A Generic Modular Data Structure for Proof Attempts Alternating on Ideas and Granularity

Serge Autexier[1,2], Christoph Benzmüller[1], Dominik Dietrich[1], Andreas Meier[2], and Claus-Peter Wirth[1]

[1] FR Informatik, Saarland University, Saarbrücken, Germany
{autexier, chris, dodi, cp}@ags.uni-sb.de
[2] DFKI GmbH, Stuhlsatzenhausweg, Saarbrücken, Germany
ameier@dfki.de

Abstract. A practically useful mathematical assistant system requires the sophisticated combination of interaction and automation. Central in such a system is the proof data structure, which has to maintain the current proof state and which has to allow the flexible interplay of various components including the human user. We describe a parameterized proof data structure for the management of proofs, which includes our experience with the development of two proof assistants. It supports and bridges the gap between abstract level proof explanation and low-level proof verification. The proof data structure enables, in particular, the flexible handling of lemmas, the maintenance of different proof alternatives, and the representation of different granularities of proof attempts.

1 Introduction

A careful and objective inspector of the history of automated theorem proving in the last fifty years would come to the following hypothesis:

> Stand-alone automated theorem provers will never develop into practically useful mathematical assistant systems.

To achieve the original design goal of a practically useful mathematical assistant system, we aim at *interactive* systems with a high degree of automated support. To combine interaction and automation into a synergetic interplay and to bridge between abstract level proof explanation and low-level proof verification is an enormous task. It requires sophisticated achievements from logic, tactics programming, proof planning, agent-based architectures, graphical user interfaces, and integration of other reasoning tools on the one hand, and a deeper experience in informal and formal human proof construction on the other hand.

The main task of the proof data structure in the center of such a system is to maintain the current states of the proof attempts with their open goals and available lemmas. To further the communication between a theorem proving system on the one hand and a human user, another system, or a proof archive on the other hand, an appropriate representation and transformation of proof

M. Kohlhase (Ed.): MKM 2005, LNAI 3863, pp. 126–142, 2006.

attempts is necessary. In this paper we describe a new proof data structure (*PDS*) for this purpose. It generalizes both the existing PDS (with its features for granularity) of the ΩMEGA system [8, 16, 5] and the proof forests (with their alternative proof attempts and lemmatization) of the QUODLIBET system [3] and incorporates the experience gained in the last dozen years.

The basic ideas are:

- Each conjectured *lemma* gets its own *proof tree* (actually a directed acyclic graph (dag)).
- In this *proof forest*, each lemma can be applied in each proof tree; either as a lemma in the narrower sense, or as an induction hypothesis in a possibly mutual induction process, see [18].
- Inside its own tree, the lemma is a goal to be proved reductively. A *reduction* step reduces a *goal* to a conjunction of *sub-goals* w.r.t. a *justification*.
- Several reduction steps applied to the same goal result in alternative proof attempts, which either represent different proof *ideas* or the same proof idea with different *granularity* (or detailedness).

Although the application of a lemma of one tree (*generative* step) results in a *reductive* step inside another tree, we do not overemphasize reduction by this:

- For purely generative abstract theory expansion we may assume some trivial reductions, which can later be refined to the reductions that will be necessarily involved in this generation on the concrete level of a logical calculus.
- All steps in a traditional sequent or tableau calculus as well as *backward* and *forward* steps in Natural Deduction can be realized as reduction steps.

A parallel representation of *different granularities* of proof attempts is necessary for increasing granularity from proof sketches to the actual elaboration of the concrete proofs, and for decreasing the granularity from huge automatically generated proofs to tactical descriptions of a size that can be stored and archived. Moreover, the inspection of proof attempts by human users requires different granularities and the possibility to switch between them for size management and modular focusing according to their varying intentions and different expertise. An important new feature compared to the existing PDS of the ΩMEGA system is that granularity does not have to be linearly ordered: there may be two incomparable subtrees that both represent a more fine-grained version of a reduction step. Note that we do not have well-defined *levels* of granularity because we have no means (yet?) to define such levels from our experience, neither as mathematicians nor as theorem-proving engineers.

Our novel data structure is *generic* insofar as it is parameterized in both the *justifications* of the reductions (ranging from tentative hopes based on insecure knowledge to inference steps in a formal logic calculus) and in the data type of the *goals*, which may reach from sentences in natural languages to the *proof task* data structure of the CORE system [10]. Note, however, that we cannot distinguish yet between *levels* of abstraction realized by different data types for goals. For example, we do not distinguish between different levels of abstraction

in the language of our goals and have no means for signature morphisms at the level of our PDS.

Although the above-mentioned tasks of the CORE system are not the subject of this paper, they may help (in form of a concrete instance) to describe the form of our novel PDS: Roughly speaking, a *task* is simply a disjunctive list of formulas (i.e. the simplest form of a sequent in classical logic) with some augmentations for different purposes, such as—among others—a distinction on one formula as the *focus*, rendering the conjugates of the other formulas as *context* formulas to be assumed when reasoning on the focus, such as a weight term for generating the ordering constraints in applications of induction hypotheses, and such as colorings for heuristic guidance.

The paper is structured as follows. We start in Section 2 with a brief summary of the old data structures of the ΩMEGA and the QUODLIBET systems and motivate their unification and generalization in the new data structure. Section 3 provides a formal description of the new generic proof data structure. Its usage is illustrated in Section 4 by a sample proof development. In Section 5 we give our answer to the question on fundamental design alternatives and Section 6 concludes the paper.

2 ΩMEGA's and QUODLIBET's Old Proof Data Structures

ΩMEGA's Proof Data Structure. ΩMEGA (see [16] for an overview and a list of further literature) is a mathematical assistant tool that supports proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system in which supplementary subsystems are placed around a central proof data structure (PDS) such that the subsystems can work together to construct a proof whose status is stored in the PDS. The facilities provided by the subsystems include support for interactive and mixed-initiative theorem proving incorporating the user, proof planning, access to external systems such as automated theorem provers and computer algebra systems, and proof expansion to and proof checking at the basic level of an underlying logic calculus (which, however, is of no interest to the human user of ΩMEGA). These facilities require, in particular, the representation of proof steps at different granularities ranging from abstract human-oriented justifications to logic-level justifications.

Technically speaking, the old PDS [5] is a dag consisting of nodes, justifications and hierarchical edges. Each node represents a sequent and can be open or closed. An open node corresponds to a sequent that is to be proved and a closed node to a sequent which is already proved or reduced to other sequents using an inference rule $R := \frac{A_1 \ldots A_k}{B}$. Such a rule says that from $A_1, \ldots, A_k$ we can conclude B or reading it the other way round that B can be reduced to $A_1, \ldots A_k$. Such an inference step is represented by a justification which connects sequents $A_1, \ldots A_k$ stored in nodes $n_1, \ldots n_k$ with a node n_b containing B. If a node has more than one outgoing justification, each of them represents a proof attempt of the sequent stored in the source node, but at different granularity. These have to be ordered with respect to their granularity using hierarchical edges.

Fig. 1. Possible views of proofs at different granularities inside a PDS

A hierarchical edge connects two justifications j_1 and j_2 with the meaning that justification j_1 represents a more detailed proof attempt than justification j_2.

If proofs of different granularity are linked together by hierarchical edges, the user normally just wants to see one proof at a specific granularity. By selecting the granularity for each node he gets a view onto the graph, called PDS-view.

An example is given in Fig. 1: It shows a node n which has two outgoing justifications j_1 and j_2, which are connected by an hierarchical edge from j_1 to j_2. The user can decide whether to see the more detailed version of the proof given by j_1 (and its subtree t_1) or the more abstract version given by j_2 (and its subtree t_2). The different possible views are indicated by shading the respective nodes and justifications.

QuodLibet's Proof Data Structure. Although ΩMEGA's old PDS can represent proofs at different granularity within one data structure, it still has some weaknesses compared to QuodLibet's PDS:

- Alternative proof steps cannot be represented. That is, it is not possible for the user to tackle different proof ideas in parallel within the same proof data structure. This holds for both the reduction of a goal to some sub-goals as well as for the expansion of a complex proof step to a lower granularity. For both cases different alternatives may exist whose parallel inspection should be supported.
- An explicit handling of lemmas is not supported by ΩMEGA's old PDS. That is, it is one monolithic dag and lemmas cannot be maintained in separated DAGs.

QuodLibet [3] is a tactic-based inductive theorem proving system for first-order clauses. It does not pursue the push-bottom technology for inductive theorem proving, but it manages more complicated proofs by an effective interplay between interaction and automation. Basically, the system does all the routine work and asks the user as early as possible if intelligence or semantic knowledge is needed. QuodLibet has been applied mostly successfully to nontrivial mathematical research, e.g. the comparison of different formalizations of the lexicographic path ordering and their properties.

The difference compared to the new PDS of the following sections is that the proof forests of QuodLibet consist of real trees instead of dags and there are no means for changing granularity. Although the user interface admits powerful tactic programming, the proofs are always represented on the calculus level. A decade ago, this seemed reasonable: The calculus was carefully developed over

years of practical evaluation to meet the requirement of being as human-oriented as possible, some of its inference steps would take ten to a hundred steps of other calculi implemented for inductive theorem proving, and the system programmer's interface admits the addition of new inference rules for further coarse grain inference steps, such as computation and decision procedures. In the current improvement phase, however, it became obvious that the system's restriction to the finest grain is a problem growing with the power of the system, and that we need the possibility for vast changes in granularity in the proof data structure.

3 Generic Proof Data Structure

The described features of the proof data structures successful in ΩMEGA and QUODLIBET are obviously orthogonal. Their combination and further generalization, including a relaxation of the granularity restrictions—following the guidelines of Section 1—result in a new proof data structure whose features exceed the features of its origins. In particular, the new data structure supports:

- the representation of alternative proof steps for *both* the reduction of a goal as well as for the expansion of a complex proof step to lower granularity
- the structuring of proof parts (i.e. lemmatization) into separate but connected parts of the data structure
- the generic representation of proof statements and justifications, biased neither to any specific calculus nor to any specific formalism for representing abstract proof plans.

In the remainder of the section, we give a formal definition of the new generic proof data structure. We start with a formal definition of the basic PDS. We then formally define PDS-views and finally we extend a single PDS to conglomerates of PDSs, so-called forests. Note that the major technical challenge to devise a mathematically sound formulation was to consistently integrate alternative proof steps for both alternatives for the reduction of goals as well as alternatives for the expansion of a complex proof step to a lower granularity.

3.1 The PDS

Our basic PDS essentially is a directed acyclic graph (dag) whose nodes contain the proof statements. The representation to be chosen for the latter is by no means constrained in our framework. The PDS has two sorts of links: *justification hyper-links* describe a relation of goal nodes to their sub-goal nodes, and *hierarchical edges* point from justifications to other justifications they refine.

Definition 1 (PDS). A PDS is composed of nodes, justifications and hierarchical edges. Each such component x of a PDS is labeled with a pair $\mathsf{label}(x) = (c, t)$, where c maintains arbitrary content and $t \in \mathbb{N}$ is a timestamp. The time information enables us to define an order in which the objects have been created. The content of the labels can be freely instantiated, for instance, with proof

statements in the case of proof nodes or with names of proof rules, tactics, and methods in the case of justifications. That is, our approach is parameterized over this sort of information that is typically very specific to different proof assistants. Formally, a PDS is defined as a triple $P := \langle \mathcal{N}, \mathcal{J}, \mathcal{H} \rangle$ where

- $\mathcal{N}$ is a nonempty finite set of *nodes*. Each node $n \in \mathcal{N}$ has a label l, denoted as $\mathsf{label}(n)$.
- $\mathcal{J}$ is a finite set of *justifications*. Each justification $j \in \mathcal{J}$ is a triple (s, T, l). $s \in \mathcal{N}$, $T \subseteq \mathcal{N}$, and l specify the *source*, the *targets*, and the *label* of j. They are denoted as $\mathsf{source}(j)$, $\mathsf{targets}(j)$, and $\mathsf{label}(j)$, respectively. We will also denote justifications as $s \xrightarrow{l} T$. Generally, a justification $s \xrightarrow{l} T$ represents a *proof step* in which proof node s is reduced to the nodes T by application of the operator l. For each node $n \in \mathcal{N}$, we define the set of *incoming justifications* by $I_n := \{j \in \mathcal{J} | n \in \mathsf{targets}(j)\}$, and the set of *outgoing justifications* by $O_n := \{j \in \mathcal{J} | \mathsf{source}(j) = n\}$. The *graph* of $\mathcal{J}$ is $\{(\mathsf{source}(j), n) | j \in \mathcal{J} \ \wedge \ n \in \mathsf{targets}(j)\}$; we require it to be acyclic.
- We require that there exists exactly one node $n_r \in \mathcal{N}$ with $I_{n_r} = \emptyset$, called the *root node*.
- $\mathcal{H}$ is a finite set of *hierarchical edges* on $\mathcal{J}$. Each hierarchical edge $h \in \mathcal{H}$ is a triple (j_1, j_2, l). $j_1 \in \mathcal{J}$, $j_2 \in \mathcal{J}$, and l specify the *source*, the *target* and the *label* of h. They are denoted as $\mathsf{source}(h)$, $\mathsf{target}(h)$, and $\mathsf{label}(h)$, respectively. We will denote hierarchical edges also as $j_1 \overset{h}{\rightarrow}_l j_2$. The *graph* of $\mathcal{H}$ is defined as the set of pairs $\{(\mathsf{source}(h), \mathsf{target}(h)) | h \in \mathcal{H}\}$; we require it to be acyclic. For all hierarchical edges $j_1 \overset{h}{\rightarrow}_l j_2$ we require:
  - $\mathsf{source}(j_1) = \mathsf{source}(j_2)$ (i.e. hierarchical edges may only connect justifications sharing the same source node), and
  - for each $n_2 \in \mathsf{targets}(j_2)$ there exists an $n_1 \in \mathsf{targets}(j_1)$ such that (n_1, n_2) is in the reflexive and transitive closure of the graph of $\mathcal{J}$ (i.e. j_1 is the first proof step of a derivation that refines the proof step characterized by j_2).

As opposed to ΩMEGA's old PDS, this definition supports alternative justifications and alternative hierarchical edges. In particular, several outgoing justifications of a node n, which are not connected by hierarchical edges, are OR-alternatives. That is, to prove a node n, only the targets of one of these justifications have to be solved. Hence they represent alternative ways to tackle the same problem n. This describes the horizontal structure of a proof. Note further that we allow sharing of refinements; i.e., two abstract justifications may be refined by one and the same justification at lower levels. Sharing justifications in refinements is motivated, for instance, as follows: Consider a justification j which represents the call to an external system that generates a set of n different solutions, all represented in a single successor node of j with outgoing alternative subproofs starting with $j_1 \ldots j_n$, one for each solution. Then, for any $i \in \{1, \ldots, n\}$, we may abstract a coarse-grain justification a_i corresponding to a path starting with $\langle j, j_i \rangle$, represented by a hierarchical edge from j to a_i. Not supporting the sharing of justifications would in this scenario require the duplication of the justification j, which is both cumbersome and not adequate.

From the problem-solving point of view we need to know if a problem—including all its related subproblems—has already been solved or which subproblems still need to be solved. We introduce the following terminology to distinguish the different situations:

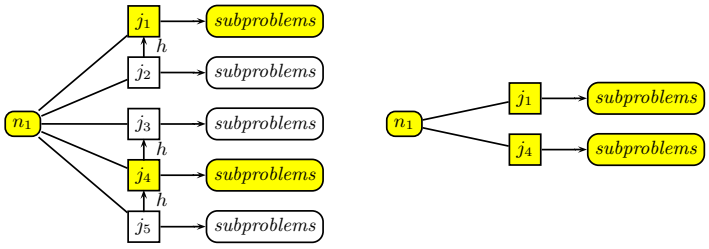
Definition 2 (Open/Closed Nodes). *Let $P = \langle \mathcal{N}, \mathcal{J}, \mathcal{H} \rangle$ be a PDS and $n \in \mathcal{N}$ be a node of P. n is called* locally closed *if and only if there exists a $j \in J$ with $\mathsf{source}(j) = n$ and $\mathsf{target}(j) = \emptyset$; i.e. n is justified without reducing it to new subproblems. n is called* tree-wide closed *if it is locally closed or if there is a $j \in O_n$ such that all $m \in \mathsf{targets}(j)$ are tree-wide closed. The latter says that n is justified by a reduction to subproblems $m \in \mathsf{targets}(j)$ which are all already (recursively tree-wide) closed. A node is called* locally/tree-wide open *if it is not locally/tree-wide closed.* □

3.2 PDS-View

Hierarchical edges construct the vertical structure of a proof. They distinguish between upper layer proof steps and related derivations which refine them at a more granular layer. This mechanism supports both recursive expansion and abstraction of proofs. A proof may be conceived at a high level of abstraction and then *expanded* to a finer grain. As opposed thereto, *abstraction* means the process of successively contracting fine-grain proof steps to more abstract proof steps.[1] Furthermore, the PDS generally supports alternative and mutually incomparable refinements of one and the same upper layer proof step. This horizontal structuring mechanism—together with the possibility to represent OR-alternatives at the vertical level—provides very rich and powerful means to represent and maintain proof attempts. In fact, such multidimensional proof attempts may easily become too complex for humans to keep an overview as a whole. In particular, since a human does not have to work simultaneously on different granularities of a proof, elaborate functionalities to access only selected parts of a PDS are useful. They are required, for instance, for user-oriented presentation of a PDS, in which the user should be able to focus on the parts of the PDS he is currently working at, while being always able to choose whether he wants to see more details for some proof step or, on the contrary, needs to be shown a coarse structure when he is lost in the details.

We define in this subsection the notion of a *PDS-view*. A PDS-view extracts from a given PDS only a horizontal structure of the represented proof attempt at chosen granularities, but with all its OR-alternatives. As an example consider the PDS fragments in Fig. 2. In the fragment on the left-hand side, the node n_1 has two alternative proof attempts and each at alternative granularities. The

[1] An application of recursive expansion in the ΩMEGA system is, for instance, proof planning [14]. Proof planning first establishes a proof at an abstract level. Afterwards, to be proof checked, this proof plan may have to be expanded to a (very granular) underlying calculus. An application of recursive abstraction in ΩMEGA is, for instance, the abstraction of Natural Deduction proofs to assertion level proofs which are better suited for presentation [9].



(a) (b)

Fig. 2. (a) PDS-node with all outgoing partially hierarchically ordered justifications, and j_1, j_4 in the set of alternatives. Justifications are depicted as boxes. (b) PDS-node in the PDS-view obtained for the selected set of alternatives j_1, j_4.

fragment on the right-hand side gives a PDS-view which results by selecting a certain granularity for each alternative proof attempt, respectively. The sets of alternatives may be selected by the user and define the granularity on which he currently wants to inspect the proof. The resulting PDS-view is a slice plane through the hierarchical PDS and is—from a technical point of view—also a PDS, but without hierarchies, i.e. without hierarchical edges.

In the remainder of this subsection, we give a formal definition of a PDS-view. First, we introduce some technical prerequisites.

Definition 3 ($\mathcal{H}$-Induced Orderings $<$ and $\leq$). *Given a PDS $S = \langle \mathcal{N}, \mathcal{J}, \mathcal{H} \rangle$ we define $<$ to be the transitive closure of the graph of $\mathcal{H}$ and $\leq$ to be the reflexive closure of $<$.*

Note that $<$ and $>$ are well-founded orderings because the graph of $\mathcal{H}$ is acyclic and finite.

A PDS-node can have multiple outgoing justifications, representing alternative proof attempts or proofs at different granularity. During the proof construction or presentation, we want to restrict this set of justifications to get a complete set of alternatives at some specific granularity:

Definition 4 (Set of Alternatives). *Let $\langle \mathcal{N}, \mathcal{J}, \mathcal{H} \rangle$ be a PDS, $n \in \mathcal{N}$, and $A \subseteq O_n$ a set of justifications for n.*

- *A is* adequate *if there are no $k, k' \in A$ such that $k<k'$.*
- *A is* complete *if for all $k \in O_n$ there is a $k' \in A$ such that $k \leq k'$ or $k' \leq k$.*

A is a set of alternatives for *n if it is adequate and complete. Given $j_1, j_2 \in O_n$, j_1 and j_2 are* comparable*, if $j_1 \leq j_2$ or $j_2 \leq j_1$; otherwise they are* not comparable*.*

The adequacy property ensures that at most one descendant is selected for each alternative, whereas the completeness property says that there must be at least one.

For instance, the node n_1 on the left-hand side of Fig. 2 has five outgoing justifications. $\leq$ splits these justifications into 2 classes: $\{j_1, j_2\}$ and $\{j_3, j_4, j_5\}$,

where the elements of a class represent the same proof alternative but at a different granularity. A set of alternatives for n is, for instance, $\{j_1, j_4\}$.

Definition 5 (PDS-View). *Let $P := \langle \mathcal{N}, \mathcal{J}, \mathcal{H} \rangle$ be a PDS, $\mathcal{N} = \{n_1, \ldots, n_m\}$ and let $A_{n_1}, \ldots, A_{n_m}$ be sets of alternatives for the nodes $n_1, \ldots, n_m$ respectively. A* PDS-view *is a PDS $\langle \mathcal{N}', \mathcal{J}', \emptyset \rangle$ such that $\mathcal{N}' \subset \mathcal{N}$ and $\mathcal{J}' \subset \mathcal{J}$ are the smallest sets with: (1) The root node n_r of P is in $\mathcal{N}'$, (2) if $n \in \mathcal{N}'$ then $A_n \subseteq \mathcal{J}'$, and (3) if $s \xrightarrow{l} T \in \mathcal{J}'$ then $T \subseteq \mathcal{N}'$.*

From a procedural point of view the computation of a PDS-view is a recursive process that starts at the root node n_r of a given PDS. For each node contained in the PDS-view, its set of alternatives are introduced as justifications of the PDS-view. The target nodes of these introduced justifications are then added to the nodes of the PDS-view etc. As an example consider the PDS-node and justifications on the right-hand side of Fig. 2 which are the corresponding part of the PDS-node on the left-hand side in the PDS-view.

Note that this definition of a PDS-view is not the only possible one. An alternative would be, for instance, to choose not only among hierarchical alternatives but also among the vertical alternatives. The result would be a PDS-view that contains neither hierarchical nor vertical alternatives.

3.3 Forests

We now extend the structure of a single PDS to a so-called *PDS-forest*, i.e. a set of PDSs which can be interdependent, indicated by special links between their graphs. The intuition is as follows: to prove a conjecture, further axioms, lemmas and theorems—uniformly called *lemmas*—can be used. The lemmas are either already proved or have been synthesized during proof search and are not yet proved. To accommodate either situation, new PDSs which live in the same forest as the PDS of the conjecture are introduced for the lemmas. A proof step in some PDS can then be justified by a lemma by linking the justification link to the root node the PDS for that lemma.

Definition 6 (Forest). *Let $\mathcal{I}$ be an index set. A forest is a pair $\langle (PDS_i)_{i \in \mathcal{I}}, \mathcal{F} \rangle$ where*

- *$(PDS_i)_{i \in \mathcal{I}} = (\langle \mathcal{N}_i, \mathcal{J}_i, \mathcal{H}_i \rangle)_{i \in \mathcal{I}}$ is a family of disjoint PDSs.*
- *$\mathcal{F}$ is a finite set of* forest edges *between PDSs. Each forest edge $f \in \mathcal{F}$ is a pair (j, n_r) consisting of the source $\mathsf{source}(f) = j$, which is a justification from some $\mathcal{J}_i$, $i \in \mathcal{I}$, and the target $\mathsf{target}(f) = n_r$, which is the root node of some PDS $PDS_{i'}$, $n_r \in \mathcal{N}_{i'}$, $i' \in \mathcal{I}$. We denote a forest link (j, n_r) by $j \dashrightarrow n_r$.*

Given a justification $j \in \mathcal{J}_i$ for some $i \in \mathcal{I}$, the set of outgoing forest links for that justification is denoted by $F_j := \{f \in \mathcal{F} | \mathsf{source}(f) = j\}$. □

Applying a new lemma on some justification j in a PDS p results in introducing a new PDS p' with root node n_r and a forest edge that connects j to n_r. Although

we intuitively apply lemmas to a goal stored in a node, a forest edge starts at a justification of this node. This is necessary to determine in which alternative the lemma is to be applied. The node n is eventually *tree-wide closed* by the justification j, but n remains *forest-wide open* until the n_r in the PDS p' (just as the target nodes $\mathsf{targets}(j)$) are (forest-wide) closed.

Note that forest edges can produce cycles. This allows us to apply a lemma to itself, which is needed to represent induction in the form of *descente infinie* [18]. Moreover, due to our AND-OR proof trees these cycles may refer to different choices of AND proofs.

We now extend the notion of a PDS-view to the notion of a forest.

Definition 7 (Forest-View). *Let $\langle \mathcal{PDS}, \mathcal{F} \rangle$ be a forest. A* forest-view with respect to some $p \in \mathcal{PDS}$ *is a forest $\langle \mathcal{PDS}', \mathcal{F}' \rangle$, such that $\mathcal{PDS}'$ and $\mathcal{F}'$ are the smallest sets with:*

- *The PDS-view for p is in $\mathcal{PDS}'$.*
- *For all justifications j in some PDS-view from $\mathcal{PDS}'$ and for all forest edges $j \dashrightarrow n_r \in \mathcal{F}$, $n_r \in p'$:*
  $j \dashrightarrow n_r$ and the PDS-View for p' are contained in $\mathcal{F}'$ and $\mathcal{PDS}'$, respectively.

4 Sample Application

Our application of the proof data structure presented in this paper within the ΩMEGA project instantiates the framework with so-called *proof tasks*; i.e. they become the nodes of our proof data structure. Tasks were developed originally to represents proof situations in ΩMEGA's proof planner MULTI [13]. We extended tasks as a general technique for "natural" reasoning with abstract steps [10]. That is, the task framework allows for all kinds of steps with tasks ranging from formal steps like rewrite steps or definition expansion/contraction steps to abstract steps involving computations of external systems or merely sketching proof ideas and their flexible combination.

Proof tasks can be seen as sequents $\varphi_1, \ldots \varphi_n \vdash \psi_1, \ldots, \psi_m$ where there is always one formula—the so-called *focus*—annotated as the currently active one. The focus may be an antecedent or a succedent formula. For example, $\varphi_1, \ldots \varphi_n \vdash \underline{\psi_1}, \psi_2, \ldots, \psi_m$ describes a task where we have the context $\varphi_1, \ldots \varphi_n \vdash \psi_2, \ldots, \psi_m$ available for showing the focus $\vdash \psi_1$. In a user-interface we may want to present tasks as

$$\frac{\varphi_1, \ldots \varphi_n}{\underline{\psi_1}, \ldots, \psi_m}$$

and use colors to further distinguish antecedent and succedent formulas, e.g. the negative formulas in red and the positive ones in black.

In the remainder of this section, we discuss the construction of a PDS with tasks for the example theorem "$\sqrt{12}$ is irrational". The general proof technique we shall apply to this problem works as follows: Given is the conjecture "$\sqrt[j]{l}$ is irrational". Assume that $\sqrt[j]{l}$ is rational. Then there are integers n, m, which have

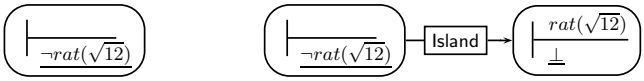


Fig. 3. PDS trees respectively after step 0 and step 1

no common divisor and for which holds that $\sqrt[j]{l} = \frac{n}{m}$. Derive a contradiction to the assumption by showing that, indeed, n, m have a common divisor. Potential candidates for the common divisor are the prime factors of l.

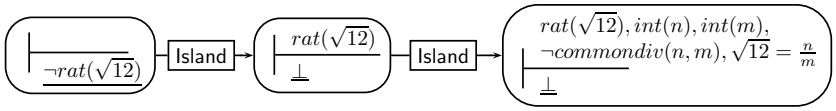
4.1 Proof Construction in a PDS

In a first step, we construct a PDS in the way a human mathematician would like to prove the given conjecture; see the proof sketch above. This is supported by so-called interactive island planning (see [17, 16] for details), a technique that expects an outline of the proof and has the user provide main subgoals, called *islands*. The details of the proof, eventually down to the logic level, are postponed. Hence, the user can write down *his* proof idea in a natural way with as many gaps as there are open at this first stage of the proof. Technically, in our framework the islands are tasks and all justifications between islands state `island`, i.e., they just indicate the intention that an island should follow from several other islands.

Step 0. The proof starts with the initial task $\vdash \underline{\neg rat(\sqrt{12})}$ and the initial PDS show on the left of Fig. 3.

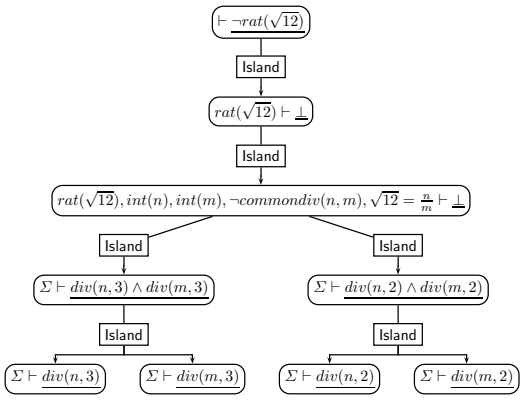
Step 1. In the first step we introduce the indirect argument and reduce the initial task to $rat(\sqrt{12}) \vdash \underline{\bot}$ in which we assume that $rat(\sqrt{12})$ holds and the basic contradiction $\bot$ is to be proved. This action extends the PDS to one viewed on the right of Fig. 3 where the justification Island states that the action introduces a new island node.

Step 2. In the second step we derive from the assumption $rat(\sqrt{12})$ that there exist two integers n, m, which have no common divisors and for which $\sqrt{12} = \frac{n}{m}$ holds. This action further refines the PDS to



with the new task $rat(\sqrt{12}), int(n), int(m), \neg commondiv(n, m), \sqrt{12} = \frac{n}{m} \vdash \bot$.

Step 3 + 4. To complete the proof a common divisor is needed. Since 12 has the prime factors 2 and 3 there are two potential candidates. Moreover, for each candidate we have to show that both n and m are divided by it. This results in the PDS (shown below) with OR-branches (outgoing edges of PDS-nodes, e.g. $\boxed{rat(\sqrt{12}), int(n), int(m), \neg commondiv(n,m), \sqrt{12} = \frac{n}{m} \vdash \underline{\bot}}$) and AND-branching (outgoing links of justification nodes, e.g. $\boxed{\text{Island}}$), where Σ abbreviates all so far available



assumptions: $rat(\sqrt{12})$, $int(n)$, $int(m)$, $\neg commondiv(n,m)$, $\sqrt{12} = \frac{n}{m}$. To accomplish a proof we have now 2 possibilities: either we solve the two tasks $\Sigma \vdash div(n,3)$ and $\Sigma \vdash div(m,3)$, which demand to prove that both n and m have divisor 3, or we solve the two tasks $\Sigma \vdash div(n,2)$ and $\Sigma \vdash div(m,2)$, which demand to prove that both n and m have divisor 2.

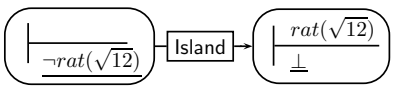
Step 5 + ... We omit the further construction of the PDS in detail and just sketch the missing steps to derive a proof. We cannot show that both n and m have divisor 2 in the given context. Hence, the right branch of the PDS does not represent any progress. However, both n and m have divisor 3. From $\sqrt{12} = \frac{n}{m}$ follows that $m^2 * 12 = n^2$. Hence, n^2 has divisor 12 and thus also divisor 3. Then n also has divisor 3, since 3 is a prime number. This implies that $n = 3 * k$ for an integer k. Substituting n by $3 * k$ in the equation $m^2 * 12 = n^2$ results in $m^2 * 12 = 9 * k^2$. This equation can be simplified to $m^2 * 4 = 3 * k^2$. This implies that m^2 has divisor 3, from which follows that m has divisor 3 since 3 is a prime number. The introduction of all these steps closes the left branch, i.e. one alternative, of the last PDS and results in a closed PDS.

We want to remark that a proof along this idea can also be automatically proof planned in ΩMEGA; for further details we refer to [16].

4.2 Proof Expansion in a PDS

So far, our proof has been developed and sketched only at an intuitive, abstract level and logical details have been neglected. Verification of this proof requires expanding it to a logic-calculus layer. How much "effort" this expansion causes and whether it succeeds depends on the island steps and the gaps they represent. In general, an island step can be arbitrarily difficult, so that each island step may again represent a proof problem in its own right. Nevertheless, the expansion can be supported by automated tools. For instance, automated theorem provers can try to solve subproblems, computer algebra systems can perform computations, and model generators can create counterexamples, which can point out missing facts in the proof. We omit a detailed discussion of automated expansion support here and refer the interested reader to [17] and [16]. Rather, we briefly discuss the expansion of two steps in our current example PDS and sketch the resulting extended and refined PDS.

Expansion 1. Consider the first step in the current PDS, which reduces the task $\vdash \underline{\neg rat(\sqrt{12})}$ to the task $rat(\sqrt{12}) \vdash \underline{\bot}$:



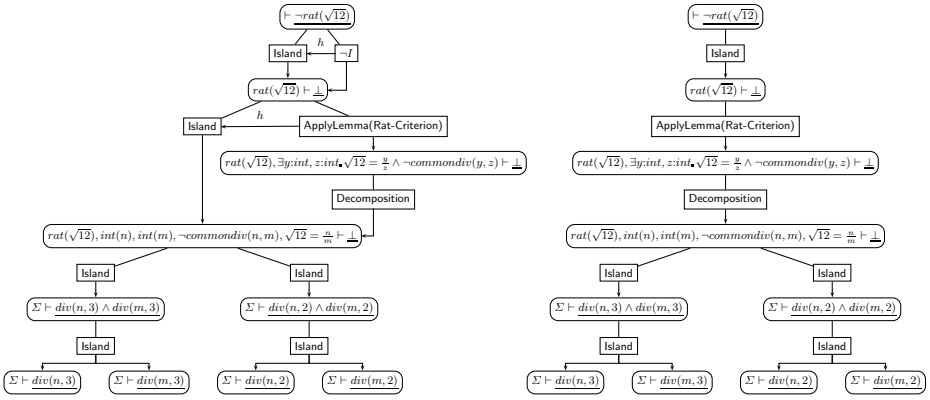
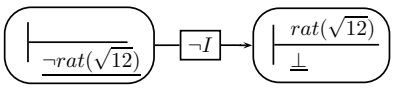
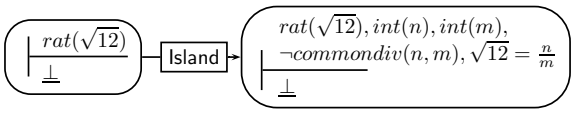


Fig. 4. (Left) Complete PDS for the running example with alternative proof attempts and different layers of granularities. (Right) A possible PDS-View determined by selection of a set of alternatives for each PDS-node in the complete PDS.



This step is already an instance of a proof step on calculus level. Indeed, it is a negation introduction step ($\neg I$). Hence, an expansion of this step simply results in a justification with $\neg I$ deriving $\vdash \underline{\neg rat(\sqrt{12})}$ from task $rat(\sqrt{12}) \vdash \bot$ by a calculus step. The resulting PDS fragment is shown above.



Expansion 2. The second step in the proof reduced the task $rat(\sqrt{12}) \vdash \underline{\bot}$ to the task $rat(\sqrt{12}), int(n), int(m), \neg commondiv(n,m), \sqrt{12} = \frac{n}{m} \vdash \underline{\bot}$, which is represented by the PDS fragment above. This step implicitly encapsulates the application of the theorem that each rational number equals the fraction of two integers that have no common divisor. In the database of ΩMEGA this theorem is called *Rat-Criterion*:

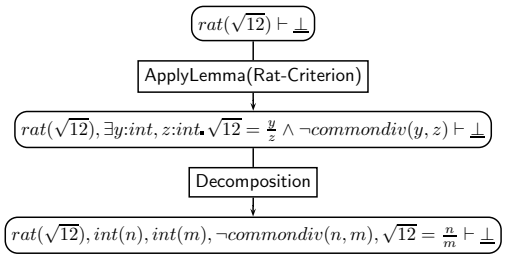
$$Rat - Criterion ::= \forall x : Rat.\ \exists y, z : int.\ \left(x = \tfrac{y}{z} \wedge \neg commondiv(y,z)\right)$$

It says that for all rational x there exists integers y,z, which have no common divisor and furthermore $x = \frac{y}{z}$.

The expansion of the abstract step makes the application of the Rat-Criterion theorem explicit. This works as follows: The application of Rat-Criterion to the assumption $rat(\sqrt{12})$ in the task $rat(\sqrt{12}) \vdash \underline{\bot}$ derives the new assumption $\exists y, z : int.\ \left(\sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z)\right)$, which results in the corresponding new task $rat(\sqrt{12}),\ \exists y, z : int.\ \left(\sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z)\right) \vdash \underline{\bot}$. Decomposition of the composed new assumption then derives the task

$$rat(\sqrt{12}),\ int(n),\ int(m),\ \neg commondiv(n,m),\ \sqrt{12} = \frac{n}{m} \vdash \underline{\bot}$$

Altogether the resulting expanded PDS fragment at a lower, more granular level has the form viewed on the right. Depending on the underlying basic calculus these steps either present already calculus steps or they can be further expanded. For instance, in the CORE system lemma appli cation is already a basic step.



As opposed thereto, in the old ΩMEGA system lemma applications have to be further expanded to derive Natural Deduction proofs.

The complete PDS maintaining simultaneously the initial abstract, less granular proof sketch and the lower, more granular verification of it is shown on the left-hand side in Fig. 4. It also contains the hierarchical edges $.\overset{h}{\longrightarrow}.$ which connect the different vertical layers. It supports four different PDS-views which result from alternative levels of granularity of the outgoing justifications of the initial node $\underline{\vdash \neg rat(\sqrt{12})}$ and of the outgoing justifications of the node $\underline{rat(\sqrt{12}) \vdash \bot}$. Selecting the upper justification for the set of alternatives for the first node and the lower justification for the latter node results in the PDS-view shown on the right-hand side in Fig 4.

5 Fundamental Alternatives for Modeling a PDS?

On a first glance, our modeling of a PDS may seem arbitrarily chosen among fundamentally different possibilities and dual or isomorphic structures. After deeper consideration, however, we believe that this is not quite the case:

Just as tasks are structured into proof attempts by recursive reduction to subtasks, alternative proof attempts result from multiple reduction of the same task. These can be considered as a hierarchy (Fig. 5): (1) multiple proof attempts, (2) task reduction to subtasks within one proof attempt, and (3) task composition from formulas.

Contrary to forms of political or juristic argumentation where total evidence is the sum of the evidences of alternative "proofs", in our area of application it typically suffices to establish a task only once, simply because a second proof does not give more evidence (under the current set of axioms) than a single one.

level	subject	subunits	connection of subunits	reason for the modus of the logical connection
1st	**Alternative Proof**	parallel reductions	disjunctive	area of application
2nd	**Reduction**	subtasks	conjunctive	disjunctive normal form together with level 1
3rd	**Task**	[signed] formulas	disjunctive	conjunctive normal form together with level 2

Fig. 5. Why the Logical Connections are not Arbitrary in Duality

As multiple mathematical proofs (1st level, Fig. 5) are thus connected disjunctively, it is advantageous to connect the subtasks resulting from a reduction (2nd level, Fig. 5) conjunctively, because the two levels together further the normalization of proof constructions to disjunctive normal form, resulting in a certain style. On the one hand, this style helps human beings to understand foreign proofs and maintain their own ones, and, on the other hand, makes it easier for automatic proof heuristics to recognize the triggering structures and applicable lemmas.

For the same reasons, it is advantageous to connect formulas inside a task (3rd level, Fig. 5) disjunctively. Indeed, we do not know of the dual choice of a conjunctive instead of a disjunctive connection of the formulas of a task in the literature. By tradition, both in informal human mathematical practice (starting form Aristotle's syllogisms and ending with lemmas in a modern textbook) and in formal logic calculi (Hilbert, resolution, Natural Deduction, tableau, sequent, and matrix calculi), tasks have a disjunctive structure.

6 Conclusion

This paper describes the PDS, a new generic proof data structure, which originates from and extends the successful data structures of the ΩMEGA and QUODLIBET systems. Among its key features are:

- the representation of alternative proof steps for *both* the reduction of a goal as well as for the expansion of a complex proof step to lower granularity
- the structuring of proof parts (i.e. lemmatization) into separate but connected parts of the data structure
- the generic representation of proof statements and justifications, biased neither to any specific calculus nor to any specific formalism for representing abstract proof plans.

The explicit introduction of hierarchical levels within one data structure supports the bridging between intuitive, abstract level proof development, proof explanation and proof verification. Whereas proofs are typically developed and presented at an abstract and intuitive level, proof verification typically requires some underlying calculus at a very low granularity. The PDS provides, for instance, the flexibility to perform alternative expansions of some abstract proof steps to represent the same proof idea in different underlying calculi. Maintaining simultaneously the proof at different levels of granularity accommodates, for instance, proof explanation systems, which can start with a presentation of the high-level proof, and on-demand generate presentations for expansions of *some* chosen proof steps [7]. Furthermore, the hierarchies represent the parts of the search space taken by automatic proof techniques, like for instance proof planning methods, tactics, and methodicals. Representing the search space as well as explored alternatives to represent the branches of the search space is well suited for debugging new proof techniques [6].

The PDS provides a flexible and general framework for storing and representing proofs under construction. However, the proof manipulations and refinements

manipulating the PDS have to be determined and controlled by the proof system making use of the PDS. This proof system has to handle and control operations such as backtracking, instantiation of variables, collection of constraints etc. Moreover, it has to decide about whether to allow and how to realize features such as local definitions and cyclic structures in the PDS.[2]

Many proof assistants actually provide proof data structures, e.g. Coq [4], Inka [11], Isabelle [15], NuPrl [12], TPS [1], and VSE [2] to mentioned only a few. However, to the best of our knowledge none of them has been designed to support such a horizontal and vertical representation mechanism for proofs as presented in this paper.

We implemented the generic PDS described in this paper in Allegro Lisp and defined a content independent XML format for exporting and importing forests, trees, or parts of them. Furthermore, we are able to visualize our three-dimensional graphs. For the interaction with the user, however, PDS-views are essential.

References

1. P.B. Andrews, M. Bishop, and C.E. Brown. System description: TPS: A theorem proving system for type theory. In *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, LNCS, pages 164–169. Springer, 2000.
2. S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, W. Stephan, R. Vogt, and A. Wolpers. VSE: Formal methods meet industrial needs. *International Journal on Software Tools for Technology Transfer, Special issue on Mechanized Theorem Proving for Technology, Springer*, september 1998.
3. J. Avenhaus, U. Kühler, T. Schmidt-Samoa, and C.-P. Wirth. How to prove inductive theorems? QuodLibet! In *Proc. of the 19th Int. Conf. on Automated Deduction (CADE-19)*, number 2741 in LNAI, pages 328–333. Springer, 2003.
4. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development — Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science, An EATCS Series. Springer, 2004.
5. L. Cheikhrouhou and V. Sorge. PDS — A Three-Dimensional Data Structure for Proof Plans. In *Proc. of the Int. Conf. on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, 2000.
6. L. Dixon. Interactive and hierarchical tracing of techniques in IsaPlanner. In *Proc. of UITP'05*, 2005.
7. A. Fiedler. Dialog-driven adaptation of explanations of proofs. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.
8. The OMEGA Group. Proof development with ΩMEGA. In *Proc. of the 18th Int. Conf. on Automated Deduction (CADE-18)*, number 2392 in LNAI, pages 143–148. Springer, 2002.
9. X. Huang. Reconstructing proofs at the assertion level. In *Proc. of the 12th Int. Conf. on Automated Deduction (CADE-12)*, LNAI, pages 738–752. Springer, 1994.

[2] Technically, cyclic structures can be realized by cyclic lemmas in PDS forests, [18].

10. M. Hübner, S. Autexier, C. Benzmüller, and A. Meier. Interactive theorem proving with tasks. *Electronic Notes in Theoretical Computer Science*, 103(C):161–181, November 2004.
11. D. Hutter and C. Sengler. INKA - The Next Generation. In *Proc. of the 13th International Conference on Automated Deduction (CADE–13)*, LNAI. Springer, 1996.
12. C. Kreitz, L. Lorigo, R. Eaton, R.L. Constable, and S.F. Allen. The nuprl open logical environment, 2000.
13. Andreas Meier. *Proof Planning with Multiple Strategies*. PhD thesis, Saarland Univ., 2004.
14. E. Melis and J. Siekmann. Knowledge-Based Proof Planning. *Artificial Intelligence*, 115(1):65–105, 1999.
15. L.C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS. Springer, 1994.
16. J. Siekmann, C. Benzmüller, A. Fiedler, A Meier, I. Normann, and M. Pollet. *Proof Development in OMEGA: The Irrationality of Square Root of 2*, pages 271–314. Kluwer Academic Publishers, 2003.
17. J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with OMEGA: Sqrt(2) is irrational. In *Logic for Programming, Artificial Intelligence, and Reasoning, 9th Int. Conf., LPAR 2002*, number 2514 in LNAI, pages 367–387. Springer, 2002.
18. C.-P. Wirth. Descente infinie + Deduction. *Logic J. of the IGPL*, 12(1):1–96, 2004.