Authoring Verified Documents by Interactive Proof Construction and Verification in Text-Editors

Dominik Dietrich, Ewaryst Schulz, and Marc Wagner

FR 6.2 Informatik, Saarland University, Saarbrücken, Germany
{dietrich,schulz,wagner}@ags.uni-sb.de

Abstract. Aiming at a document-centric approach to formalizing and verifying mathematics and software we integrated the proof assistance system ΩMEGA with the standard scientific text-editor TEXMACS. The author writes her mathematical document entirely inside the text-editor in a controlled language with formulas in LATEX style. The notation specified in such a document is used for both parsing and rendering formulas in the document. To make this approach effectively usable as a real-time application we present an efficient hybrid parsing technique that is able to deal with the scalability problem resulting from modifying or extending notation dynamically. Furthermore, we present incremental methods to quickly verify constructed or modified proof steps by ΩMEGA. If the system detects incomplete or underspecified proof steps, it tries to automatically repair them. For collaborative authoring we propose to manage partially or fully verified documents together with its justifications and notational information centrally in a mathematics repository using an extension of OMDOC.

1 Introduction

Unlike widely used computer-algebra systems, mathematical assistance systems have not yet achieved considerable recognition and relevance in mathematical practice. One significant shortcoming of the current systems is that they are not fully integrated into or accessible from standard tools that are already routinely employed in practice, like, for instance, standard mathematical text-editors. Integrating formal modeling and reasoning with tools that are routinely employed in specific areas is the key step in promoting the use of formal logic based techniques.

Therefore, in order to foster the use of proof assistance systems, we integrated the theorem prover ΩMEGA [7] into the scientific text-editor TEXMACS [15]. The goal is to assist the author inside the editor while preparing a TEXMACS document in a publishable format. The vision underlying this research is to enable a document-centric approach to formalizing and verifying mathematics and software. We tackle this vision by investigating two orthogonal approaches in parallel: On the one hand we start with mathematical documents written without any restrictions and try to extract the semantic content with natural language analysis techniques and accordingly generate or modify parts of the document using natural language generation. On the other hand we start with the semantic content and lift it to an abstract human-oriented representation without losing the benefits of machine processability. This paper describes our recent results for the second approach, resulting in a system in which the author can write a document in $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$, which gets automatically proof-checked by ΩMEGA. When the document is changed, the dependent parts are automatically rechecked.

In our scenario the workflow for the author, who is preparing a mathematical document to be verified, consists of arbitrary combinations of the following operations: (1) writing theory with notation or citing theory and eventually redefining notation, (2) developing proofs by constructing or modifying proof steps that are continuously checked and possibly repaired by the proof assistance system if incomplete or underspecified, and (3) saving the current state of the document including the verification information in order to continue at a later date. This raises the following requirements for effectively supporting the author in real-time: (1) a fast parsing and rendering mechanism with efficient adaptation to changes in the notation rules, (2) quick incremental proof checking and repair techniques, and (3) an output format containing the formalized content of the document together with its justifications and notational information.

The paper is organized as follows: Section 2 presents in more detail our general architecture consisting of a mediator, a proof assistance system and a semantic repository. Section 3 introduces the hybrid parsing technique that efficiently deals with the controlled authoring language and notational extensions or modifications. The formal proof representation of the ΩMEGA system is defined in Section 4 including the notion of *proof view*. The techniques for management of change needed (1) to incrementally verify proof steps constructed or modified in the text-editor, and (2) to lift corrections or complete proofs from the internal format of the proof assistance system to the document are described in Section 5. In Section 6 we discuss how OMDOC [10] can be extended such that it can also store proof steps at different levels of granularity as well as parsing and rendering knowledge. We discuss the current situation for authoring verified mathematical documents as related work in Section 7 and summarize the paper in Section 8.

2 Architecture

Although this paper focuses on the interplay between a mediator and a proof assistance system, we propose to fill the authoring gap for semantic mathematics repositories with our complementary architecture. The envisioned architecture is a cooperation between a mediator, a proof assistance system and a semantic repository. The mediator parses and renders the informal content authored in a text-editor and propagates changes to the proof assistance system that provides services for verification and automatic proof construction. The semantic repository takes care of the management of formalized mathematics. Figure 1 illustrates the flow of mathematical knowledge. The big circles in the figure are abstract components of the architecture that can be instantiated by the concrete components attached to them, e.g. PLATΩ as mediator. The text between the arrows indicates the kind of knowledge that is exchanged. In detail, the roles and requirements of the components are:

Mediator. Following our document-centric philosophy, the document in the text-editor is both the human-oriented input and output representation for the proof assistance system, thus the central source of knowledge. The role of the mediator PLATΩ [16] is to preserve consistency between the text-editor and the proof assistance system

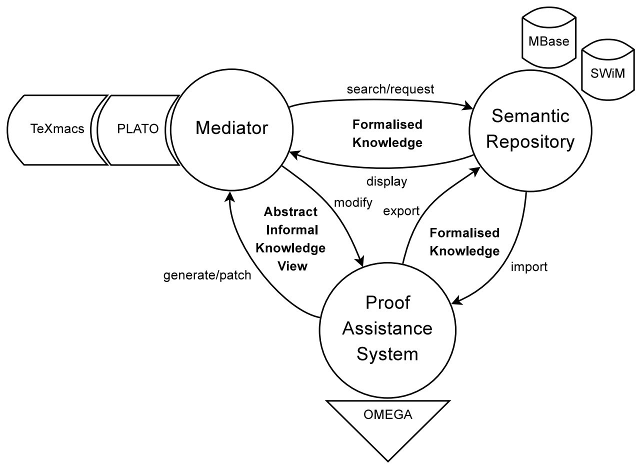


Fig. 1. Architecture for Authoring Mathematics Repositories

by incrementally propagating changes. Additionally, services and feedback of the proof assistance system are provided inside the text-editor through context-sensitive menus. The mediator allows to define, modify and overload the notation used in the document dynamically within the document.

Proof Assistance System. The role of the proof assistance system is to maintain the formal proof object in a way that it is verifiable and such that at the same time a human readable presentation can be extracted. It must be able to verify and integrate updates sent from the mediator, and provide means to automate parts of the proof. Finally, the system should be able to import from and export to standards such as OMDoc.

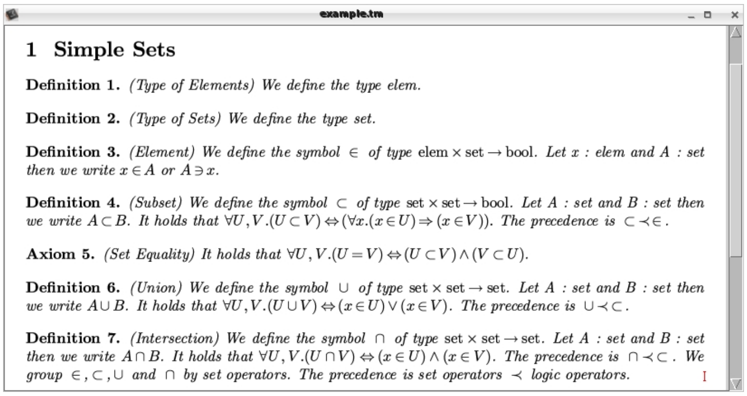
In our architecture we use the proof assistance system ΩMEGA, which is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. Proof planning is interesting for our architecture because it naturally supports to express proof plans and their expansion, i.e. verification.

Semantic Repository. The role of the semantic repository (e.g. in form of a database or a wiki) is to store and maintain the mathematical knowledge using structural semantic markup or scripting languages, including possibilities to search and retrieve knowledge and access control. The MBase system [8] is for example a web-based, distributed mathematical knowledge base that allows for semantic-based retrieval. Semantic Wiki technologies like SWiM [11] are a current subject of research for collaboratively building, editing and browsing mathematical knowledge. Both types of semantic repositories are well-suited for our architecture because they store mathematical theories and statements in the OMDoc format and support dependency-aware semantic content retrieval.

Altogether, the proposed architecture allows for the incremental interactive development of verified mathematical documents at a high level of abstraction. By using the scientific WYSIWYG text-editor TeXmacs, the author additionally benefits from professional type-setting and powerful macro definition facilities like in LaTeX.

3 Hybrid Parsing

Let us first introduce an example of the kind of documents we want to support. Figure 2 shows a theory about *Simple Sets* written in the text-editor TEXMACS. This theory defines two base types and several set operators together with their axioms and notations. In general, theories are built on top of other theories and may contain definitions, notations, axioms, lemmas, theorems and proofs. Note that in the example set equality is written as an axiom because equality is already defined in the base theory.


example.tm

1 Simple Sets

Definition 1. *(Type of Elements) We define the type elem.*

Definition 2. *(Type of Sets) We define the type set.*

Definition 3. *(Element) We define the symbol* $\in$ *of type* $\mathrm{elem} \times \mathrm{set} \to \mathrm{bool}$. *Let* x : *elem and* A : *set then we write* $x \in A$ *or* $A \ni x$.

Definition 4. *(Subset) We define the symbol* $\subset$ *of type* $\mathrm{set} \times \mathrm{set} \to \mathrm{bool}$. *Let* A : *set and* B : *set then we write* $A \subset B$. *It holds that* $\forall U, V.(U \subset V) \Leftrightarrow (\forall x.(x \in U) \Rightarrow (x \in V))$. *The precedence is* $\subset \prec \in$.

Axiom 5. *(Set Equality) It holds that* $\forall U, V.(U = V) \Leftrightarrow (U \subset V) \wedge (V \subset U)$.

Definition 6. *(Union) We define the symbol* $\cup$ *of type* $\mathrm{set} \times \mathrm{set} \to \mathrm{set}$. *Let* A : *set and* B : *set then we write* $A \cup B$. *It holds that* $\forall U, V.(U \cup V) \Leftrightarrow (x \in U) \vee (x \in V)$. *The precedence is* $\cup \prec \subset$.

Definition 7. *(Intersection) We define the symbol* $\cap$ *of type* $\mathrm{set} \times \mathrm{set} \to \mathrm{set}$. *Let* A : *set and* B : *set then we write* $A \cap B$. *It holds that* $\forall U, V.(U \cap V) \Leftrightarrow (x \in U) \wedge (x \in V)$. *The precedence is* $\cap \prec \subset$. *We group* $\in, \subset, \cup$ *and* $\cap$ *by set operators. The precedence is set operators* $\prec$ *logic operators.*

Fig. 2. Document in the text-editor TEXMACS

In our previous approach [16], the author had full freedom in writing her document but had to manually provide semantic annotations. We now use a controlled authoring language to skip the burden of providing annotations, thus increasing the overall usability by dealing with pure TEXMACS documents. The grammar of the concrete syntax is given in Table 1. `NAME` and `LABEL` are unique string identifiers. `URI` is a resource location. `SYM`, `VAR`, `TYPE`, `PAT` and `TERM` represent symbol, variable, type, pattern and term respectively. Please note that TEXMACS renders for example the macro "`<definition|text>`" into "**Definition 1.** text".

Dynamic Notation. This initial grammar can be extended on the fly by introducing new types and symbols as well as defining, extending or overloading their notations within a document. In [3] we presented a basic mechanism that allows the user to define notations by declaring local variables (e.g. A,B) and specifying notation patterns (e.g. $A \subset B, B \supset A$). The mechanism synthesizes automatically parsing rules from all patterns and the default rendering rule from the first pattern. The author can group operators, specify their associativity and define precedences as a partial ordering. Furthermore, if the notation is modified all affected formulas in the document are adapted efficiently, the right order of notation and formulas is checked, ambiguities are prevented using a family of theory-specific parsers and resolved by exploiting type information. The hierarchical structure of theories allows the reuse of concepts together with their notation even from other documents. Note that one can import theories together with their proofs and notations from other files by a reference in the document. Dynamic Notation

Table 1. Grammar for the Concrete Syntax of the Authoring Language

DOC	::= THY*	AXM	::= ‘<axiom|’ ‘(’ NAME ‘)’ ALTC ‘>’
THY	::= ‘<section|’ NAME ‘>’ CTX$^?$ THYC	LEM	::= ‘<lemma|’ ‘(’ NAME ‘)’ ALTC ‘>’
CTX	::= ‘We’ ‘use’ CREFS ‘.’	TEO	::= ‘<theorem|’ ‘(’ NAME ‘)’ ALTC ‘>’
CREFS	::= CREF (‘,’ CREF)* (‘and’ CREF)$^?$	ALTC	::= ‘It’ ‘holds’ ‘that’ FORM ‘.’
CREF	::= NAME|URI	PRF	::= ‘<proof|’ STEPS$^?$ ‘>’
THYC	::= (DEF|AXM|LEM|TEO|PRF)*	STEPS	::= (OSTEP STEPS$^?$)|CSTEP
DEF	::= ‘<definition|’ ‘(’ NAME ‘)’ DEFC ‘>’	OSTEP	::= SET|ASS|FACT|GOAL|CGOAL
DEFC	::= (DEFT|DEFS) NOTC$^?$ ALTC$^?$ SPEC*	CSTEP	::= GOALS|CASES|CGOALS|TRIV
DEFT	::= ‘We’ ‘define’ ‘the’ ‘type’ NAME ‘.’	TRIV	::= ‘Trivial’ BY$^?$ FROM$^?$ ‘.’
DEFS	::= ‘We’ ‘define’ ‘the’ ‘symbol’ NAME ‘of’ ‘type’ TYPE ‘.’	SET	::= ‘We’ ‘define’ FORMS ‘.’
		ASS	::= ‘We’ ‘assume’ FORMS FROM$^?$ ‘.’
NOTC	::= ‘Let’ TVARS ‘then’ ‘we’ ‘write’ PATS ‘.’	FACT	::= ‘It’ ‘follows’ ‘that’ FORMS BY$^?$ FROM$^?$ ‘.’
TVARS	::= VAR ‘:’ TYPE (‘,’ VAR ‘:’ TYPE)* (‘and’ VAR ‘:’ TYPE)$^?$	GOAL	::= ‘We’ ‘have’ ‘to’ ‘prove’ FORM BY$^?$ FROM$^?$ ‘.’
		GOALS	::= ‘We’ ‘have’ ‘to’ ‘show’ FORMS BY$^?$ FROM$^?$ ‘.’ SPRF*
PATS	::= PAT (‘,’ PAT)* (‘or’ PAT)$^?$		
PAT	::= (VAR|STRING)$^+$	CASES	::= ‘We’ ‘have’ ‘the’ ‘cases’ FORMS BY$^?$ FROM$^?$ ‘.’ SPRF*
SPEC	::= GROUP|PREC|ASSOC		
GROUP	::= ‘We’ ‘group’ SYM (‘,’ SYM)* (‘and’ SYM)$^?$ ‘by’ NAME ‘.’	CGOAL	::= ‘We’ ‘have’ ‘to’ ‘prove’ CFORM BY$^?$ FROM$^?$ ‘.’
		CGOALS	::= ‘We’ ‘have’ ‘to’ ‘show’ CFORMS BY$^?$ FROM$^?$ ‘.’ SPRF*
PREC	::= ‘The’ ‘precedence’ ‘is’ (SYM|NAME) (‘$\prec$’ (SYM|NAME))$^+$ ‘.’	SPRF	::= ‘We’ ‘prove’ LABEL ‘.’ STEPS$^?$
		BY	::= ‘by’ NAME
ASSOC	::= ‘The’ ‘operator’ SYM ‘is’ ‘right-associative’ ‘.’	FROM	::= ‘from’ LABEL (‘,’ LABEL)* (‘and’ LABEL)$^?$
FORMS	::= FORM (‘,’ FORM)* (‘and’ FORM)$^?$	CFORMS	::= CFORM (‘,’ CFORM)* (‘and’ CFORM)$^?$
FORM	::= (‘(’ LABEL ‘)’)$^?$ TERM	CFORM	::= FORM ‘assuming’ FORMS

is aware of the positions of defining and using occurrences for notation but it does not take into account notational knowledge obtained by proven theorems yet; this is future work.

Speed Issues. Although we minimized the need for compiling parsers, the processing of the standard example in [3] took $\approx 1min$. The main reasons for inefficiency were (1) the parsers were compiled in interpreted mode in the text-editor, and (2) the scalability problem of LALR parser generators. Problem (1) has been solved by moving the parser generation to the mediator, but even in compiled mode the processing took $\approx 6sec$ which is still not sufficiently fast for real-time usage. The remaining issue (2) is severe because when notations are changed or extended all parsers for dependent theories in the hierarchy have to be recompiled. Therefore we integrated a directly-executable variant of an Earley parser [6] which is substantially faster than standard Earley parsers to the point where it is comparable with standard LALR(1) parsers. Although the time for parsing a single formula increases slightly, the overall processing of the example takes $\approx 0.1sec$ which is perfectly suitable for a real-time application.

Algorithm. First of all, the document is preprocessed and split into segments almost corresponding to sentences. Then the following steps are incrementally performed for each segment: (1) the static parts of the authoring language, i.e. the controlled phrase structure, are parsed using a precompiled LALR(1) parser; (2) the dynamic parts of the authoring language, i.e. the formulas and notations, are parsed using a theory-specific Earley parser; (3) the segment is propagated to the proof assistance system. Note that the dynamic parts are always strictly separated from the static parts in the document because they are written inside a math mode macro.

Normalization and Abstraction. The concrete syntax of the authoring language allows for variant kinds of syntax sugaring that has to be normalized for machine

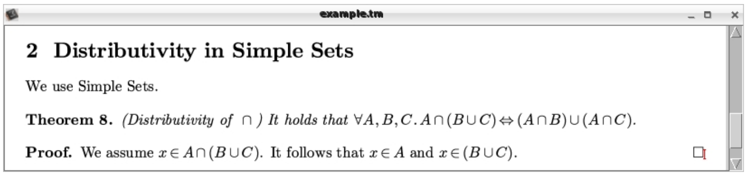
Table 2. Grammar for the Abstract Syntax of the Proof Language

PROOF	::= STEPS	TRIVIAL	::= **trivial** BY FROM
STEPS	::= (OSTEP;STEPS)$\mid$CSTEP	SET	::= **set** FORMULA
OSTEP	::= SET$\mid$ASSUME$\mid$FACT	ASSUME	::= **assume** FORMULA FROM
CSTEP	::= GOALS$\mid$CASES$\mid$COMPLEX$\mid$TRIVIAL$\mid\varepsilon$	FACT	::= **fact** FORMULA BY FROM
FORMULA	::= (LABEL :)$^?$ TERM	GOALS	::= **subgoals** (FORMULA { PROOF })$^+$ BY FROM
BY	::= **by** NAME$^?$	CASES	::= **cases** (FORMULA { PROOF })$^+$ BY FROM
FROM	::= **from** (LABEL (, LABEL)*)$^?$	COMPLEX	::= **complex** COMP$^+$ BY FROM
		COMP	::= FORMULA **under** FORMULA { PROOF }

processing, e.g. formula aggregation ($x \in A, x \in B$ and $x \in C$) or the ordering of subproofs. Table 2 defines a normalized abstract syntax for the proof part of the authoring language. Aggregated formulas are composed to one formula by conjunction or disjunction depending on whether they are hypotheses or goals respectively. If an abstract proof step is generated by the system, the mediator tries to decompose the formula for aggregation accordingly. A proof is implicitly related to the last stated theorem previous to this proof in the document. Subproofs are grouped together with their subgoal or case they belong to. A single goal reduction is normalized to a subgoals step.

Management of Change. Using management of change we propagate incrementally arbitrary changes between the concrete and abstract representation. By additionally considering the semantics of the language we can optimize the differencing mechanism [12]. For example the reordering of subgoals or their subproofs in the text-editor is not propagated at all because it has no impact on the formal verification. The granularity of differencing is furthermore limited to the reasonable level of proof steps and formulas, s.t. deep changes in a formula are handled as a complete modification of the formula. The propagation of changes is essential for this real-time application because complete re-transformation and re-verification slows down the response time too much. Apart from that the differencing information allows for the local re-processing of affected segments instead of a global top-down re-processing using a replay mechanism. In order to recheck only dependent parts ΩMEGA uses Development Graphs [4] for the management of change for theories.

Let us now continue our example with a new theory in Figure 3 that refers to the previous one and that states a theorem the author already started to prove.


example.tm

2 Distributivity in Simple Sets

We use Simple Sets.

Theorem 8. *(Distributivity of* $\cap$ *) It holds that* $\forall A,B,C. A\cap(B\cup C) \Leftrightarrow (A\cap B)\cup(A\cap C)$.

Proof. We assume $x \in A\cap(B\cup C)$. It follows that $x \in A$ and $x \in (B\cup C)$.

Fig. 3. Theorem with partial proof

This partial proof in concrete syntax is then abstracted as shown in Table 3, where we additionally emphasized underspecified parts with a dot.

Table 3. Proof in abstract syntax

Proof.
assume $x \in A\cap(B\cup C)$ **from .** ;
fact $x \in A \wedge x \in (B\cup C)$ **by . from .** ; ε

4 Formal Proof Representation

In this section we describe how proofs are internally represented in ΩMEGA. This will allow us to describe how proof scripts are processed by the prover. In the ΩMEGA system proofs are constructed by the TASKLAYER that uses an instance of the generic proof data structure (PDS) [1] to represent proofs. One main feature of the PDS is the ability to maintain subproofs at different levels of granularity simultaneously, including a so-called *PDS-view* representing a complete proof at a specific granularity.

Task. At the TASKLAYER, the main entity is a task, a multi-conclusion sequent $F_1,\dots,F_j \vdash G_1,\dots,G_k$. Each formula can be named by assigning a label l to the formula. We denote the set of all term positions by **Pos**, the set of all admissible positions of a task T by $\mathbf{Pos}(T)$, and the position of the formula with label l by $\mathrm{pos}(l)$. Moreover, we write T_π to denote the subformula at position π and write $T_{\pi \leftarrow s}$ for the type compliant replacement of the subterm T_π by s. We use the notation $\Gamma \star \varphi$ to denote the set $\Gamma \cup \{\varphi\}$.

Agenda. The proof attempt is represented by an agenda. It maintains a set of tasks, that are the subproblems to be solved, and a global substitution which instantiates meta-variables. Formally an agenda is a triple $\mathcal{A} = \langle T_1,\dots,T_n;\sigma;T_j\rangle$ where $T_1,\dots,T_n$ are tasks, σ is a substitution, and T_j is the task the user is currently working on. We will use the notation $\langle T_1,\dots,T_{j-1},\underline{T_j},T_{j+1}\dots T_n;\sigma\rangle$ to denote that the task T_j is the current task. Note that the application of a substitution is a global operation. To reflect the evolutional structure of a proof, a substitution is applied to the open tasks of the agenda. Whenever a task is reduced to a list of subtasks, the substitution before the reduction step is stored within the PDS in the node for that task.

The Figure on the right shows the reconstruction of the first proof step of the partial proof of **Theorem 8**. Tasks are shown as oval boxes connected by justifications, where the squared boxes indicate which inference has been applied. The agenda to the shown PDS consists of the two leaf tasks. The global substitution σ is the identity id.

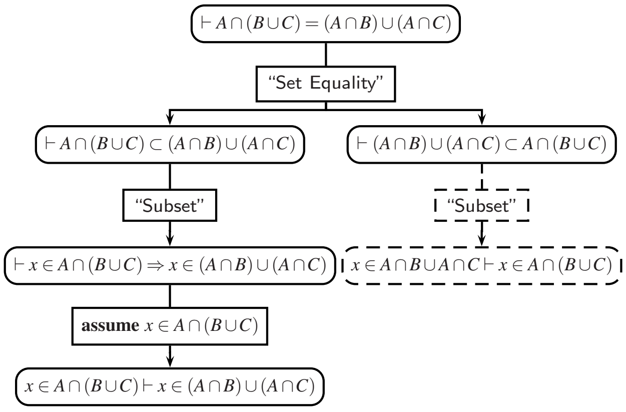

$\vdash A\cap(B\cup C) = (A\cap B)\cup(A\cap C)$
"Set Equality"
$\vdash A\cap(B\cup C) \subset (A\cap B)\cup(A\cap C)$
$\vdash (A\cap B)\cup(A\cap C) \subset A\cap(B\cup C)$
"Subset"
"Subset"
$\vdash x\in A\cap(B\cup C) \Rightarrow x\in(A\cap B)\cup(A\cap C)$
$x\in A\cap B\cup A\cap C \vdash x\in A\cap(B\cup C)$
assume $x\in A\cap(B\cup C)$
$x\in A\cap(B\cup C) \vdash x\in(A\cap B)\cup(A\cap C)$

Fig. 4. Proof in ΩMEGA

5 Incremental Proof Step Verification and Correction

In this section we describe how information encoded as a proof script can be exchanged between the mediator and the prover. There are two possible information flows: First, a proof script sent by the mediator must be converted into the internal proof structure of the prover and thereby be checked. Second, given a proof generated by the prover, a corresponding proof script must be extracted, which can then be propagated to the mediator. Note that the last step is necessary as there is no guarantee that all parts of the proof have been constructed by a proof script. Before describing both directions we give an overview of the proof operators of the prover, which are *inferences* and *strategies*.

Inferences. Intuitively, an *inference* is a proof step with multiple premises and one conclusion augmented by (1) a possibly empty set of hypotheses for each premise, (2) a set of *application conditions* that must be fulfilled upon inference application, (3) a set of *completion functions* that compute the values of premises and conclusions from values of other premises and conclusions (see [2] for a formal definition). Each premise and conclusion consists of a unique name and a formula scheme. Consider for example the inference *ImplIntro* shown above. It consists of one conclusion with name C and formula scheme $\varphi \Rightarrow \psi$. Moreover, it has one premise with name P_1, formula scheme ψ, and hypothesis H_1, which has the formula scheme φ associated with it. In the sequel we write C to denote the conclusion of the inference and P_i to denote the ith premise of the inference.

$$\dfrac{\begin{matrix}[H_1 : \varphi] \\ \vdots \\ P_1 : \psi\end{matrix}}{C : \varphi \Rightarrow \psi}\ \mathit{ImplIntro}$$

Given a task, we can instantiate an inference with respect to the task by trying to find formulas in the task unifying with the formula scheme of a premise or conclusion. Technically, such an instantiation is represented by an *inference substitution* σ which binds premises and the conclusion to formulas or positions. We consider two parts of the substitution: σ_x contains instantiations of meta-variables of the task, and σ_c maps a premise or conclusion A to positions of the task (see [2] for details), which is $\perp$ in case that A is not matched. The instantiated formula scheme is denoted by $\mathrm{fs}(A)$.

Strategies. As basis for automation ΩMEGA provides so-called strategies, which tackle a problem by some mathematical standard problem solving workflow that happens to be typical for this problem. To achieve a goal, a strategy performs a heuristically guided search using a dynamic set of inferences (as well as other strategies), and control rules. A strategy application either fails or constructs a subproof using the specified inferences and substrategies. In the latter case the constructed subproof is abstracted by inserting a justification labelled with the strategy application and connecting the task the strategy has been applied to with the nodes resulting from the strategy application. This has the advantage that the user can switch between the abstract step or the more detailed step. Note that from a technical point of view a strategy is similar to a tactic. However, by using explicit control knowledge its specification is declarative, whereas tactics are usually procedurally specified.

$$\begin{tikzcd}
 & \boxed{\mathit{strat}} \arrow[rr] & & T' \\
T \arrow[ur, no head] \arrow[dr, no head] & & & \\
 & \boxed{s_1} \arrow[uu, "h"'] \arrow[r] & \cdots \arrow[uur, bend right] &
\end{tikzcd}$$

Consider a strategy *strat* which is applied to a task T, and constructs a subproof starting with the application of s_1 and finally leading to the task T'. In this case, a new justification labelled with *strat* is inserted connecting T and T'. To indicate that the strategy application is more abstract than the subproof a hierarchical edge h is inserted, defining an ordering on the outgoing edges of T. The resulting PDS is shown above.

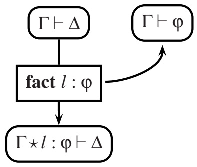
5.1 Proof Checking and Repair

The verification of a single proof step can become time consuming if some information is underspecified. In the worst case a complete proof search has to be performed. To obtain adequate response times a given step is worked off in two phases. First, we perform a quickcheck, where we assume that the given step can be justified by the prover by a single inference application. This can be tested by a simple matching algorithm. If the test proceeds the step is sound, as the inference application is proved to be correct.

If it is not possible to justify the step with a single inference application, a more complex repair mechanism is started. This mechanism tries to find the missing information needed to justify the step by performing a heuristically guided resource bounded search. If we are not able to find a derivation within the given bound, a failure is reported and sent to the mediator, which can then initiate a specific reaction.

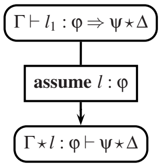
In the sequel we describe for each construct of our proof language its quickcheck and its repair mechanism, which are both modeled as a proof strategy in ΩMEGA. The quickcheck rules are summarized in Table 4. We use the notation $\langle\{\underline{\Gamma \vdash \Delta}, T_2, \ldots, T_n\}, \sigma\rangle : s\,;\, S \rightarrow \langle\{\underline{\Gamma' \vdash \Delta'}, T_2, \ldots, T_n\}, \sigma\rangle : S'$ to indicate that under the agenda $\langle\{\underline{\Gamma \vdash \Delta}, T_2, \ldots, T_n\}, \sigma\rangle$ the proof step s in the sequence $s\,;\,S$ can be checked, and that the checking results in a new agenda $\langle\{\underline{\Gamma' \vdash \Delta'}, T_2, \ldots, T_n\}, \sigma\rangle$ where the steps S' have to be checked.

Fact. The command **fact** derives a new formula φ with label l from the current proof context. The quickcheck tries to justify the new fact by the application of the inference or strategy *name* to term positions in the formulas with labels $l_1, \ldots, l_n$ in the current task. Although the specification of these formulas speeds up the matching process, both informations **by** and **from** can be underspecified in general. In this case, all inferences are matched against all admissible term positions, and the first one which delivers the desired formula φ is applied.



If the above check fails, the repair strategy for **fact** is started. It generates a new lemma, i.e. a new proof tree, containing the assumptions of the current task and the newly stated fact φ as goal. It then tries to automatically close the lemma by standard forward and backward reasoning. If the new lemma can be proved, the lemma is automatically transformed to an inference using the mechanism described in [2], which then justifies the step by a single inference application in the original proof.

Stating a new lemma has several advantages: Even if the lemma cannot automatically be checked, we can continue to check subsequent proof steps. The new lemma can then be proved with user interaction at a later time. Moreover, as the lemma is transformed into a single inference, it is globally available and can be used in similar situations by the quickcheck without performing any search.


$\Gamma \vdash l_1 : \varphi \Rightarrow \psi \star \Delta$
assume $l : \varphi$
$\Gamma \star l : \varphi \vdash \psi \star \Delta$

Assume. The command **assume** introduces a new assumption φ on the left hand side of the current task. The quickcheck for **assume** checks whether one of the following situations occurs, each of which can be justified by a particular inference application:

- Δ contains $\varphi \Rightarrow \psi$. The implication is decomposed and $l : \varphi$ is added to the left side of the task.
- Δ contains $\neg\varphi$. Then $l : \varphi$ is added to the left side.
- Δ contains $\psi \Rightarrow \neg\varphi$. Then $l : \varphi$ is added to the left side and $\neg\psi$ to the right side of the task.

If the quickcheck fails we try to derive one of the above situations by applying inferences to the goal of the current task. The hypotheses of the task remain untouched.

Table 4. Quick checking rules

$$\langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{fact}\ l:\varphi\ \textbf{by}\ \mathit{name}\ \textbf{from}\ l_1,\ldots,l_n;S \rightarrow \langle\{\underline{\Gamma\star l:\varphi\vdash\Delta},T_2,\ldots,T_m\},\sigma_X^{name}\circ\sigma\rangle : S$$
$$\text{with } \sigma^{name}(C)=\bot \text{ and } \bigcup\sigma_C^{name}(P_i)=\bigcup\{\mathrm{pos}(l_i)\}$$

$$\langle\{\underline{\Gamma\vdash\varphi\Rightarrow\psi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{assume}\ l:\varphi\ \textbf{from}\ l_1;S \rightarrow \langle\{\underline{\Gamma\star l:\varphi\vdash\psi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : S$$
$$\text{with } \sigma_C^{implintro}(C)=\mathrm{pos}(l_1) \text{ and } \sigma_C^{implintro}(P)=\bot$$

$$\langle\{\underline{\Gamma\vdash\neg\varphi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{assume}\ l:\varphi\ \textbf{from}\ l_1;S \rightarrow \langle\{\underline{\Gamma\star l:\varphi\vdash\Delta\star\bot},T_2,\ldots,T_m\},\sigma\rangle : S$$
$$\text{with } \sigma_C^{contradiction}(C)=\mathrm{pos}(l_1) \text{ and } \sigma_C^{implintro}(P)=\bot$$

$$\langle\{\underline{\Gamma\vdash\psi\Rightarrow\neg\phi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{assume}\ l:\varphi\ \textbf{from}\ l_1;S \rightarrow \langle\{\underline{\Gamma\star l:\phi\vdash\neg\psi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : S$$
$$\text{with } \sigma^{contrapositive}(C)=\mathrm{pos}(l_1) \text{ and } \sigma^{contrapositive}(P)=\bot$$

$$\langle\{\underline{\Gamma\vdash\psi\star\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{subgoals}\ l_1':\varphi_1\{S_1\},\ldots,l_n':\varphi_n\{S_n\}\ \textbf{by}\ \mathit{name}\ \textbf{from}\ l_1,\ldots,l_n;S$$
$$\rightarrow \langle\{\underline{\Gamma\vdash\Delta\star l_1':\sigma^{name}(P_1)},\ldots,\Gamma\vdash\Delta\star l_{n+k}':\sigma^{name}(P_{n+k}),T_2,\ldots,T_m\},\sigma_X^{name}\circ\sigma\rangle : S_1;\ldots;S_n;S$$
$$\text{with } T|_{\sigma^{name}(C)}=\psi \text{ and } (\bigcup\{\sigma_C^{name}(P_i)\}\cup\{\sigma_C^{name}(C)\})=\bigcup\{\mathrm{pos}(l_i)\}$$

$$\langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{cases}\ l_1':\varphi_1\{S_1\},\ldots,l_n':\varphi_n\{S_n\}\ \textbf{from}\ l;S$$
$$\rightarrow \langle\{\underline{\Gamma\vdash\Delta|_{\mathrm{pos}(l)\leftarrow\varphi_1}},\ldots,\Gamma\vdash\Delta|_{\mathrm{pos}(l)\leftarrow\varphi_n},T_2,\ldots,T_m\},\sigma\rangle : S_1;\ldots;S_n;S \qquad \text{with } T|_{\mathrm{pos}(l)}=\varphi_1\vee\ldots\vee\varphi_n$$

$$\langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{set}\ x=t;S \rightarrow \langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},[x=t]\circ\sigma\rangle : S \qquad \text{if } x \text{ occurs in } (\Gamma,\Delta)$$

$$\langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{set}\ x=t;S \rightarrow \langle\{\underline{\Gamma\star x=t\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : S \qquad \text{if } x \text{ is new wrt. } \Gamma,\Delta$$

$$\langle\{\underline{\Gamma\star\bot\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{trivial} \rightarrow \langle\{\underline{T_2},\ldots,T_m\},\sigma\rangle :$$

$$\langle\{\underline{\Gamma\vdash\Delta\star\top},T_2,\ldots,T_m\},\sigma\rangle : \textbf{trivial} \rightarrow \langle\{\underline{T_2},\ldots,T_m\},\sigma\rangle :$$

$$\langle\{\underline{\Gamma\star\varphi\vdash\Delta\star\varphi},T_2,\ldots,T_m\},\sigma\rangle : \textbf{trivial} \rightarrow \langle\{\underline{T_2},\ldots,T_m\},\sigma\rangle :$$

$$\langle\{\underline{\Gamma\vdash\Delta},T_2,\ldots,T_m\},\sigma\rangle : \textbf{trivial}\ \textbf{by}\ \mathit{name}\ \textbf{from}\ l_1,\ldots,l_n \rightarrow \langle\{\underline{T_2},\ldots,T_m\},\sigma\rangle$$
$$\text{with } \bigcup\sigma_C^{name}(P_i)\cup\sigma_C^{name}(C)=\bigcup\{\mathrm{pos}(l_i)\}$$

$$\langle\{T_1,\ldots,T_{j-1},\underline{T_j},T_{j+1},\ldots,T_m\},\sigma\rangle : \varepsilon \rightarrow \langle\{T_1,\ldots,T_{j-1},T_j,\underline{T_{j+1}},\ldots,T_m\},\sigma\rangle$$

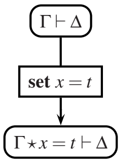
To increase the readability, subsequent steps which reduce a task to a single subtask are grouped together to a single step. Technically this is done by inserting a hierarchical edge. As default the most abstract proof step is propagated to the mediator.

Subgoals. The command **subgoals** reduces a goal of a given task to $n+m$ subgoals, each of which is represented as a new task, where n corresponds to the subgoals specified by the user and m denotes additional underspecified goals the user has omitted or forgotten. Each new task stems from a premise P_i of the applied inference, where the goal of the original task is replaced by the proof obligation for the premise, written as $pob(P_i)$. The quickcheck succeeds if the specified inference *name* introduces at least the subgoals specified by the user.

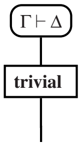
$$\begin{tikzcd}
 & & & \boxed{\Gamma\vdash\Delta} \arrow[d, dash] & & & \\
 & & & \boxed{\textbf{subgoals}\ l_1:\varphi_1,\ldots,l_n:\varphi_n} \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
\boxed{\Gamma\vdash l_1:\varphi_1} & \cdots & \boxed{\Gamma\vdash l_n:\varphi_n} & & \boxed{\Gamma\vdash\varphi_{n+1}} & \cdots & \boxed{\Gamma\vdash\varphi_{n+m}}
\end{tikzcd}$$

If there is no inference introducing the subgoals specified by the user within a single step the repair strategy tries to further reduce the goal in the current task, thus introducing further subgoals, until all specified subgoals are found. As in the assume case the antecedent of the sequent is untouched. If a subgoal matches a specified goal, it is not further refined. If all subgoals are found by a sequence of proof steps, these steps are abstracted to a single justification, which is by default propagated to the mediator.

Cases. The command **cases** reduces a task containing a disjunction on the left hand side of the task into $n+m$ subtasks where in each case an additional premise is added. As for the **subgoals** the user can leave out some of the cases. If the task does not contain a suitable disjunction, the repair strategy is executed, which tries to derive a desired disjunction by forward reasoning. The goal remains untouched. As for the subgoal case the sequence introducing the disjunction is abstracted to a single step.



Set. The command **set** is used to bind a meta-variable or to introduce an abbreviation for a term. If x is an unbound meta-variable in the proof state, **set** will instantiate this variable with the given term t. The substitution $x \to t$ is added to the proof state. If x is already bound, a failure is generated. If x does not occur in the proof state, the command **set** serves as a shortcut statement for the given term t. The formula $x = t$ will be added as a new premise to the task. The last case is shown on the right. Adding an equation $x = t$ with a fresh variable x as premise is conservative in the sense, that a proof using the new variable x can be converted in a proof without x by just substituting all occurrences of x by t. There is no repair strategy for the **set** command.



Trivial. The command **trivial** is used to indicate that a task is solved. This is the case if a formula φ occurs on both the left and the right hand side of the task, the symbol false occurs at top level on the left hand side of the task, or the symbol true occurs at top level on the right hand side of a task. A task can also be closed if the inference *name* is applied and all its premises and conclusions are matched to term positions in the current task. In case that the quickcheck fails the repair strategy tries to close the task by a depth limited forward backward reasoning.

Complex. The command **complex** is an abstract command which subsumes an arbitrary sequence of the previous commands. It is used to represent arbitrary abstract steps. Note that it is generally not possible to justify such a step with a single inference application, and without further information a blind search has to be performed to justify the step. Hence there is no quickcheck for **complex** . If however in the **by** slot a strategy is specified, this strategy needs to be executed and the result to be compared.

Example. Looking at our running example, the user wanted to show **Theorem 8** and stated already a partial proof (c.f. Table 3). As none of the proof checking rules for **assume** are applicable, the repair mode is started. The repair strategy tries to further refine the goal $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ to construct a situation in which the **assume** step is applicable. Indeed, after three refinements the proof step becomes applicable (see Figure 4). Unnecessary derivations, indicated by a dotted line in the Figure, are deleted. The repair process brought a second subgoal out that now can be lifted to the abstract proof view. We offer the following lifting modes: (i) *fix* : repaired proof fragments are automatically patched into the document (ii) *inform* : the author is informed about repair patches and decides their execution. (iii) *silent* : the author is only informed about errors, no repair patches are offered.

5.2 Proof Lifting

Whenever a part of the proof is changed, it must be propagated back to the mediator. In principle the prover can insert arbitrary large parts and multiple hierarchies during the

Table 5. Proof lifting rules

$\{\text{diff}(T,T_i) \mid T_i \in succ(T)\}$	proof step in abstract syntax
$\emptyset$	**trivial by** *name* **from** $\text{lab}(name)$
$\{\langle\{\varphi\},\{\xi\}\rangle\}$	**assume** φ **by** *name* **from** $\text{lab}(name)$
$\{\langle\emptyset,\emptyset\rangle\}$	**set** $\text{diff}(\sigma,\sigma')$
$\{\langle\emptyset,\{\xi_1\}\}\rangle\ldots\langle\emptyset,\{\xi_m\}\}\rangle\}$	**subgoals** $\xi_1,\ldots,\xi_m$ **by** *name* **from** $\text{lab}(name)$
$\{\langle\{x=t\},\emptyset\rangle\}$	**set** $x=t$
$\{\langle\{\varphi\},\emptyset\rangle\}$	**fact** φ **by** *name* **from** $\text{lab}(name)$
$\{\langle\{\varphi_1\},\emptyset\rangle\ldots\langle\{\varphi_n\},\emptyset\rangle\}$	**cases** $\varphi_1,\ldots,\varphi_n$ **by** *name* **from** $\text{lab}(name)$
$\{\langle\Gamma_1,\Delta_1\rangle,\ldots,\langle\Gamma_m,\Delta_m\rangle\}$	**complex** Δ_1 **under** $\Gamma_1,\ldots,\Delta_m$ **under** Γ_m

repair phase. As default the most abstract proof hierarchy is communicated as a proof script to the mediator. However, the mediator can ask for a more detailed or a more abstract version of the proof script. Given a selected proof hierarchy, each proof step has to be transformed into a command of the proof script language. This is done by a static analysis of the proof step.

Task Difference. Technically, a proof step is executed with respect to an agenda $\langle\{\underline{T_1},\ldots,T_n\},\sigma\rangle$ and results in a new agenda $\langle\{\underline{T_1'},\ldots,T_k',T_2,\ldots,T_n\},\sigma'\rangle$. The step has reduced the task T_1 to subtasks $succ(T)=\{T_1',\ldots,T_k'\}$. In a first analysis, only the differences between the tasks are analyzed, defined as follows:

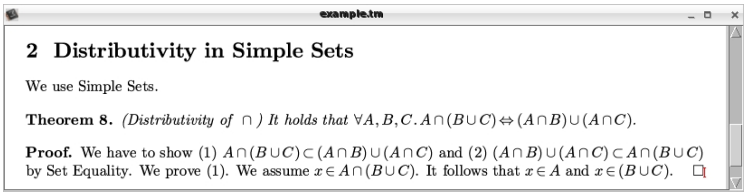
$$\text{diff}(T,T') = \langle\{\varphi\in\Gamma' \mid \varphi\notin\Gamma\},\{\xi\in\Delta' \mid \xi\notin\Delta\}\rangle$$

If a task is reduced to several subtasks, we obtain a set of differences for each subtask. Moreover, we require that the *name* of the applied proof operator, i.e. the inference or strategy, for the reduction is given and we denote a substitution introduced by the proof step with σ. We assume a function *lab* which returns the set of those labels which are used in premises and conclusions of the proof operator and **.** if none of them has a label.

Table 6. Proof repaired in abstract syntax

Proof.
subgoals
$A\cap(B\cup C)\subset(A\cap B)\cup(A\cap C)$: {
assume $x\in A\cap(B\cup C)$ **from .** ;
fact $x\in A\wedge x\in(B\cup C)$ **by . from .** ; ε}
$(A\cap B)\cup(A\cap C)\subset A\cap(B\cup C)$: { ε}
by Set Equality **from .**

Lifting Rules. If $succ(T)=\emptyset$, the proof step is translated into a **trivial** step. If $succ(T)=\{T'\}$, there are the following possibilities: If the task T and its successor task T' are the same, we analyze the difference between σ and σ' to obtain the formula $x=t$ needed for the **set** case. If the difference between T and T' is only one formula, and this formula has been added on the left hand side of the sequent, and is of the form $x=t$, where x is new, then the step is classified as a **set** case, otherwise as a **fact** step. If the new formula has been introduced as a goal, we classify the step to be a **subgoals** step. If several hypotheses are introduced, the step is classified to be a **cases** step. A formal definition of the proof lifting rules are given in



example.tm

2 Distributivity in Simple Sets

We use Simple Sets.

Theorem 8. *(Distributivity of* $\cap$ *) It holds that* $\forall A,B,C. A\cap(B\cup C)\Leftrightarrow(A\cap B)\cup(A\cap C)$.

Proof. We have to show (1) $A\cap(B\cup C)\subset(A\cap B)\cup(A\cap C)$ and (2) $(A\cap B)\cup(A\cap C)\subset A\cap(B\cup C)$ by Set Equality. We prove (1). We assume $x\in A\cap(B\cup C)$. It follows that $x\in A$ and $x\in(B\cup C)$. □

Fig. 5. Theorem with repaired partial proof

Table 5. Considering our running example, the abstract proof is repaired as shown in Figure 4, and presented to the author as illustrated in Figure 5.

6 Extending OMDoc for Authoring Verified Proofs

In ΩMEGA we use the OMDoc format already for theory repositories with acyclic theory dependencies, axioms, simple definitions and assertions. So far, we only support a subset of the OMDoc features and exclude e.g. complex theory morphisms and complex definitions. OMDoc's current proof module (PF) is designed for representing proofs given in a declarative or procedural proof language together with comments in natural language. Additionally, there is the possibility to store formal proofs as proof terms in `proofobject` elements. In the following we refer to the extension of the PF module proposed in [5] to store proofs with proofsteps on different levels of granularity. Moreover, as we use TeXMACS for authoring, we need to store additional *parsing* and *rendering* knowledge beyond the coverage of OMDoc's presentation module. For space reasons we use the compact form `<proof>...</>` for XML instead of `<proof>...</proof>`.

6.1 Hierarchical Proof Data Structure

As we require both, fast reconstruction of the PDS from OMDoc proofs and presentation of a given view of the proof, we need to store the proofs in OMDoc with all levels of granularity. As a simple example (with only one level of granularity) we show in Listing 1 the proof from Figure 4 in OMDoc format.

Each `proof` element represents a sequence of proofsteps. The proofstep consist of a `derive` element where the `type` attribute contains the proof command and the `method` element the **by** information, encoded in its `xref` attribute. The **from** information is stored in `premise` subelements which point to the corresponding labeled formulas. The labels are realized using the `id` attribute of the `OMOBJ` element.

An *assume* proofstep introduces always a new local hypothesis which is represented by the `hypothesis` element after the assume `derive` block. We represent a **subgoals** proofstep inside a `method` block by a sequence of `OMOBJ` - `proof` pairs corresponding to a subgoal followed by its subproof. We encode formulas in OMDoc using OPENMATH.

```
<proof xml:id="p1" for="#distr_inter">
  <derive xml:id="p1_d1" type="subgoals">
    <method xref="#definition_of_set_equality">
      <OMOBJ>A∩(B∪C) ⊂ A∩B∪A∩C</OMOBJ>
      <proof><derive xml:id="p1_d2" type="assume"><method xref="#definition_of_subset"/></>
             <hypothesis><FMP>x ∈ A∩(B∪C)</FMP></>
             <derive xml:id="p1_d3" type="fact"><FMP>x ∈ A∧x ∈ (B∪C)</FMP>
               <method xref="#definition_of_intersection"/></></>
      <OMOBJ>A∩B∪A∩C ⊂ A∩(B∪C)</OMOBJ></></></>
```

OMDoc Listing 1: Proof as XML tree

Finally, Listing 2 shows how we encode proof steps at different levels of granularity. The `alt` element contains as first `proof` element the strategy justification and then the expansion or refinement of this strategy. Table 7 gives an overview of the added attributes and content specifications.

```
<proof xml:id="p1" for="#distr_inter">
  <alt>
    <proof><derive xml:id="p1_dx" type="complex">
             <method xref="#strategy1"/></></>
    <proof> ... proof from Listing 1 ... </></></>
```

OMDoc Listing 2: Hierarchical Justifications

Table 7. Extension of PF

Element	Opt. Attrib.	Content
`proof`		`alt`
`alt`		`proof+`

Table 8. Extension of PRES

Element	Opt. Attrib.	Content
`use`	`cop`	`symbol*, OMOBJ*`

6.2 Parsing and Rendering Knowledge

The parsing and rendering facility of PLATΩ uses the following knowledge which we want to store for each theory and for each community of practice separately in OMDoc using the presentation module (PRES). Table 8 gives an overview of the extensions to the PRES module. Each knowledge item is encoded in a `presentation` block like

```
<presentation for="URI"><use format="texmacs" cop="name" attributes="type=TYPE">...</></>
```

Notations for mathematical symbols are given by `NOTC`. Typed variables are encoded as `symbol` elements and each pattern as an `OMOBJ` element[1]. While all patterns are allowed as parser input, the renderer uses by default the first pattern.

```
<presentation for="#union"><use ... attributes="type=symbol">&lt;cup&gt;</></>
<presentation for="#union"><use ... attributes="type=notation">
  <symbol xml:id="x"><type>set</type></symbol>
  <symbol xml:id="y"><type>set</type></symbol>
  <OMOBJ><OMA>
    <OMS cd="local" name="x"/><OMSTR>&lt;cup&gt;</><OMS cd="local" name="y"/></></></></>
```

Symbolgroups are given by `GROUP`. We declare a symbolgroup by using a `symbol` element before specifying its elements in a `presentation` block.

```
<symbol xml:id="setops" role="symbolgroup"/>
<presentation for="#setops"><use ... attributes="type=symbolgroup">
  <OMOBJ><OMS cd="th1" name="union"/></OMOBJ>
  <OMOBJ><OMS cd="th1" name="intersection"/></OMOBJ></></>
```

[1] We encode a list of terms as argument list of an `OMA` element.

Associativity information for symbols, given by `ASSOC`, with values *right* or *left*.

```
<presentation for="#union"><use ... attributes="type=associativity">left</></>
```

Precedence constraints for two or more symbol(group)s, expressing that $s_1 \prec \ldots \prec s_n$, are given by `PREC`. The symbol(group) s_1 is referred to by the `for` attribute and the $s_2, \ldots, s_n$ are encoded as `OMOBJ` elements.

```
<presentation for="#intersection"><use ... attributes="type=precedence">
  <OMOBJ><OMS cd="th1" name="union"/></OMOBJ></></>
```

By storing this knowledge separately for each community of practice, our architecture supports for free the automatic notational translation of a document across communities of practice. The default notation for all symbols is prefix notation.

7 Related Work

The most prominent system for the publication of machine checked mathematics is MIZAR [14] with one of the largest libraries of formalized mathematics. The language of the library is a well-designed compromise between human-readability and machine-processability. Since the MIZAR system is not interactive, the usual workflow is to prepare an article, compile it and loop both steps until there is no error reported. In contrast to that, our architecture allows for both a human-oriented and a machine-oriented representation as well as techniques to lift or expand these representations respectively.

ISABELLE/ISAR [17] is a generic framework for human-readable formal proof documents, both like and unlike MIZAR. The ISAR proof language provides general principles that may be instantiated to particular object-logics and applications. ISABELLE tries to check an ISAR proof, shows the proof status but does not patch the proof script for corrections. We try to repair detected errors or underspecifications in proof steps.

A very promising representative of distributed systems for the publication of machine checked mathematics is LOGIWEB [9]. It allows the authoring of articles in a sophisticated customizable language but strictly separates the input from the output document, resulting in the usual LaTeX workflow. By using the WYSIWYG text-editor TeXmacs we combine input and output representation in a document-centric approach.

Regarding parsing techniques the Matita system provides currently the best strategies for disambiguation [13]. Definitely, we plan to adapt these methods to our setting since they reduce efficiently the amount of alternative proofs to be verified.

8 Conclusion

In this paper we presented an architecture for authoring machine checked documents for mathematics repositories within a text-editor. To meet the real-time requirements of our scenario, we presented fast parsing and rendering mechanisms as well as incremental proof checking techniques. To increase the usability, the checking rules are enhanced by repair strategies trying to fix incomplete or underspecified steps. Finally, we presented an extension of the OMDOC format to store the formalized content of the document together with its justifications and notational information. Thus, the proof situations can be efficiently restored and verified at a later date and by other authors. With the approach presented in this paper we have a solid basis for further linguistic improvements. The plan is on the one hand to generate natural language with aggregation, topicalisation etc. from the controlled language and on the other hand to be able to understand that constantly increasing fragment of natural language to extract the controlled language. As future work we are going to investigate whether the proposed architecture is a well-suited foundation for collaborative authoring inside and across *communities of practice*.

References

1. Autexier, S., Benzmüller, C., Dietrich, D., Meier, A., Wirth, C.-P.: A generic modular data structure for proof attempts alternating on ideas and granularity. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 126–142. Springer, Heidelberg (2006)
2. Autexier, S., Dietrich, D.: Synthesizing proof planning methods and oants agents from mathematical knowledge. In: Borwein, J.M., Farmer, W.M. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 94–109. Springer, Heidelberg (2006)
3. Autexier, S., Fiedler, A., Neumann, T., Wagner, M.: Supporting user-defined notations when integrating scientific text-editors with proof assistance systems. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, Springer, Heidelberg (2007)
4. Autexier, S., Hutter, D., Mossakowski, T., Schairer, A.: The development graph manager MAYA. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422. Springer, Heidelberg (2002)
5. Autexier, S., Sacerdoti-Coen, C.: A formal correspondence between OMDoc with alternative proofs and the $\bar{\lambda}\mu\tilde{\mu}$. In: Borwein, J.M., Farmer, W.M. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 67–81. Springer, Heidelberg (2006)
6. Aycock, J., Horspool, N.: Directly-executable Earley parsing. In: Wilhelm, R. (ed.) CC 2001 and ETAPS 2001. LNCS, vol. 2027, p. 229. Springer, Heidelberg (2001)
7. Siekmann, J., et al.: Proof development with ΩMEGA: $\sqrt{2}$ is irrational. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 367–387. Springer, Heidelberg (2002)
8. Franke, A., Kohlhase, M.: MBASE: Representing mathematical knowledge in a relational data base. In: Systems for Integrated Computation and Deduction. Elsevier, Amsterdam (1999)
9. Grue, K.: The Layers of Logiweb. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, Springer, Heidelberg (2007)
10. Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006)
11. Lange, C.: A semantic wiki for mathematical knowledge management. Diploma thesis. IUB Bremen, Germany (2006)
12. Radzevich, S.: Semantic-based diff, patch and merge for XML-documents. Master thesis, Saarland University, Saarbrücken, Germany (April 2006)
13. Sacerdoti-Coen, C., Zacchiroli, S.: Spurious disambiguation error detection. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, Springer, Heidelberg (2007)

14. Trybulec, A., Blair, H.: Computer assisted reasoning with MIZAR. In: Joshi, A. (ed.) Proceedings of the 9th Int. Joint Conference on Artifical Intelligence. M. Kaufmann, San Francisco (1985)
15. J.v.d. Hoeven.: GNU $\TeX_{\text{MACS}}$: A free, structured, WYSIWYG and technical text editor. Number 39-40 in Cahiers GUTenberg (May 2001)
16. Wagner, M., Autexier, S., Benzmüller, C.: PLATΩ: A mediator between text-editors and proof assistance systems. In: Benzmüller, C., Autexier, S. (eds.) 7th Workshop on User Interfaces for Theorem Provers (UITP 2006), Elsevier, Amsterdam (2006)
17. Wenzel, M.: Isabelle/Isar — a generic framework for human-readable proof documents. In: Matuszewski, R., Zalewska, A. (eds.) From Insight to Proof — Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar, and Rhetoric, University of Bialystok (2007)