including the cases of one and zero subtasks), (2) tasks for which alternatives of such actions A_i have been introduced. For the latter we introduce the notion of *OrAction*s

$$\mathsf{OrAction}(\text{description}, \langle \mathsf{action}_1 \parallel \ldots \parallel \mathsf{action}_n \rangle)$$

which describe the different possible alternative refinements of some goal. This allows us to represent alternative proof attempts (such as application of different possible facts), as well as complex actions which generate different possible proof attempts, for instance, with the help of a model generator as illustrated in the previous section.

Finally, an *action* can be of three types: (1) Either it is a simple action which refines the task to a list of subtasks

$$\mathsf{Action}(\text{description}, \langle \mathsf{TT}_1, \ldots, \mathsf{TT}_n; \sigma; \rho \rangle),$$

$$\begin{array}{rcl}
\mathsf{TF} & ::= & (\mathsf{ID};\mathsf{TT});\mathsf{TF}^* \\
\mathsf{TT} & ::= & \mathsf{Open}(\mathsf{TASK}) \mid \mathsf{Node}(\mathsf{Task},\mathsf{ALTACTION}) \\
\mathsf{ALTACTION} & ::= & \mathsf{ACTION} \\
 & \mid & \mathsf{OrAction}(\mathsf{DESC},\langle \mathsf{ACTION}(\| \ \mathsf{ACTION})^+\rangle) \\
\mathsf{ACTION} & ::= & \mathsf{Action}(\mathsf{DESC},\langle \mathsf{TT}^*;\mathsf{SUBST};\mathsf{SUBST}\rangle) \\
 & \mid & \mathsf{Hyp}(\mathsf{Id};\mathsf{SUBST};\mathsf{SUBST}) \\
 & \mid & \mathsf{Lem}(\mathsf{Id};\mathsf{SUBST};\mathsf{SUBST})
\end{array}$$

Figure 1. Grammar for the task interface TI.

or it discharges the task either (2) by simple lemma application

$$\mathsf{Lem}(n;\sigma;\rho),$$

where n refers to some task tree in the task forest and different from the actual task tree, or (3) by some inductive argument as presented in [19]

$$\mathsf{Hyp}(n;\sigma;\rho),$$

where n now denotes *any* task tree in the task forest. Note that we do not make any assumption about how the action was applied. More specifically, we do not enforce the application of substitutions for free variables, and only require that each action indicates

- the substitution σ of free variables it introduced, and
- which part ρ of the overall substitution that governs the task, but were not yet applied, were necessary in order to apply the action.

Intuitively the σs determine for each subtree the *maximal* instantiation computed by actions to that subtree, while the ρs represent the *minimal* instantiations that are actually *necessary* to reach that subtree. For instance, given some maximal and minimal substitutions σ' and ρ', the maximal and minimal substitutions for the subtrees of the action

$$\mathsf{Action}(\text{description}, \langle \mathsf{TT}_1,\ldots,\mathsf{TT}_n;\sigma;\rho\rangle),$$

are $\sigma' \circ \sigma$ and $\rho' \circ \rho$, where $\circ$ denotes the composition of substitutions. Note that these substitutions are always unique and that it must always hold that the minimal substitution of each subtree is *more general* than the maximal substitution of that subtree.

The whole grammar for our TI is given in Fig. 1. Based on that grammar we now discuss the invariants we enforce for task trees. The first invariant is that we want to enforce a global substitution which is common to all task trees contained in a task forest. This requires the notion of a global substitution for some single task tree, which, in turn, can be roughly characterized by requiring that all leaf nodes in the task tree share the same substitution.

However, trying to define that notion formally is hampered by the presence of alternative branches inside a task tree, which may result in different substitutions of the leaf nodes.

Pure Task Trees and Forests. In order to solve that problem we first define the notion of *pure task trees* which do not contain alternatives by restricting the grammar rule for $\mathsf{ALTACTION}$ to $\mathsf{ALTACTION} ::= \mathsf{ACTION}$.

For pure task trees we now define the notion *well-formed pure task trees* by requiring that *all* leaf nodes share the same substitutions. Based on that notion we can define a function μ that maps each (named) task tree into a forest of (named) pure task trees. Thereby the self-references contained in hyp-actions need to be adapted accordingly. Exploiting that function enables us to formally define *well-formedness* and other notions, such as *open* (non-finished proof attempt) and *closed* (finished proof), for arbitrary task trees.

The TI grammar and the substitution invariants provide frame constraints for the structuring of proofs at the AL. Concrete proof refinement operators 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 action by declaratively describing the refinement of a task into successor tasks; such *oracle steps* adapt the idea of interactive island proofs as introduced in [16]. Thus, the concretely available actions in a PA may vary from rather "safe" ones (those who directly guarantee logical soundness at the BL, e.g., traditional tactics constructed on top of the PA's BL) to highly "unsafe" ones (those with non-sharp application constraints causing frequent failing BL expansions, e.g., human constructed oracle steps).

4 Example Proof Development

In this section, we develop the example from Sec. 2 in the TI introduced in the previous section. We do not address how task trees and tasks can be suitably presented to a user (see [10]) but focus on the development of a proof in the TI.

Although we do not distinguish between different types of actions in the TI (in particular, not with respect to their "logical meaning", i.e., their expansion to the BL) we can classify the following types of actions that we use in the example:

1. Application of *assumptions and definitions*. Steps such as the expansion of a definition or the application of an assumption are typically carried out rather directly in interactive theorem proving formalisms such as ΩMEGA^{CORE} or the *proof by pointing* approach (see [5]). Since these proof steps are realized by rewriting steps in the ΩMEGA^{CORE} system we refer to them uniquely as ContRew. When presenting a ContRew action in a task tree we will ignore details such as which assumption or definition has been applied and at which position in a goal formula of a task.
2. Application of *decomposition steps*, i.e., steps that decompose logical connectives and quantifiers. These actions will be uniquely described as Decomp.
3. Application of *abstract proof steps* or proof steps that require to carry out *computations*. Such proof steps are encoded as methods or tactics in classical interactive theorem proving systems and can comprise calls of external systems such Computer Algebra Systems or constraint solvers. We use in the example two methods from the residue class domain (see [13]). The method SimpRes simplifies equations on residue classes. The method $\mathsf{FindIso}(S_1,S_2)$ invokes a model generator to find isomorphisms from structure S_1 into S_2. If possible, $\mathsf{FindIso}(S_1,S_2)$ returns candidate isomorphisms as polynoms. However, if it fails to create polynom representations it returns a candidate isomorphism as a pointwise function.

Action types 1 and 2 are relatively safe, i.e., usually easy to expand to the BL, while action types such as 3 involving calls to external reasoners are often far less safe. As an alternative to the use of these actions as illustrated below, the user could also structure the proof in the TI as a network of absolutely unsafe oracle steps similar to the island proofs in [16].

The conjecture is formally described by the initial task $\Sigma \rhd \mathit{isomorphic}(S_1, S_2)$, where Σ contains the following definitions:[2]

$$\mathit{isomorphic}(S,T) \Leftrightarrow \exists h.\mathit{iso}(h,S,T) \tag{3}$$

$$\mathit{iso}(h,S,T) \Leftrightarrow \mathit{hom}(h,S,T) \wedge \mathit{bij}(h,S,T) \tag{4}$$

$$\begin{aligned}&\mathit{hom}(f,S,T) \Leftrightarrow \\ &\qquad \forall x{:}\mathcal{D}_S, y{:}\mathcal{D}_T\textbf{.}\, f(x \circ_S y) = (f(x) \circ_T f(y))\end{aligned} \tag{5}$$

TI proof development starts with the initial task tree

$$\begin{tikzcd}[row sep=small] \mathsf{TT} \arrow[d, dash] \\ \mathsf{Open}(\Sigma \rhd \mathit{isomorphic}(S_1, S_2)) \end{tikzcd}$$

In the following, we will successively refine this task tree to develop a TI proof. The complete resulting TI proof tree is illustrated in Fig. 2 in the Appendix. To ease the presentation we show in this section only the changes of the fringe of the task tree that result from the refinements.

Step 1: The first step applies the definition (3), which results in a $\mathsf{ContRew}$ action. The new task tree is then

$$\begin{tikzcd}[row sep=small] \mathsf{TT} \arrow[d, dash] \\ \mathsf{Node}(\Sigma \rhd \mathit{isomorphic}(S_1, S_2), \mathsf{ContRew}(\emptyset,\emptyset)) \arrow[d, dash] \\ \mathsf{Open}(\Sigma \rhd \exists h.\mathit{iso}(h,S,T)) \end{tikzcd}$$

where $\mathsf{ContRew}(\emptyset,\emptyset)$ states that the action neither introduces nor depends on substitutions.

Step 2: In the second step we apply the method $\mathsf{FindIso}(S_1, S_2)$ to generate two candidate isomorphisms. The method finds the isomorphisms we described in Sec. 2 and returns a polynomial representation for the first candidate $h_1(x) = \lambda x\textbf{.}x\bar{+}\bar{3}_4$, whereas it presents the second isomorphism pointwise $h_2(\bar{0}_4) = \bar{3}_4$, $h_2(\bar{1}_4) = \bar{2}_4$, $h_2(\bar{2}_4) = \bar{1}_4$, $h_2(\bar{3}_4) = \bar{0}_4$. Both solutions give rise to introduce an or-branch in our task tree

$$\begin{tikzcd}[row sep=small] & \vdots \arrow[d, dash] & \\ & \mathsf{Node}\begin{pmatrix}\Sigma \rhd \exists h.\mathit{iso}(h,S,T), \\ (\mathsf{FindIso}h \mapsto \lambda x\textbf{.}x\bar{+}\bar{3}_4, \emptyset)||(\mathsf{FindIso}h \mapsto h_2, \emptyset)\end{pmatrix} \arrow[d, dash] & \\ & \| \arrow[dl, dash] \arrow[dr, dash] & \\ A & & B \end{tikzcd}$$

where

$\mathsf{A} \;=\; \mathsf{Open}(\Sigma \rhd \mathit{iso}(h, S_1, S_2))$

$\mathsf{B} \;=\; \mathsf{Open}(\Sigma, h_2(\bar{0}_4) = \bar{3}_4, \ldots, h_2(\bar{3}_4) = \bar{0}_4 \rhd \mathit{iso}(h, S_1, S_2))$

$\mathsf{FindIso}$ is an OrAction that introduces two different substitutions for h. Thus, the resulting two tasks are disjunctively related. This step corresponds to a human proof attempt, where a mathematician conjectures mappings h and subsequently tests whether they are bijective homomorphisms. In case the first tests fails he would switch to the next possible candidate for h. Since $\mathsf{FindIso}$ provides two promising candidates for h both are introduced as alternatives.

We only follow the right branch (A) to show that the first candidate is an isomorphism.

[2] $\mathcal{D}_S$ and $\mathcal{D}_S$ are the domains of structure S and T, respectively, $\circ_S$ and $\circ_T$ are the binary operations of S and T, respectively.

Step 3: We expand the definition of *iso* by a contextual rewriting step.

$$\begin{tikzcd}[row sep=small] {} \arrow[d, dash] \\ \mathsf{Node}(\Sigma \rhd \mathit{iso}(h, S_1, S_2), \mathsf{ContRew}(\emptyset,\emptyset)) \arrow[d, dash] \\ \mathsf{Open}(\Sigma \rhd \mathit{hom}(h, S_1, S_2) \wedge \mathit{bij}(h, S_1, S_2)) \end{tikzcd}$$

Step 4: A decomposition of the open node yields two new nodes at the fringe:

$$\begin{tikzcd}[row sep=small] & \vdots \arrow[d, dash] & \\ & \mathsf{Node}(\Sigma \rhd \mathit{hom}(h, S_1, S_2) \wedge \mathit{bij}(h, S_1, S_2), \mathsf{Decomp}(\emptyset,\emptyset)) \arrow[dl, dash] \arrow[dr, dash] & \\ \mathsf{Open}(\Sigma \rhd \mathit{hom}(h, S_1, S_2)) & & \mathsf{Open}(\Sigma \rhd \mathit{bij}(h, S_1, S_2)) \end{tikzcd}$$

Next, we focus on the left task.

Step 5: The expansion of the definition of *hom* results in the following new node at the fringe of the task tree:

$$\begin{tikzcd}[row sep=small] \mathsf{Node}(\Sigma \rhd \mathit{hom}(h, S_1, S_2), \mathsf{ContRew}(\emptyset,\emptyset)) \arrow[d, dash] \\ \mathsf{Open}(\Sigma \rhd h(x\bar{+}y) = (h(x)\bar{+}h(y))\bar{+}\bar{1}_4) \end{tikzcd}$$

Step 6: To show that the equation holds, we apply the simplification method $\mathsf{SimplifyRes}$ for residue class expressions. This method takes all substitutions on the path to the task node into account, in our case the mapping for the isomorphism $h \mapsto \lambda x\textbf{.}x\bar{+}\bar{3}_4$ introduced in step 2. With this instantiation the method can perform simplifications of the resulting equation with respect to modulo arithmetic (see also Sec. 2). This results in the following task at the fringe of the tree:

$$\begin{tikzcd}[row sep=small] \mathsf{Node}\begin{pmatrix}\Sigma \rhd h(x\bar{+}y) = (h(x)\bar{+}h(y))\bar{+}\bar{1}_4, \\ \mathsf{SimplifyRes}(\emptyset, h \mapsto \lambda x\textbf{.}x\bar{+}\bar{3}_4)\end{pmatrix} \arrow[d, dash] \\ \mathsf{Open}(\Sigma \rhd (x\bar{+}y)\bar{+}\bar{3}_4 = (x\bar{+}y)\bar{+}\bar{3}_4) \end{tikzcd}$$

Since the application of $\mathsf{SimplifyRes}$ depends on the substitution $h \mapsto \lambda x\textbf{.}x\bar{+}\bar{3}_4$ this dependency is explictly recorded at the action.

Step 7: The final task holds because equality is reflexive. This step does not produce new subtasks.

$$\begin{tikzcd}[row sep=small] \vdots \arrow[d, dash] \\ \mathsf{Node}(\Sigma \rhd (x\bar{+}y)\bar{+}\bar{3}_4 = (x\bar{+}y)\bar{+}\bar{3}_4, \mathsf{Reflexive}(\emptyset,\emptyset)) \end{tikzcd}$$

5 Validation of Abstract Level Proofs

A key objective of our work is to distinguish better between BL validation of mathematical proofs and their structural, human-oriented development at the AL. However, strong mediation support between the AL and the BL is crucial for our approach. At the AL we *can* work with actions in the range from completely safe ones to very unsafe ones. Our approach particularly supports the flexible combination of such actions — including free oracle proof sketches and faulty proof attempts resulting from the use of unsafe actions. However, in order to classify the AL proof as sound, *all* proof steps at the AL have to be successfully expandable and verifiable at the BL. The *expansion distance* of an AL proof from a verifiable BL proof (i.e., how "far away" an AL proof from a verifiable BL proof is) depends on (1) the actions used at the AL as well as (2) on the chosen BL.

(1): For instance, the ContRew and Decomp steps used in our example are rather safe and easy to expand to (almost any) BL. When complex AL actions are based on complex algorithms, e.g., when they use Computer Algebra Systems to perform numerical simplifications such as in the SimplifyRes steps of the example, then the expansion mappings can become quite complex (ΩMEGA uses special expansion tools such as TRAMP [11] and SAPPER [18]). Human constructed oracle proof steps at the AL are often hard to expand since they become itself again new proof problems (which, however, can be automatically supported as discussed in [16]).

(2): In ΩMEGA the BL consists of a higher-order natural deduction calculus and the distance from abstract proof plans via expansion to this particular BL is in many of our case studies very huge; e.g., as reported in [16] the expansion of the AL island proof plan for the *Irrationality of* $\sqrt{2}$ consists of only *24* proof nodes, while its fully expanded counterpart contains *282* proof nodes. Such an complexity at the BL overwhelms the cognitive resources of a human and sometimes even the technical resources of ΩMEGA. In ΩMEGA^{CORE} we employ Autexier's CORE system [1] at BL as a sound and complete logic engine for classical higher-order logic. Due to this move from higher-order natural deduction calculus to CORE we are able to drastically reduce the expansion distance from the AL to the BL in ΩMEGA^{CORE}. For instance, expansion of the AL reasoning step 3 maps one-to-one to a CORE proof manipulation step in ΩMEGA^{CORE}. In contrast, it expands in long derivations employing Leibniz equality in the old ΩMEGA's natural deduction BL.

Exchanging the BL by a "better one" should ideally not affect the tools provided at the AL. In our approach we therefore consider the BL and BL expansion tools as parameters. Ideally user interfaces, external reasoning specialists, and other support tools mainly operate on the AL such that they are only weakly affected by modifications of these parameters.[3] Such a separation is achieved in ΩMEGA^{CORE} due to the TI presented in this paper.

6 Conclusion

In this paper we have (i) at a conceptual level argued for a top-down approach in PAs and (ii) at a concrete level introduced and illustrated the TI as an AL proof development framework applied in the new ΩMEGA^{CORE} system. We furthermore briefly discussed expansion and validation of TI proofs in CORE, which is our new soundness-guaranteeing BL of choice. Our TI supports a conceptually cleaner integration of support tools (such as external specialist reasoners, graphical user interfaces, proof explanation tools, etc.) at an appropriate level in our hierarchical approach; improved re-usability is just one benefit we gain. We have also introduced or-branches in or TI proof language as, for instance, required in agent-oriented approaches to theorem proving where different alternatives may be tackled simultaneously [18].

The evaluation of our approach is ongoing work: (i) the empirical data we obtain from the DIALOG project is used to evaluate our TI with respect to its suitability to support tutorial natural language proof developments and the first corpus gained in DIALOG [20] is successfully supported by our TI, (ii) the replay of proofs from the mathematical textbook [8] has just started but is already very promising, (iii) a case study on mixed initiative reasoning is future work to evaluate our TI with respect to its capability to interleave interactive and automated proof development.

[3] External reasoners, for instance, may also be employed as support tools for the expansion task as reported in [2]. This kind of usage would then be highly affected by changes in the BL.

REFERENCES

[1] S. Autexier, *Hierarchical Contextual Reasoning*, Ph.D. dissertation, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2003.

[2] C. Benzmüller, M. Bishop, and V. Sorge, 'Integrating TPS and OMEGA', *Journal of Universal Computer Science*, **5**, 188–207, (1999).

[3] C. Benzmüller et al, 'Tutorial dialogs on mathematical proofs', in *Proc. of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, Acapulco, Mexico, (2003).

[4] C. Benzmüller et al, 'Proof planning: A fresh start?', in *Workshop on Future Directions in Automated Reasoning (W1) on IJCAR 2001*, ed., M. Kerber, (2001).

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

[6] A. Bundy, 'The use of explicit plans to guide inductive proofs', in *Proc. of CADE–9*, eds., E. Lusk and R. Overbeek, number 310 in LNCS, pp. 111–120, Argonne, Illinois, USA, (1988). Springer.

[7] A. Bundy, 'A critique of proof planning', in *Computational Logic: Logic Programming and Beyond*, number 2408 in LNCS, 160–177, Springer, (2002).

[8] J. A. Dieudonné, *Foundations of Modern Analysis*, volume 10-I of *Pure and Applied Mathematics*, Academic Press, New York and London, 1969.

[9] M. Gordon, R. Milner, and C. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, number 78 in LNCS, Springer, 1979.

[10] M. Hübner et al, 'Interactive proof construction at the task level', in *Proc. of the Workshop User Interfaces for Theorem Provers (UITP 2003)*, Rome, Italy, (2003).

[11] A Meier, 'TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level', in *Proc. of CADE–17*, ed., D. McAllester, volume 1831 of *LNAI*, pp. 460–464, Pittsburgh, USA, (2000). Springer.

[12] A. Meier, *Proof Planning with Multiple Strategies*, Ph.D. dissertation, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2003.

[13] A. Meier, M. Pollet, and V. Sorge, 'Comparing Approaches to the Exploration of the Domain of Residue Classes', *JSC Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, **34**(4), 287–306, (2002).

[14] E. Melis and J. Siekmann, 'Knowledge-based proof planning', *Artificial Intelligence*, **115**(1), 65–105, (1999).

[15] S. Owre et al, 'PVS: Combining specification, proof checking, and model checking', in *Computer-Aided Verification, CAV '96*, eds., R. Alur and T. Henzinger, number 1102 in LNCS, pp. 411–414, New Brunswick, NJ, (1996). Springer.

[16] J. Siekmann et al, *Proof Development in OMEGA: The Irrationality of Square Root of 2*, 271–314, Kluwer Applied Logic series (28), Kluwer Academic Publishers, 2003.

[17] K. Slind, M. Gordon, R. Boulton, and A. Bundy, 'System description: An interface between CLAM and HOL', *LNCS*, **1421**, (1998).

[18] V. Sorge, *OANTS — A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*, Ph.D. dissertation, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

[19] C.-P. Wirth, 'Descente infinie + Deduction', *96 pp., accepted by Logic Journal of the IGPL*, (2004).

[20] M. Wolska et al, 'An annotated corpus of tutorial dialogs on mathematical theorem proving', in *Proc. of International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Potugal, (2004). To appear.

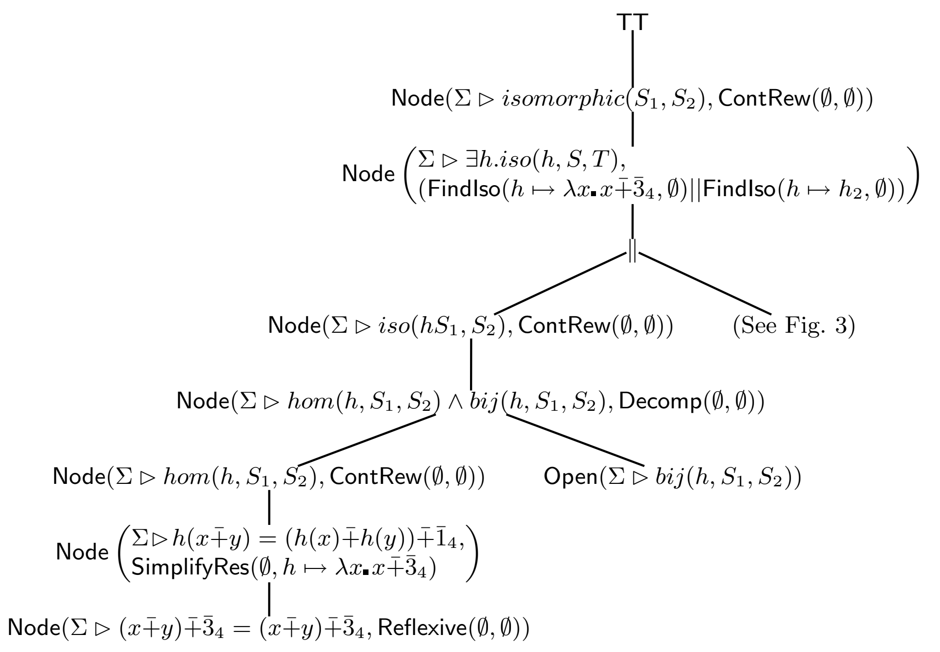
TT
$\mathsf{Node}(\Sigma \rhd \mathit{isomorphic}(S_1, S_2), \mathsf{ContRew}(\emptyset, \emptyset))$
$\mathsf{Node}\begin{pmatrix}\Sigma \rhd \exists h.\mathit{iso}(h, S, T), \\ (\mathsf{FindIso}(h \mapsto \lambda x. x \bar{+} \bar{3}_4, \emptyset) \| \mathsf{FindIso}(h \mapsto h_2, \emptyset))\end{pmatrix}$
$\|$
$\mathsf{Node}(\Sigma \rhd \mathit{iso}(hS_1, S_2), \mathsf{ContRew}(\emptyset, \emptyset))$
(See Fig. 3)
$\mathsf{Node}(\Sigma \rhd \mathit{hom}(h, S_1, S_2) \wedge \mathit{bij}(h, S_1, S_2), \mathsf{Decomp}(\emptyset, \emptyset))$
$\mathsf{Node}(\Sigma \rhd \mathit{hom}(h, S_1, S_2), \mathsf{ContRew}(\emptyset, \emptyset))$
$\mathsf{Open}(\Sigma \rhd \mathit{bij}(h, S_1, S_2))$
$\mathsf{Node}\begin{pmatrix}\Sigma \rhd h(x \bar{+} y) = (h(x) \bar{+} h(y)) \bar{+} \bar{1}_4, \\ \mathsf{SimplifyRes}(\emptyset, h \mapsto \lambda x. x \bar{+} \bar{3}_4)\end{pmatrix}$
$\mathsf{Node}(\Sigma \rhd (x \bar{+} y) \bar{+} \bar{3}_4 = (x \bar{+} y) \bar{+} \bar{3}_4, \mathsf{Reflexive}(\emptyset, \emptyset))$

Figure 2. The complete task tree for the example from Section 4 the right hand side of the or-branch is continued on the next figure.

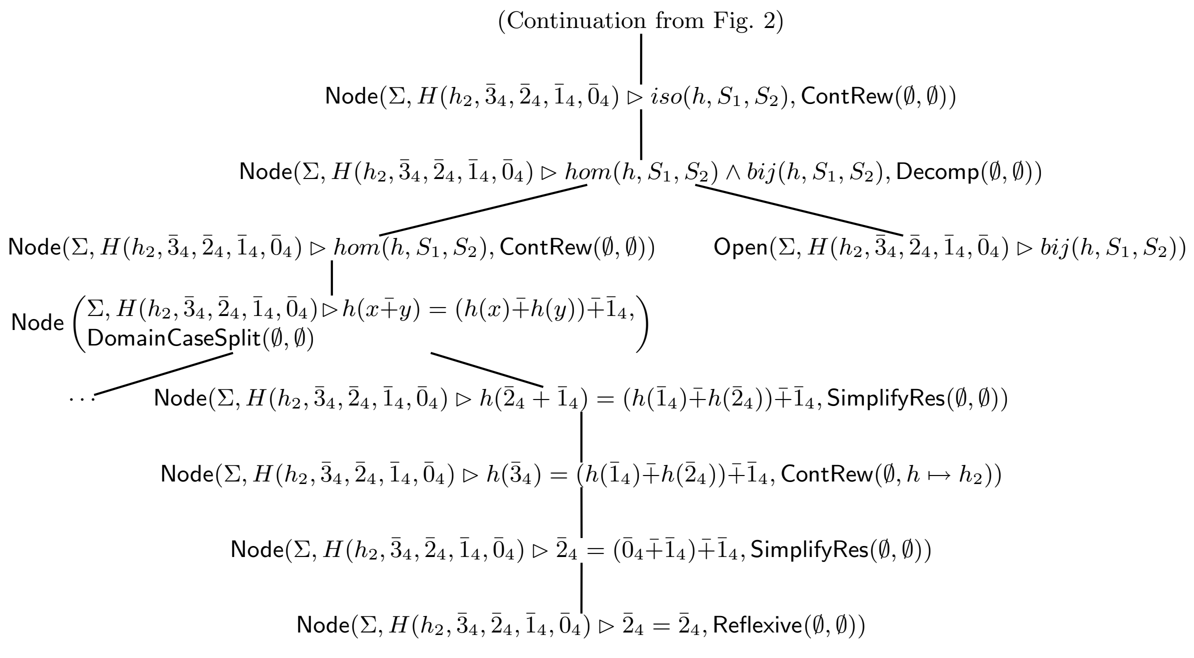
(Continuation from Fig. 2)
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \mathit{iso}(h, S_1, S_2), \mathsf{ContRew}(\emptyset, \emptyset))$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \mathit{hom}(h, S_1, S_2) \wedge \mathit{bij}(h, S_1, S_2), \mathsf{Decomp}(\emptyset, \emptyset))$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \mathit{hom}(h, S_1, S_2), \mathsf{ContRew}(\emptyset, \emptyset))$
$\mathsf{Open}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \mathit{bij}(h, S_1, S_2))$
$\mathsf{Node}\begin{pmatrix}\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd h(x \bar{+} y) = (h(x) \bar{+} h(y)) \bar{+} \bar{1}_4, \\ \mathsf{DomainCaseSplit}(\emptyset, \emptyset)\end{pmatrix}$
$\ldots$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd h(\bar{2}_4 \bar{+} \bar{1}_4) = (h(\bar{1}_4) \bar{+} h(\bar{2}_4)) \bar{+} \bar{1}_4, \mathsf{SimplifyRes}(\emptyset, \emptyset))$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd h(\bar{3}_4) = (h(\bar{1}_4) \bar{+} h(\bar{2}_4)) \bar{+} \bar{1}_4, \mathsf{ContRew}(\emptyset, h \mapsto h_2))$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \bar{2}_4 = (\bar{0}_4 \bar{+} \bar{1}_4) \bar{+} \bar{1}_4, \mathsf{SimplifyRes}(\emptyset, \emptyset))$
$\mathsf{Node}(\Sigma, H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4) \rhd \bar{2}_4 = \bar{2}_4, \mathsf{Reflexive}(\emptyset, \emptyset))$

Figure 3. An alternative proof. $H(h_2, \bar{3}_4, \bar{2}_4, \bar{1}_4, \bar{0}_4)$ is an abbreviation for $h_2(\bar{0}_4) = \bar{3}_4$, $h_2(\bar{1}_4) = \bar{2}_4$, $h_2(\bar{2}_4) = \bar{1}_4$, $h_2(\bar{3}_4) = \bar{0}_4$, this representation of the pointwise defined function is introduces as hypothesis by FindIso. The method DomainCaseSplit introduces a case split over the $4 \times 4 = 16$ possible instantiations for x and y, which have to be checked. The other 15 cases are omitted.