ΩMEGA – A Mathematical Assistant System

Jörg Siekmann, Michael Kohlhase, Erica Melis
Universität des Saarlandes, Saarbrücken, Germany

Abstract

Classical automated theorem provers can prove non-trivial mathematical theorems in highly specific settings. However they are generally unable to cope with even moderately difficult theorems in mainstream mathematics. While there are many reasons for the failure of the classical search-based paradigm, it is apparent that mathematicians can cope with long and complex proofs and have strategies to avoid less promising proof paths without suffering from the exponential search spaces. Consequently, a combination of the power of automated tools with human-like capabilities seems necessary to prove main-stream mathematical theorems with the help of a machine. In the following, we shall describe the prototypical system ΩMEGA that explores proof planning together with high-level proof tools. ΩMEGA is a mixed-initiative system with the ultimate goal of supporting theorem proving in main-stream mathematics and mathematics education.

Contents

Over time we become trapped in our shared visions of appropriate ways to tackle problems,and even more trapped by our funding sources where we must constantly justify ourselves by making incremental progress. Sometimes it is worthwhile stepping back and taking an entirely new (or perhaps very old) look at some problems and to think about solving them in new ways. This takes courage as we may be leading ourselves into different sorts of solutions that will for many years have poorer performance than existing solutions. With years of perseverance we may be able to overcome initial problems with the new approaches and eventually leapfrog to better performance. Or we may turn out to be totally wrong. That is where the courage comes in.

Rodney Brooks, MIT, AAAI-96

1 Introduction

Traditional automated theorem provers (ATP) are essentially black boxes, their input consists of the theorem to be proved, and (hopefully exactly) those axioms that are necessary to prove the theorem. Starting from this input and based on a fixed logic calculus such as resolution [Rob65], the system generates a search space at the level of this logic calculus. The search through this space is guided by general, syntactic heuristics, such as unit preference or set-of-support which have little resemblance to human mathematical reasoning about a proof.

If the system does not find a proof, then the user will adjust the settings of the ATP that determine the search heuristics and she might tweak the formulation of the theorem and axioms in the hope to find a proof eventually. In the 80ies, Woody Bledsoe characterized this kind of ATP in the following way:
Automated theorem proving ... is not the beautiful process we know as mathematics. This is 'cover your eyes with blinders and hunt through a cornfield for a diamond-shaped grain of corn'. [Ble86]

Now, this kind of approach might have worked out successfully as, e.g., the chess programs, which work essentially on similar principles, have demonstrated: in order to beat a world champion player in chess a program does not necessarily have to work on the same principles as the human player. But, alas, after more than forty years of research and development, automated reasoning systems can sometimes prove difficults mathematical theorems in highly specific settings – but generally still fail utterly when it comes to ordinary mainstream mathematics. Now, either we may need another forty years – or we happen to live in the wrong paradigm and tackle inappropriate research problems.

Mathematicians prove theorems differently: They follow an intuitive plan rather than blindly search for a proof and they structure their proof attempts accordingly. They start with some proof ideas which they refine later on.[1]

Rather than blindly exploring the whole search space, humans prefer the mathematically more promising paths and neglect the others. It seems that this control is largely knowledge-based. General proof patterns, such as that of the well-known diagonalization proofs, and reasons to apply a method contribute to the restriction of the search as well as theory-specific knowledge on how to proceed, say in an ϵ-δ-proof [Mel98].

Much of the reasoning in the proof discovery is at the meta level: "If we would have a certain result the next result may follow and then the next etc. Afterwards we have to fill in the details, and to check whether the plan really works", or "Since

[1] To quote the German mathematician Gerd Faltings who received the field medal for his solution to Mordell's Conjecture. When asked how he found the proof he said: "Man hat Erfahrungen, dass bestimmte Schlüsse unter bestimmten Voraussetzungen funktionieren... Man überlegt sich also im Groben: Wenn ich das habe, könnte ich das zeigen und das nächste. Hinterher muss man die Details einfügen und sieht, ob man es auch wirklich so machen kann". Gerd Faltings 1983

condition C is not true in general, we need a case split for $C \vee \neg C$" etc. Different from investigations on safe reflections, this meta-reasoning epitomized by Polya's work [Pol45] or Hadamard's "Psychology of Mathematical Invention" is often of heuristic nature.

General problem solving strategies such as analogy or the use of examples and counterexamples is employed to guide or restrict the search for a proof. Moreover, specialized (or theory-specific) mathematical techniques such as calculations and algebraic simplifications are rather common and restrict the search space by their algorithmical computation.

Furthermore, humans prove theorems in the context of one or more theories, i.e., rather than listing all needed axioms prior to the proof, they select an axiom by need from a large knowledge base. This helps to keep the explored search space small.

Apart from the proof discovery process, traditional automated theorem proving systems and human mathematical theorem proving strikingly differ in their result, namely the final proof and its presentation. While an automated theorem prover, say OTTER [McC90] outputs a sequence of (hyper)resolution and paramodulation steps, a typical presentation of a mathematical proof is more abstract and (hierarchically) structured.

For this and other reasons we departed from the calssical paradigm of automated theorem proving many years ago to try new alleys and this paper gives a first account and survey of our findings since then.

2 The Architecture and Components of ΩMEGA

Theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, plan execution and the call of external tools such as an ATP or a constraint solver and finally verification. This process is driven by the partially completed proof plan which is represented in the *Proof Plan Data Structure* ($\mathcal{PDS}$), a hierarchical data structure that represents a (partial) proof at different levels of abstraction (called partial *proof plans*). A $\mathcal{PDS}$ is a directed acyclic graph, where the nodes are labelled by the appropriate wff of the proof plan and justified by (LCF-style) tactic applications. Conceptually, each such justification represents again a proof plan (the *expansion* of the justification) at a lower level of abstraction that is computed when the tactic is executed[2]. In ΩMEGA, we keep the original proof plan explicitly in an expansion hierarchy. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation, the analogical transfer of plans, or its translation into a natural language representation.

Once a proof plan is completed, its justifications can successively be expanded to verify the well-formedness of the ensuing $\mathcal{PDS}$. This verification phase is necessary, since the correctness of the different components (in particular, that of the external ones) cannot be guaranteed. When the expansion process is carried out down to the underlying ND-calculus, the soundness of the overall system relies solely on the correctness of the verifier and of ND. This provides the basis for the controlled integration of external reasoning components, where each reasoner's results have to be be transformed (on demand) into a sub-$\mathcal{PDS}$.

A $\mathcal{PDS}$ can be constructed by automated or mixed-initiative planning or by pure user interaction using the integrated tools. In particular, new pieces of $\mathcal{PDS}$ can be added by directly calling tactics, by inserting facts from a data base, or

[2]Thus a proof plan can be recursively expanded, until we have reached a final proof plan, which can be executed to yield a calculus-level proof, since all nodes are now justified by the inference rules of a higher-order variant of Gentzen's calculus of natural deduction (ND).

by calling an external reasoner (cf. 3) such as an automated theorem prover or a computer algebra system.

Let us look at a concrete example (which was proposed by Woody Bledsoe in [Ble90] as a challenge to automated theorem provers) to get an intuition of the proof planning process in ΩMEGA. The example is a limit theorem which states that the limit of the sum of two real functions is equal to the sum of their limits.

> **Theorem (LIM+)**:*If* $\lim_{x \to a} f(x) = L_1$ *and* $\lim_{x \to a} g(x) = L_2$*, then* $\lim_{x \to a}(f(x) + g(x)) = L_1 + L_2$*.*

Similar theorems hold for multiplication, exponentiation, etc. of the limits which makes this example especially suited for our purposes. Since the limit of a function at a is defined by

$$\lim_{x \to a} h(x) = L \Leftrightarrow \forall \epsilon (0 < \epsilon \to \exists \delta (0 < \delta \wedge \forall x (x \neq a \wedge |x - a| < \delta \to |h(x) - L| < \epsilon))),$$

to prove LIM+ one needs to show that

$$\forall \epsilon (0 < \epsilon \to \exists \delta (0 < \delta \wedge \forall x (x \neq a \wedge |x - a| < \delta \to |(f(x) + g(x)) - (L_1 + L_2)| < \epsilon)))$$

holds under the assumptions:

$$\begin{array}{l} \forall \epsilon_1 (0 < \epsilon_1 \to \exists \delta_1 (0 < \delta_1 \wedge \forall x_1 (x_1 \neq a \wedge |x_1 - a| < \delta_1 \to |f(x_1) - L_1| < \epsilon_1))) \\ \forall \epsilon_2 (0 < \epsilon_2 \to \exists \delta_2 (0 < \delta_2 \wedge \forall x_2 (x_2 \neq a \wedge |x_2 - a| < \delta_2 \to |g(x_2) - L_2| < \epsilon_2))) \end{array}$$

This requires the construction of a real number δ for a given ϵ, which is why this part of mathematics is often called ϵ-δ-proofs. In a mathematical textbook, typically the following solution is given:

> For a given ϵ choose δ smaller than δ_1 and δ_2, while δ_1 and δ_2 are chosen according to the assumptions for ϵ_1 and ϵ_2 smaller or equal to $\epsilon/2$. Then for δ with $|x - a| < \delta < \delta_1, \delta_2$ we have
>
> $$|f(x) + g(x) - (L_1 + L_2)| \leq |f(x) - L_1| + |g(x) - L_2| < \epsilon_1 + \epsilon_2 \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

If we want to mimic this way of proving the limit theorem by proof planning,[3] we need several estimation *methods*. One is called `ComplexEstimate` (see section 5) and it uses the triangle inequality to reduce the estimation of the magnitude of a complex term b to the estimation of simpler ones (where a term b is represented as a linear combination $k \cdot a + l$). This decomposition of b can be oracled by a *computer algebra system* integrated into the proof planner. For limit theorems that involve particular functions, such as $\sqrt{}$ or sin, some rewrite theorems may have to be retrieved from the mathematical knowledge base in order to simplify and manipulate the terms involving these functions. For the construction of δ for a given ϵ various constraints are collected and propagated by an external *constraint solving system* for linear inequalities.

In the rest of the paper, we shall take a closer look at some of ΩMEGA's components. In the next section, we will present a general architecture for the integration of external reasoning systems, then we will have a look at the mathematical knowledge base, and finally, the proof planning process as sketched above will be made more concrete.

[3]While a proof for (a simplified formalization of) LIM+ has also been found by a specialized (hand-crafted) search heuristic in OTTER, LIM* and the other variants are way beyond the capability of traditional systems – that is why Bledsoe posed them as a challenge.

3 Integration of Mathematical Services

Logical reasoning and symbolic computation capabilities are provided by the first-order theorem provers BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER (see [SS97] for references), two higher-order theorem provers TPS [ABI+96] and $\mathcal{LEO}$ [BK98], and the computer algebra systems MAPLE, MAGMA, GAP, and μ $\mathcal{CAS}$ (see [KKS98] for references).

The reasoning modules interact in order to complete open subgoals in the development of a proof plan which can be initiated and supervised on-line by the user or it can be guided by the ΩMEGA system itself, for instance, during proof planning. Unfortunately, it is not always clear in advance, which prover is best suited for the problem at hand. Furthermore, the user could be asked to support the system with additional knowledge. Thus, ΩMEGA will call several 'services' in parallel in order to maximize the likelihood of success and minimize the time the user has to spend waiting for the system. The proprietary proofs found by these systems are finally transformed into the internal format of the ΩMEGA system; again, this transformation process should run in parallel to the ongoing user interaction.

Such an integrated mathematical assistant system and its application, for instance, in program verification [HLS+96], calls for an open and distributed architecture, where the developer of a deduction system or a mathematical tool upgrades it to a so-called *mathematical service* [HC96] by providing it with an interface to a common *mathematical software bus*. For the ΩMEGA system, we have implemented and experimented with such a network design, where the integrated theorem provers and mathematical tools are distributed over the Internet, they can be dynamically added to and subtracted from the coordinated reasoning repertoire of the complete system. This is achieved via a special software package called MATHWEB.

The MATHWEB [FK99, FHJ+99] system is an object-oriented tool-box that provides the functionality for building a society of software agents that render mathematical services as mentioned above: i.e., each deduction or computer algebra system is encapsulated into an agent. The functionality of the software bus functionality is realized in the current implementation by a model quite similar to the *Common Object Request Broker Architecture* (CORBA [Sie96]) in which a central *broker* agent provides routing and authentication information to the mathematical services (see [SHS98] for details). The agents are realized in the distributed programming system called MOZART which is a programming environment for the highly innovative programming language Oz and which provides the full infrastructure for distributed applications. Currently, we have integrated the following mathematical and reasoning services:

Automated Theorem Provers MATHWEB currently features the first-order theorem provers BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER (see [SS97] for references), and the higher-order systems TPS [ABI+96] and $\mathcal{LEO}$ [BK98]. Furthermore, there is a service `competitive-atp` that calls sets of ATP concurrently as competing services (this strategy is known to yield super-linear speedups in practice).

Computer Algebra Systems There are services wrapping the systems MAPLE, MAGMA, GAP, and $\mu\mathcal{CAS}$ (see [KKS98] for references). Here, the MATHWEB approach is particularly interesting, since a licensee of commercial software systems like MAPLE and MAGMA can export the corresponding services to the deduction community.

Constraint Solvers A general approach of integrating constraint solvers into proof planning [Mel98a] is realized with the solver LINEQII treating (linear) inequalities.

Mediators are mathematical services that transform mathematical knowledge from one format to another. The agent-oriented MathWeb approach allows to encapsulate the zoo of conversion programs currently available to generally available mathematical services and avoid thus duplication of efforts.

Proof Transformers are rather substantial mediators that transform between proof formats. Currently MathWeb features a proof transformation service from the proof formats of the theorem provers mentioned above [HF96, Mei99] into the natural deduction calculus.

Knowledge Bases MathWeb currently includes the MBase service, which will be described in more detail below. MBase is a web-based mathematical knowledge base system that stores mathematical facts like theorems, definitions and proofs and performs type checking, definition expansion, and semantic search. It communicates with other mathematical services by mediators and with the user via the interaction unit OctOpus.

Human Interaction Units are MathWeb services that provide visualization and control features for user interaction. Currently, MathWeb includes the $\mathcal{L}\Omega\mathcal{UI}$ graphical user interface for interactive theorem provers [SHB+98], the OctOpus front-end for MBase and the ProVerb proof presentation system [HF96], which transforms ND proofs into a more abstract representation and into natural language.

The MathWeb approach has been a key factor in keeping the system maintainable [SHS98, FHJ+99] and the near future will see further modularization and agentification of system components, which will lead to simpler system maintenance and a more open development model.

4 A Mathematical Knowledge Base System

Ωmega includes a mathematical knowledge base, MBase, for storing, browsing, and manipulating the factual and control knowledge, necessary for proving a theorem. Note that a knowledge base is not only needed as a knowledge repository for mathematical facts, theorems, and definitions, but also for the integration of software systems (or agents) via protocols. Indeed, a meaningful communication between such systems is only feasible if there is a common ontology (knowledge base) to which systems can relate in their communication. In a sense the common knowledge base gives the *semantics* for the integration.

The MBase system is a web-based, distributed knowledge base for mathematics that is accessible through MathWeb. The current implementation (V.0.1) is still a first prototype for the internal testing of the design decisions. It consists of the MBase server, which acts as a MathWeb service and OctOpus (a simple MBase browser), which acts as a MathWeb client. Other mathematical services – currently MBase supports connections to the Ωmega system [BCF+97] and to the induction provers InKa [HS96] and λ*Clam* [RSG98] – can access MBase through a system of *mediators* which are integrated into MBase.

For this overview we will describe the MBase server and its underlying data model. Since the system will eventually contain large amounts of data, a file-oriented library storage mechanism is currently implemented. Systems like Ωmega, λ*Clam*, or Isabelle [Pau94] load all mathematical knowledge in the library that is possibly relevant to a given problem into their main memory and this would result in a very inefficient usage of this resource.

4.1 Architecture of MBase

The general architecture is that of a Relational Data Base Management System (RDBMS), such as, e.g., Oracle, embedded into a mOZart process (yielding a MathWeb service). This architecture combines the storage facilities of the RDBMS with the flexibility of a an interactive and distributed implementation of the concurrent, logic-based programming language Oz [Smo95]. mOZart is a distributed implementation of Oz whose main strength comes from its network transparency, i.e., its full support of remote computations in the base language (lexical scoping, logical variables, objects), and its network awareness, i.e., the full control of the programmer over network operations, such as the choice between stationary and mobile objects (called Oz functors).

Most importantly for MBase, mOZart offers a mechanism called pickling, which allows for a limited form of persistence: Oz objects can be efficiently transformed into a pickled form, which is a binary representation of the (possibly cyclic) data structure. This can then be stored in a byte-string and efficiently read by the mOZart application effectively restoring the object. This feature makes it possible to represent complex objects like logical formulae as Oz data structures,[4] manipulate them in the mOZart engine, but at the same time store them as strings in the RDBMS. The functionality of MBase can be enhanced at run-time by loading remote functors as "Ozlets" (mOZart functors). For instance, complex data base queries that can be compiled into mOZart functors by the OctOpus client can be sent (via the Internet) to the MBase server and applied to the local data e.g. for specialized searching.

The current implementation (V.01.) of MBase uses the "Gnu database manager", a simple file-oriented database system, integrated into mOZart. We have extended this integration by transaction management and logging facilities needed to ensure consistency of a knowledge base. All of this will be extended after successful experimentation to a real RDBMS.

4.2 Logic of MBase

The choice of the logic is not really central to the issues discussed here but it leads to the problems of the infrastructure (in particular the mediators), which are reflected in the structure of the MBase system.

Logic formulae in MBase are implemented as Oz data structures and stored as Oz-pickles in the underlying DBMS. They are then loaded into the Oz engine and manipulated there. The logical language supported by MBase is a polymorphically typed record λ-calculus. It is a generalization of the ML-polymorphic λ-calculus and of the language used in Isabelle and Hashimoto & Ohori's polymorphic record calculus [Oho95]. At the moment, this is implemented as an ad-hoc generalization of [Oho95], but we will soon change to a HM(X)-scheme [SOW97] of polymorphism with constraints, of which our calculus is an instance.

Records allow for a clean formalization of mathematical structures, such as groups or fields and the polymorphism is needed to reuse definitions and theorems in the knowledge base and ensure a modular structure of the theory. Finally the mechanism of "kinds" adds to the practical expressivity of the polymorphism and is used in many theorem proving systems (such as λ*Clam*, Isabelle, and others).

Finally, the MBase logic supplies the infrastructure for a sorted λ-calculus (see [Koh94] for the general development of this language. Conceptually, sorts are unary predicates (corresponding to sets in mathematics) that are treated specially in the inference procedures (sorted matching and unification). This added structure leads to a more concise representation and a more guided search. In MBase, we

[4]The object-oriented, constraint-logic programming features make Oz an ideal choice for that.

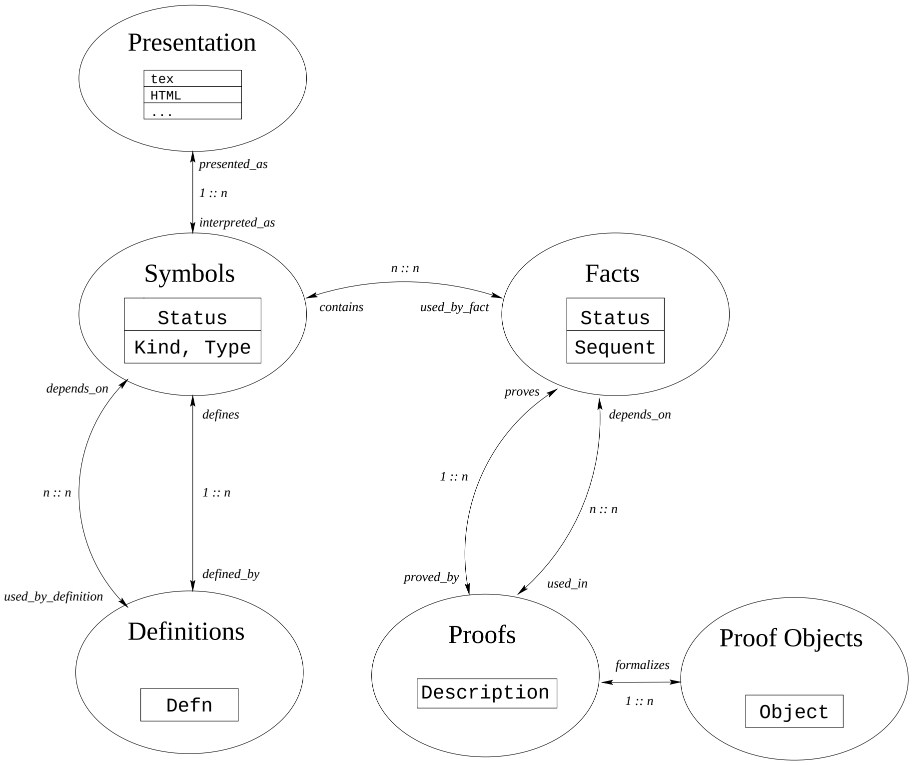

Figure 1: The structure of MBase

provide the infrastructure for a very general sort mechanism, but leave the implementation of sorted inference procedures to other mathematical services that act as MBase clients. For clients that cannot manipulate sorted logics, the mediators built into MBase can relativize the sorts.

4.3 The Database Model

We shall now briefly summarize the primary data base objects of MBase.

Symbols are used as usual for mathematical concepts, such as 1 for the natural number one, + for addition, = for equality, or `group` for the property of being a group. Furthermore, there are symbols for kinds, types and sorts.

Definitions give meanings to symbols in terms of already defined symbols. For example the number 1 can be defined as the successor of 0 (specified by the Peano axioms). Addition is usually defined recursively, etc. Definitions are separated from the symbols they define in MBase, since there can be more than one (equivalent) definition for a symbol in a mathematical theory.

A second reason for this separation is that constants can be introduced as symbols without definition.

Facts are axioms, theorems, conjectures, and lemmata. They all have the same structure as a logical sentence. The differences are largely *pragmatic* (theorems are normally more important in some theory than say, a lemma) or proof-theoretic (conjectures become theorems once there is a proof in the knowledge base).

Proofs are representations of evidence for the truth of **facts**. As in the case of definitions, there can be more than one proof for a given fact. Furthermore, it will be, at least initially, infeasible to totally formalize all mathematical proofs needed for the correctness management of the knowledge base in one universal proof format. Therefore MBase supports multiple formats for proofs or *evidence* such as e.g. a $\mathcal{PDS}$ (which is eventually the universal proof format), various proof scripts (Ωmega replay files, Isabelle proof scripts,...), references to published proofs, resolution proofs, matrix or tableau proofs, etc.

Proof Objects encapsulate the actual proof. There can be more than one proof object for a given proof. Informal proofs can be formalized, formal proofs can be transformed from one format to the other (e.g. from resolution style to natural deduction style), or can be presented in natural language. All the time, they represent the same "proof".

Presentation The presentation objects (see Figure 1) are not logical objects but they represent the presentation information for symbols in various presentation formalisms (such as ASCII, MathML [Ion98], LaTeX, HtML [Rag98],...), fonts or even natural language. It is a central concern of MBase to separate content information from presentation information, therefore, we have not included the presentation information into the symbol information itself. However presentation is such a central practical issue of knowledge bases that we have made the presentation objects primary.

5 Proof Planning in Ωmega

Our ultimate goal is to automate theorem proving as much as possible within the interactive Ωmega system. The central component of the system towards this end is the proof planner that treats proof problems as planning problems and integrates previously described mathematical services.

Proof planning was originally developed as an extension of tactical theorem proving [GMW79], where a tactic encapsulates a common sequence of calculus-level proof steps. The application of a tactic is controlled by the user. Now, Alan Bundy [Bun88] defined planning operators as tactics enhanced by pre- and postconditions which determine the applicability of such a "mathematical" operator. These operators are then used for a standard planning process as known from artificial intelligence. In order to cope with the new search space, a difference-reduction heuristic was encoded into the operator specifications to guide the search. Such a guidance works fine for proofs by mathematical induction, where the induction hypothesis is syntactically similar to the induction conclusion and where the induction conclusion has to be rewritten such that the induction hypothesis can be applied eventually. In mathematics in general, however, this is just one particular way/strategy to prove theorems.

Therefore, we extended proof planning to knowledge-based proof planning that employs

- domain-independent (logical) operators such as `CaseSplit`, and domain-dependent operators such as `ComplexEstimate`,
- domain- and situation-dependent control knowledge, and
- domain-specific external reasoners such as computer algebra systems or constraint solvers.

In the following, we describe the main ideas of knowledge-based proof planning.

A planning problem consists of an *initial state* describing some initial situation and of *goals* to be achieved. The planning domain is defined by the *operators* that usually represent actions, e.g. of a robot. A partial plan is a partially ordered set of steps, i.e. of (partially) instantiated operators with additional instantiation constraints. A partial plan can be seen as an implicit representation of a set of sequences (set of potential solutions) consistent with the ordering and instantiation constraints. A solution of a planning problem, a complete plan, is a fully instantiated linearization of a partial plan that transforms the initial state into a goal state, i.e., a state in which the goals hold.

The initial state in proof planning is a collection of sequents,[5] the proof assumptions, and the goal is the sequent to be proven. The proof planning domain consists of the operators which are called *methods* in this domain, of *control-rules*, and *theory-specific reasoners* such as the ones described above.

For example, the method `PeanoInduction` below has the object-level specification $\oplus$L1, $\oplus$L2, and $\ominus$L3, where $\oplus$L1 is an abbreviation for the sequent in proof line L1 in the *proof schema*. The annotations mean that, e.g., the sequent in L3 is deleted as a goal and the sequents in L1 and L2 are added as new subgoals.

method: `PeanoInduction`	
premises	$\oplus$L1, $\oplus$L2
conclusions	$\ominus$L3
appl.cond	sort(n) = Nat
proof schema	L1. $\vdash P(0)$ (baseCase) L2. $\vdash P(k) \rightarrow P(k+1)$ (stepCase) L3. $\vdash \forall n. P(n)$ (IndAxiom; L1,L2)

The *application conditions* (*appl.cond*) are formulated in a meta-language using decidable meta-predicates. They specify *local* and *legal* conditions for the method's application. For instance, in the above `PeanoInduction`, the *application condition* requires that n is a natural number. The slot *proof schema* provides the information for the schematic expansion of the method.

The method `ComplexEstimate` that is needed for many ϵ-δ-proofs looks as follows.

method: `ComplexEstimate`$_<(a, b, e_1, \epsilon)$	
premises	(0), $\oplus$(1),$\oplus$ (2), $\oplus$(3)
conclusions	$\ominus$ L12
appl.cond	$\exists\sigma. GetSubst(a,b) = \sigma$ **and** $\exists k,l. CASsplit(a\sigma, b) = (k,l)$ **and** $b = k * a\sigma + l$
proof schema	(0). $\Delta \vdash \lvert a\rvert < e_1$ () (1). $\Delta \vdash \lvert k\rvert \leq \mathbf{M}$ (OPEN) (2). $\vdash \lvert a\sigma\rvert < \epsilon/(2 * \mathbf{M})$ (OPEN) (3). $\Delta \vdash \lvert l\rvert < \epsilon/2$ (OPEN) L0. $\vdash b = b$ (Ax) L1. $\vdash b = k * a\sigma + l$ (`CAS`;L0) L2. $\vdash 0 < M$ (OPEN) L12. $\Delta \vdash \lvert b\rvert < \epsilon$ (fix;L1,L2,(0),(1),(2),(3))

Planning repeatedly refines a partial plan,i.e., it adds instantiated operators and constraints and thus restricts its set of potential solutions until a solution can be picked from its set of potential solutions. Since in proof planning most operators represent complex inference actions, they have to be expanded in order to obtain a

[5] A sequent is an object $(\Delta \vdash F)$, where F is a formula and the hypothesis Δ is a set of formulae. It denotes that F is derived from Δ.

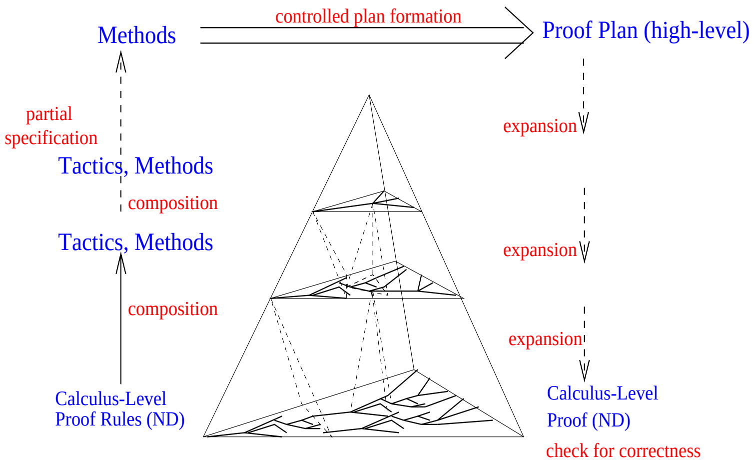

Figure 2: Proof plan data structure with expansions

proof at the calculus-level that can then be proof-checked. Therefore, an additional operation of the (hierarchical) planner is the expansion of complex operators to partial (sub)plans.

Control-rules express heuristics for the choice of methods, goals, or instantiations in the planning process. For example, the following rule `case-analysis-intro` expresses the heuristic that if the method `Rewrite` whose parameter is instantiated by a rule `(C -> R)` is not applicable because the formula `C` is not (trivially) provable, then a `CaseSplit` method should be introduced into the plan.

```
(control-rule case-analysis-intro
        (kind method-choice)
        (IF  (last-method (Rewrite (?C -> ?R))) AND
                 (failure-condition (trivial ?C)))
        (THEN (select (CaseSplit (?C or not ?C)))))
```

Proof planning methods usually correspond to well known mathematical tricks of the trade and the decision support for choosing these steps, the *control-rules*, often correspond to mathematical intuition about how to prove a theorem in a particular situation [MS98]. Control-rules are an appropriate means to express meta-level reasoning that is so common in mathematics and this declaratively represented control knowledge can express conditions for a decision that depends on the current planning state, the planning history, failed proof attempts, the current partial proof plan, the constraint state, the available resources, the user model, the theory in which to plan, typical models of the theory, etc [MS99].

6 Conclusion

The ΩMEGA system is an experiment in designing and integrating components for a mathematical assistant system. It is still under development and in a rather prototypical state. The components where we have made significant progress are: the MATHWEB architecture for distributed theorem proving by mathematical services, the MBASE knowledge base system, and the process of proof planning. This progress has extended the reasoning capabilities of the ΩMEGA system which is now well beyond the scope of monolithic automated deduction systems, as has been demonstrated, e.g., in the case of limit theorems (described in this paper) [Mel98], completeness proofs for resolution-based systems, diagonalization proofs, or optimization problems in an economics masters exam [KKS98].

References

[ABI+96] P.B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In William McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[BK98] Ch. Benzmüller and M. Kohlhase. LEO – a higher order theorem prover. *Proceedings of the 15th Conference on Automated Deduction*, Claude Kirchner and Hélène Kirchner (eds.), number 1421 in LNAI, Springer Verlag. pages 139–144, 1998.

[Ble86] W.W. Bledsoe. The use of analogy in automatic proof discovery. Tech.Rep. AI-158-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.

[Ble90] W.W. Bledsoe. Challenge problems in elementary analysis. *Journal of Automated Reasoning*, 6:341–359, 1990.

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE-9)*, Lecture Notes in Computer Science 310, pages 111–120, Argonne, 1988. Springer.

[FHJ+99] A. Franke, S. M. Hess, Ch. G. Jung, M. Kohlhase, and V. Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 1999. forthcoming.

[FK99] A. Franke and M. Kohlhase. System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving. CADE'99, 1999.

[GMW79] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. Lecture Notes in Computer Science 78. Springer, Berlin, 1979.

[HC96] K. Homann and J. Calmet. Structures for symbolic mathematical reasoning and computation. In Jacques Calmet and Carla Limogelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, LNCS 1128, pages 216–227, Karlsruhe, Germany, 1996. Springer Verlag.

[HF96] X. Huang and A. Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney (eds), *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, Springer Verlag, pages 221–225, 1996.

[HLS+96] D. Hutter, B. Langenstein, C. Sengler, J. H. Siekmann, W. Stephan, and A. Wolpers. Verification support environment. *High Integrity Systems*, 1(6), 1996.

[HS96] D. Hutter and C. Sengler. INKA - The Next Generation. In M.A. McRobbie and J.K. Slaney (eds), *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, Springer Verlag, pages 288–292, 1996.

[Ion98] Mathematical Markup Language (MathML) 1.0 specification. W3C Recommendation REC-MathML-19980407, World Wide Web Consortium, April 1998. Available at http://www.w3.org/TR/REC-MathML/.

[KKS98] M. Kerber, M. Kohlhase, and V. Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[Koh94] M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Universität des Saarlandes, 1994.

[McC90] W.W. McCune. Otter 2.0 users guide. technical reportANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.

[Mei99] A. Meier. Translation of automatically generated proofs at assertion level. Technical Report, Universität des Saarlandes, 1999.

[Mel98] E. Melis. The "limit" domain. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence in Planning Systems (AIPS-98)*, pages 199–206, 1998.

[Mel98a] E. Melis. Combining proof planning with constraint solving. In *6th CALCULEMUS and TYPES Workshop*, Eindhoven, The Netherlands, July 13–15 1998. Electronic Proceedings: http://www.win.tue.nl/math/dw/pp/calc/proceedings.html.

[MS99] E. Melis and J.H. Siekmann. Knowledge-based proof planning. submitted to *Artificial Intelligence*.

[MS98] E. Melis and J.H. Siekmann. Concepts in Proof Planning. In *Intellectics and Computational Logic. Papers in Honor of Wolfgang Bibel*, pages 249–264, Kluwer, 1999.

[Oho95] A. Ohori. A polymorphic record calculus and its compilation. *ACM Transactions on Programming Languages and Systems*, 17(6):844–895, 1995.

[Pau94] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS. Springer Verlag, 1994.

[Pol45] G. Polya. *How to Solve it*. Princeton University Press, Princeton, 1945.

[Rag98] HTML 4.0 Specification. W3C Recommendation REC-html40-19980424, World Wide Web Consortium, April 1998. Available at http://www.w3.org/TR/PR-xml.html.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12, 1965.

[RSG98] J. D.C. Richardson, A. Smaill, and I. M. Green. System description: Proof planning in higher-order logic with λ*clam*. *Proceedings of the 15th Conference on Automated Deduction*, Claude Kirchner and Hélène Kirchner (eds.), number 1421 in LNAI, Springer Verlag, 1998.

[SHB+98] J. Siekmann, S. Hess, Ch. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, and V. Sorge. A distributed graphical user interface for the interactive proof system OMEGA. In Roland C. Backhouse, editor, *User Interfaces for Theorem Provers*, number 98-08 in Computing Science Reports, pages 130–138, Department of Mathematics and Computing Science, Eindhoven Technical University, 1998.

[SHS98] M. Kohlhase S. Hess, Ch. Jung and V. Sorge. An implementation of distributed mathematical services. In *6th CALCULEMUS and TYPES Workshop*, Eindhoven, The Netherlands, July 13–15 1998. Electronic Proceedings: http://www.win.tue.nl/math/dw/pp/calc/proceedings.html.

[Sie96] J. Siegel. *CORBA: Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.

[Smo95] G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, LNCS 1000, pages 324–343. Springer-Verlag, Berlin, 1995.

[SOW97] M. Sulzmann, M. Odersky, and M. Wehr. Type inference with constrained types (extended abstract). In Benjamin Pierce, editor, *Proc. 4th Int. Workshop on Foundations of Object-Oriented Languages*, 1997.

[SS97] G. Sutcliffe and Ch. Suttner. The results of the cade-13 atp system competition. *Journal of Automated Reasoning*, 18(2):259–264, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.