Ω–MKRP
A Proof Development Environment

Xiaorong Huang Manfred Kerber Michael Kohlhase Erica Melis
Dan Nesmith Jörn Richts Jörg Siekmann
Fachbereich Informatik
Universität des Saarlandes
6600 Saarbrücken
Germany

Abstract

This report presents the main ideas underlying the Ω–MKRP-system, an environment for the development of mathematical proofs. The motivation for the development of this system comes from our extensive experience with traditional first-order theorem provers and aims to overcome some of their shortcomings. After comparing the benefits and drawbacks of existing systems, we propose a system architecture that combines the positive features of different types of theorem-proving systems, most notably the advantages of human-oriented systems based on methods (our version of tactics) and the deductive strength of traditional automated theorem provers.

In Ω–MKRP a user first states a problem to be solved in a typed and sorted higher-order language (called $\mathcal{POST}$) and then applies natural deduction inference rules in order to prove it. He can also insert a mathematical fact from an integrated database into the current partial proof, he can apply a domain-specific problem-solving method, or he can call an integrated automated theorem prover to solve a subproblem. The user can also pass the control to a planning component that supports and partially automates his long-range planning of a proof. Toward the important goal of user-friendliness, machine-generated proofs are transformed in several steps into much shorter, better-structured proofs that are finally translated into natural language.

This work was supported by the Deutsche Forschungsgemeinschaft, SFB 314 (D2, D3)

1 Introduction

The dream of machine assistance in proving mathematical theorems by far predates the advent of electronic computers: it is at least as old as Leibniz' idea of a *lingua characteristica universalis* and its corresponding *calculus ratiocinator*, whose alleged purpose was to solve mathematical and everyday problems stated in the *lingua universalis* by mere calculations (*"calculemus"* [30, 29]). This dream inspired logicians over the centuries and led at the end of the nineteenth century to a first realization in Frege's "Begriffsschrift" [16] (compare, e.g., [42, 18, 20, 17]).

Work toward this ideal was continued in early artificial intelligence (AI) research with the implementation of inference machines, which were among the first existing AI systems [35].

A theorem-proving system may be built with various purposes in mind. One goal is the construction of an autonomous theorem prover, whose strength achieves or even surpasses the ability of human mathematicians. Another may be the modeling of human problem-solving behavior on a machine, that is, cognitive aspects are the focus. A third purpose might be to build a system where the user derives the proof, with the system guaranteeing its correctness.

By and large, these goals have been investigated and implemented independently in different systems. We believe, however, that each of these facets should have its place in a general proof-development tool that can one day serve as an assistant.

With the Ω–MKRP-system we are taking a first step in this direction. In contrast to traditional automated theorem proving, our goal is not to replace a human mathematician completely, but, in the spirit of a tactics-based proof-checking environment, to relieve him from the more tedious part of his daily work. Unlike existing proof checkers, however, we intend to incorporate strong automated deductive tools.

The requirements for the Ω–MKRP-system have been derived from our experience in proving an interrelated collection of theorems of a typical textbook on semigroups and automata [12] with the first-order theorem prover MKRP [36]. The main conclusion we have drawn from these experiments is that although current automated theorem provers have obviously reached the power to solve non-trivial problems, they do not provide sufficient assistance for systematically proving interrelated mathematical theorems of a given mathematical field. Shaped by this experience, the objectives we initially had in our earlier work on the MKRP-system evolved, resulting in the Ω–MKRP-architecture.

In section 2 we will discuss the strengths and weaknesses of existing inference systems, and in section 3 we illustrate a typical theorem-proving session in Ω–MKRP. The architecture of Ω–MKRP is described in section 4, and we conclude this report with an outlook.

2 Strength and Limits of Existing Systems

Let us divide existing theorem-proving systems roughly into three categories: machine-oriented theorem provers, human-oriented (plan-based) theorem provers, and proof checkers.

2.1 Machine-Oriented Automated Theorem Provers

By machine-oriented theorem provers, we mean theorem provers based on computational logic theories such as resolution, paramodulation, or the connection method, i.e., systems based upon some computer-oriented inference system, and which derive their strength from their ability to maintain and manipulate very large search spaces (on the order of 10^4 clauses in the 1960s, several hundred thousands in the following decade; current systems handle search spaces of several million clauses and soon we shall witness systems whose capabilities are in the billions).

Such systems have been successfully applied in different fields of logic and mathematics (see, e.g., [43, chapters 9,10]). The strength of these systems is truly remarkable. For example, the theorem that an involution group (for all x, $x \circ x = e$) is commutative has become trivial for the MKRP-system. As another example, the theorem that a ring whose multiplication is idempotent is also commutative is not really difficult for the system (compare [39]). One's respect for such a system grows when one seeks for half an hour in vain for the perhaps once-known but long-forgotten proof. Harder and even open problems have been solved by such theorem provers.

On the other hand, observing the blind search behavior of such a system as it fails to solve a problem that seems trivial to us can be disappointing. What is missing is the mathematical knowledge and semantics that guide human search: for instance, function symbols such as "powerset" are treated in the same way as function symbols such as "successor", even in contexts a mathematician would never contemplate. Furthermore, although many applications of these systems are important and interesting (for instance, a test for the independence of the axioms of a given system, or their use in software and hardware verification), such applications differ substantially from the daily work of a mathematician when proving theorems. They are, rather, analogous to the tasks of a calculator or of a computer algebra system.

2.2 Experiences with MKRP

Throughout the development of the MKRP-system, testing was carried out on numerous theorems of a mathematical textbook on semigroups and automata [12]. During a time span of two years, about one third of the text book was fully encoded and finally proved by the MKRP-system. As the shortcomings of the system became more and more apparent, however, our ultimate goal of proving the entire book in a systematic way was then abandoned. This experience greatly influenced the conception and design of the Ω–MKRP-system. Below we discuss the strengths and weaknesses of the old MKRP-system, which are typical of existing fully automated theorem provers.

- The representation of the mathematical concepts in the sorted first-order input language of MKRP is often clumsy and unnatural. Since the concepts and constructs of a typical mathematics textbook such as [12] are mostly higher-order, we were forced to use sophisticated encoding techniques to translate them manually into the MKRP first-order input language. While the availability of sorts and the built-in equality predicate allow for a tolerably adequate translation (and without sorts and equality such an encoding is well-nigh impossible), it is not always obvious what the theorems proved by MKRP have to do with the textbook theorems and hence what is actually proven. The minimum that would be required for more accurate translations would be an automatic translation technique from higher-order to first-order logic.

Although a large proportion of the theorems could probably then be encoded into a first-order language with this transformational approach, some problems cannot adequately be handled this way, i.e., those that are truly higher-order.

Since the underlying logical language plays such an important role for the design of a system, we discuss our choice for Ω–MKRP's language in detail in section 4.

- The MKRP-system, as other current automated theorem provers, has no integrated *mathematical* knowledge. Each time definitions and lemmas are used as preconditions for the actual theorem, they must be coded and reinput. This is not only rather boring, but is also a serious source of error. The user is responsible for the correctness of the preconditions. Often (slightly) different formulations are chosen in different contexts, with the consequence that the correctness of the whole procedure of machine verification of textbooks is no longer assured. Moreover, the user may insert lemmas that cannot be proven in the given formulation. Discipline may be helpful—just as with enough discipline a modern operating system could in principle be developed in octal machine code—but as practice shows, automated assistance is indispensable. In short, a system that supports human mathematicians in proving theorems must include a database of mathematical knowledge that can be accessed and updated in a controlled way. This in itself is a major research task.

- More often than not, real mathematical theorems are too hard to be proven automatically. This state of affairs could be ameliorated by strengthening the deductive power of the prover in various ways: the integration of sorts; theory unification; sophisticated search strategies; better handling of equality. For every system, however, there exist theorems that cannot be shown automatically. In order to nonetheless use such a system, the user must be given the opportunity to guide the proof process interactively. In a classical theorem-proving system this is almost impossible: the cycle of interaction consists of a complete restart with a different setting of the parameters or a reformulation of the clauses. The main influence of the user is the appropriate choice and formulation of the problem. The way the preconditions of a theorem are selected, for instance, is of paramount importance for the performance of the system. Even if this is done optimally by giving a *minimal* set of preconditions, the system may still be unable to achieve a proof automatically. Sometimes a slight reformulation, different preconditions, or simply a reordering of the clauses may help. An additional necessary facility is one for splitting the proof into subproofs manually, so that they can be proved separately and then used as lemmas later in the proof of the original theorem. Traditional theorem provers lack such support and the situation is far from satisfactory, as all structuring decisions and all proof plans are hand-crafted. In short, all of this requires too much care and skill from the user, and not surprisingly there are fewer than a handful of well-known experts who are renowned for their skill in proving difficult theorems with the help of a machine.

- Traditional theorem provers like the MKRP-system operate on a normal form of formulas, usually the clausal normal form. It is a non-trivial task to ingest a long (several hundred steps) and complex proof in the resolution (or a similar) format. Hence, there is a need to present the proofs in a more intelligible way. This becomes even more important when the system is used as a mathematical assistant, where a user wants to read the proof in a language that corresponds to his own mode of representation. In particular, mathematicians untrained in the field of automated reasoning will not read cryptic clausal proofs, but expect polished and structured representations that highlight the essential steps and ideas of the proof.

Although automated theorem provers have solved difficult and open mathematical problems (see e.g. [43, chapter 9]), these problems are, generally speaking, relatively untypical of workaday mathematics. Today's automated theorem provers seem to outperform human mathematicians only in very technical and highly special domains, where there is very little (or almost no) human intuition, or where enormous syntactical calculations are the main hurdle. A collection of typical theorems of a mathematical text on the other hand, cannot, as a practical matter, be automatically proved. We believe therefore that the future is in systems that intimately interact with the user, so that the user can provide needed guidance, but in which the automatic component can carry out large parts on its own.

Further, automated theorem provers have to date been primarily employed to prove *single* theorems. If we prove *interrelated* theorems, such as they typically occur in a mathematical text, we encounter quite different problems.

In addition, an automated theorem prover does not benefit from its own experiences: once a problem is solved, its solution is forgotten, or at best stored in a protocol that is, unfortunately, not helpful to the system toward the proof of another problem.

In summary, traditional work on automated theorem proving has overemphasized the pure task of the mechanization of deductive inference. Although this is an essential ingredient of a mathematical assistant, there is far more to do, and considerable additional automated support is necessary in order to have a truly useful tool for developing and discovering proofs.

2.3 Interactive Systems

Interactive proof checking and proof development systems have been built with the aim of achieving a new standard of rigor in mathematical proof. As pointed out by Nicolaas Govert de Bruijn, the developer of one of the earliest systems (AUTOMATH), only a small part of mathematical literature today is absolutely flawless. To improve this situation, interactive proof checkers have been developed that carry out the meticulous final checking. In a similar spirit, proof development systems such as Nuprl were built, where the user essentially develops a proof and the system ensures that every step is correct. In future, this might result in new standards for the acceptance of mathematical papers: for each of his theorems an author would have to deliver a machine-readable proof that could be checked by the proof checking system of the editor [7, p.580]. For this purpose it is not important whether the author produces the proof himself or it was generated automatically. For simplicity, proof-checking system designers have as a first step offered interactive-only environments, where users must specify all details themselves.

For several reasons, AUTOMATH and most other comparable systems have not reached any broad acceptance as a working instrument for mathematicians. One reason is that there is a *loss factor* of 10 to 20 when using AUTOMATH. The loss factor expresses what is lost in brevity when translating ordinary mathematics into AUTOMATH. It is an important observation, however, that this loss factor is constant over the range of a book, that is, it does not increase (nor does it decrease) later in the book [7, p.603]. Although it is possible in principle to check a whole mathematical textbook with the system, this is a clumsy task. For instance, L.S. van Benthem Jutting was able to check Landau's "Grundlagen der Analysis" in the AUTOMATH-system [24], but it took more than five years. An important lesson learned from the AUTOMATH-approach is that the user-friendliness of such a system is important for its acceptance and performance. Consequently, in more recent systems like Nuprl [11], Muscadet [37], Isabelle [38], and IMPS [14], much more attention has been paid to the user. In particular, these systems are no longer mere interactive proof checkers but normally incorporate some human-oriented proof techniques that are encoded and represented in so-called tactics. Tactics are programs that manipulate the current state of the proof not only by the application of a single calculus step, but by a whole series of such steps. In this way one user interaction, namely the call of a single tactic, results in a sequence of steps. Nevertheless, while these systems are finding increasing acceptance and have also been demonstrated with remarkable success, there is one major objection. They incorporate far too little automated deductive support; the proof is found by the user with a little help from the machine, rather than vice versa: a lot of machine support with a little (conceptual) help from the user.

2.4 Human-Oriented Theorem Provers

Human-oriented theorem-proving systems have attracted growing attention after the initial enthusiasm for machine-oriented theorem provers died down and the limitations of later systems became more apparent. By human-oriented theorem provers, we mean systems that model human proof-search behavior, for example, by representing it as a planning process [9]. In contrast to machine-oriented logics, the object language is not a normalized first-order language such as clauses, but "a language closer to the one with which we ourselves describe the problem" [8, p.91]. Correspondingly, the calculus of such systems is not based on a machine-oriented formalism like resolution, but usually on some variant of a natural deduction calculus. Moreover, while the strength of traditional theorem provers essentially relies on fixed domain-independent search strategies, the deductive power of plan-based systems mainly resides in user-written domain-specific methods.

A successful realization of such a plan-based approach is the OYSTER-CLAM-system [9], where the Nuprl-tactics have been extended to so-called methods. A method can be viewed as a unit consisting of a procedural tactic *and* a declarative specification. The latter allows reasoning about methods and in particular allows for an automatic planning of proofs.

These techniques have been applied in particular to systems based on mathematical induction. For example, a large part of the heuristic knowledge of the Boyer-Moore prover [6] has been encoded into such methods.

Since the working language and the proof formalism are usually human-oriented, these systems also support user interaction in the spirit of a proof checker.

To summarize, a plan-based framework allows for a natural encoding of *domain-specific* problem-solving knowledge. However, the deductive strength of traditional theorem provers with their sophisticated *domain-independent* proof search strategies is not available.

2.5 Requirements for a Proof Development Environment

A comfortable environment for proving mathematical theorems is a computer-aided proof development system that acts like a CASE tool for mathematics. As such it should support at a minimum the positive features of each of the above three classes of systems. The following table gives an overview of the features of plan-based inference systems and standard automated theorem provers:

plan-based prover	traditional theorem prover
domain-dependent reasoning power	domain-independent reasoning power
sorted higher-order logic	first-order logic
interactive	automatic
communication of proof ideas	fixed proof strategies
mathematical knowledge	logical knowledge
human-oriented proof presentation	machine-oriented proof presentation

Taking account of the complementary structures of existing systems, our basic idea is to build a proof development environment, called Ω–MKRP, which combines the reasoning power of automated theorem provers as logic engines *with* the proof-planning paradigm. In particular, attention will also be paid to the support for a comfortable user interaction.

3 Illustration of a Concrete Problem-Solving Cycle

Let us look at an example to demonstrate how an interactive human-oriented theorem prover can be employed with an integrated machine-oriented theorem-proving system. As a first observation, mathematicians usually have a good intuition of what to do globally in a given situation, and they quite comfortably explain these domain-specific strategies to other mathematicians or to students. In contrast, they usually offer no explanation for the minute sequence of deductive steps that constitutes the final proof. Based on this observation, we assert that a mathematician can be best served by a system that supports the global planning and searching for a proof on a human-oriented interactive level, whereas the second task, filling in the details, is delegated to a logical engine such as a resolution-based theorem prover. Such a division of labor is advantageous, not least because it allows the user to provide as much of the problem-solving knowledge as he can, while the rest is left to the machine.

Now consider the following theorem (see [12, p.37]):
"*If ρ and σ are two equivalence relations, then the transitive closure of their union, $(\rho\cup\sigma)^{\mathrm{t}}$, is also an equivalence relation.*"
The following definitions are necessary for proving the theorem:

- definition of equivalence relation (in terms of reflexivity, symmetry, transitivity)
- definitions of reflexivity, symmetry, transitivity
- definition of union, $\forall a,b:S \quad (\rho\cup\sigma)(a,b) \iff \rho(a,b)\vee\sigma(a,b)$
- definition of transitive closure as $\rho^{\mathrm{t}} := \bigcup_{n\in\mathbb{N}} \rho^n$

- inductive definition of ρ^n as $\rho^1 = \rho$ and $\rho^{n+1} = \rho \circ \rho^n$.
- definition of composition of relations $\rho \circ \sigma$

A human proof plan for this theorem might look as follows:

1. The original problem can be decomposed quite naturally into three subproblems: $(\rho \cup \sigma)^{\mathrm{t}}$ is reflexive, $(\rho \cup \sigma)^{\mathrm{t}}$ is symmetric, and $(\rho \cup \sigma)^{\mathrm{t}}$ is transitive. Furthermore—and this observation is more sophisticated—a mathematician would rely on the heuristics that the three properties can be decoupled; in order to prove one property (reflexivity, symmetry, transitivity) of the transitive closure, the other two are not needed as preconditions.

1.1. The reflexivity of $(\rho \cup \sigma)^{\mathrm{t}}$ is trivial, because it contains ρ and σ. This subtheorem can be proved immediately by an automated theorem prover (MKRP did this, in fact). Note that at this level of minute logical details a mathematician has hardly any explicit proof strategies.

1.2. In order to show the symmetry of $(\rho \cup \sigma)^{\mathrm{t}}$, show the lemmas "*If ρ and σ are symmetric then $(\rho \cup \sigma)$ is symmetric.*" and "*If τ is symmetric then τ^n is symmetric for all $n \in \mathbb{N}$.*". The rest is trivial and should (and can) be done by a logic engine.

1.3. Show that for an arbitrary relation ρ, the transitive closure ρ^{t} is transitive. This requires showing by induction that if $\rho^n(a, b)$ and $\rho^m(b, c)$ then $\rho^{n+m}(a, c)$. The gaps should be filled in by an automated theorem prover (and in fact they were).

During the attempt to show the lemma "*If τ is symmetric then τ^n is symmetric for all $n \in \mathbb{N}$.*", a mathematician might notice that a further lemma, "$\tau \circ \tau^n = \tau^n \circ \tau$" is needed, which can be shown by mathematical induction. It is important to notice that it is quite common that not all necessary lemmas can be determined a priori, rather that some must be formulated during the proof process. This is particularly the case with so-called bridge lemmas [6].

As this example demonstrated, a supporting system must have at least the following features:

- The language should be higher-order, because there are abundant assertions about relations, natural numbers, functions on relations and so on. This language should be sorted, since this helps (as is well-known from first-order theorem proving) to structure the domain of discourse into sets of elements. Sorts thereby radically improve the search behavior of a system.
- Different types of logic engines are needed, including higher-order provers, induction-based provers and, since many subproblems can only be solved by first-order logic engines of a certain type, several of such engines.
- There should be a mechanism that allows the logic engines to generate new subproblems.

4 The Ω–MKRP-System

In this section we describe the key features of the Ω–MKRP-system, an interactive proof development environment that is based on the ideas presented in the previous sections.

4.1 The Architecture

Generally speaking, the Ω–MKRP-system is an integration of plan-based and machine-oriented theorem-proving systems with a strong emphasis on user interaction. The overall architecture is illustrated in figure 1. This is to be understood as follows: the current state of the proof under construction is represented in a data structure called the *proof tree*, representing a partial natural deduction proof [17]. To proceed, the user has the following options to manipulate the proof tree:

- He can insert mathematical facts contained in the Ω-DB (database) into the proof tree (e.g., as an already-proved premise). Furthermore, he can update the database with new definitions, theorems, etc.
- He can apply an existing method in the method-DB. Methods play the role of the tactics of proof checking systems. They generate a sequence of natural deduction steps that constitute a typical subproof. Faced with a new problem, the user can also create new methods or modify existing ones by invoking so-called meta-methods.
- In case a subproblem is such that it can be proved by one of the integrated logic engines, this subproblem is then handed over to the corresponding logic engine.
- After a successful run of a logic engine, the user may choose to initiate the automatic translation of the resulting machine-oriented subproof into the natural deduction format.
- Since the correctness of the various components, in particular of the user generated methods, are not generally guaranteed, the user must verify the final proof in terms of the inference rules of the natural deduction calculus by activating the verifier.

To further relieve the user, a planner can be invoked to propose a proof plan, that is, a structure composed of the operations mentioned above.

4.2 The Language

Since the technical mathematical language of a typical textbook is essentially a sorted higher-order logic augmented by many special-purpose representational constructs that are typical for the field at hand, we have developed an appropriate input language called $\mathcal{POST}$ [27, 33]. Since we are interested mainly in applications of standard mathematics, we also made the choice for *classical* higher-order logic as opposed to a nonstandard logic such as intuitionistic logic. In particular, our logic is built on Alonzo Church's simple theory of types [10] (an excellent introduction to classical higher-order logic can be found in [1]), but enriched by sorts (in the same sense that first-order logic is extended to sorted first-order logic).

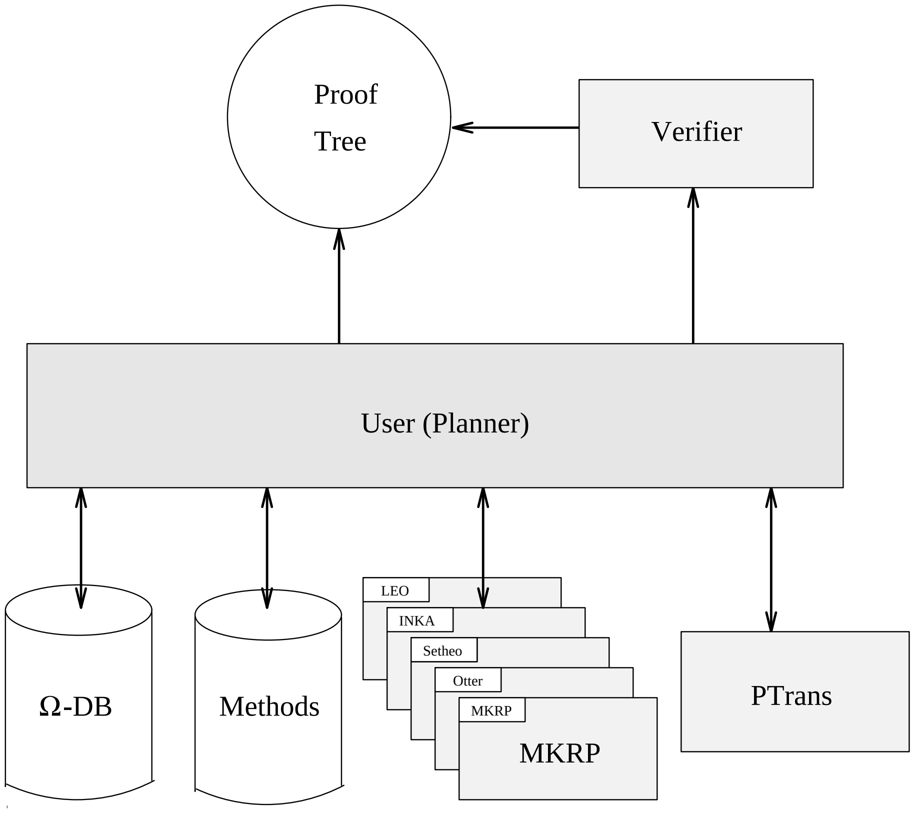

Figure 1: Architecture of Ω–MKRP

The input languages of classical automated theorem provers are usually variants of classical first-order logic. Most mathematicians agree that first-order logic is in principle sufficient to express common mathematical properties (Hilbert's Thesis [2, p.41]). The standard idea is to build the common concepts on a set theory like that of Zermelo-Fraenkel [44, 15] or von Neumann-Gödel-Bernays [34, 19, 3] as proposed in [5] and carried out in [40]. In practice, however, mathematicians use set theory only in a naive way as one paradigm among many others. Moreover, building an automated theorem prover on top of a set theory is not without complications, even if the more suitable formulation of set theory in [40] is used, as there are far too many set-theoretic inference steps possible at each stage of the search space that have little to do with the problem at hand.

Our main reason for choosing a sorted and typed higher-order logic (and not set theory) is based on the experience that the straightforward use of the MKRP-system for set theory was computationally too costly, because the key notion of a function must be defined in set theory as a left-total, right-unique relation, a relation as a subset of the Cartesian product of two sets and so on. In higher-order logic these notions are first-class primitive objects. In addition, sorted higher-order logic in many cases can be straightforwardly translated to first-order logic and often this is adequate [26]. Furthermore, higher-order unification, which is the crucial operation for efficient theorem proving in unsorted higher-order logic,

No	S;D		Formula	Reason
1.	1;	$\vdash$	$\mathrm{Eqrel}(\sigma)$	(Hyp)
2.	2;	$\vdash$	$\mathrm{Eqrel}(\rho)$	(Hyp)
3.	1;	$\vdash$	$\mathrm{ref}(\sigma) \wedge \mathrm{symm}(\sigma) \wedge \mathrm{trans}(\sigma)$	(Def-Eqrel 1)
4.	2;	$\vdash$	$\mathrm{ref}(\rho) \wedge \mathrm{symm}(\rho) \wedge \mathrm{trans}(\rho)$	(Def-Eqrel 2)
5.	5;	$\vdash$	$\forall \tau_{\blacksquare} \forall \mu_{\blacksquare} \forall x_{\blacksquare} (\tau \cup \mu)(x) \iff (\tau(x) \vee \mu(x))$	(Def-Union)
97.	97;	$\vdash$	$\mathrm{ref}((\sigma \cup \rho)^{t})$	(PLAN)
98.	98;	$\vdash$	$\mathrm{symm}((\sigma \cup \rho)^{t})$	(PLAN)
99.	99;	$\vdash$	$\mathrm{trans}((\sigma \cup \rho)^{t})$	(PLAN)
Thm.	1,2;5	$\vdash$	$\mathrm{Eqrel}((\sigma \cup \rho)^{t})$	(Def-Eqrel 97 98 99)

Figure 2: Example of a partial proof

can be generalized to sorted higher-order unification [28].

4.3 Proof Format and Methods

In addition to the problem formulation language, the proof format is also crucial for an adequate interface. As a common proof format for both the user and the system, we have chosen a generally established natural deduction formalism [17]. In figure 2 we present a snapshot of a partial proof for the example introduced in section 3 above: not all definitions and lemmas necessary for proving this problem are already included in the partial proof at this point.

As argued above, an advanced tool for proving mathematical theorems should support the user in communicating his proof strategies to the system. In our case, the user can guide the search for a proof by providing high-level proof plans [9], while the remaining gaps in the plan are filled in by a planning component or by the underlying logic engines. These proof plans are the essence of the knowledge that constitutes a mathematical field, such as the completeness proofs for resolution-based strategies that are typically shown first at the ground level by induction on what is known as the excess-literal number and then by lifting.

More concretely, in Ω−MKRP we basically follow the framework proposed by Alan Bundy et al. for the planning of proofs. Bundy's method concept provides a general framework for proof planning. It is, however, too restricted and rigid in the following way. The deductive power of human beings relies to an extraordinary extent on their ability to adapt known proof techniques to new and original situations. That is, methods have to be modified to the given situation. In Ω−MKRP this is done by meta-methods. In this sense, analogical theorem proving, for example, plays a role. To enable the modification of methods by meta-methods, we extend the method structure by splitting the tactic part of a method into two parts: a declarative part containing the proper information and a procedural part that interprets the declarative part. The declarative part is accessible for modifications (for details see [23, 22, 32]).

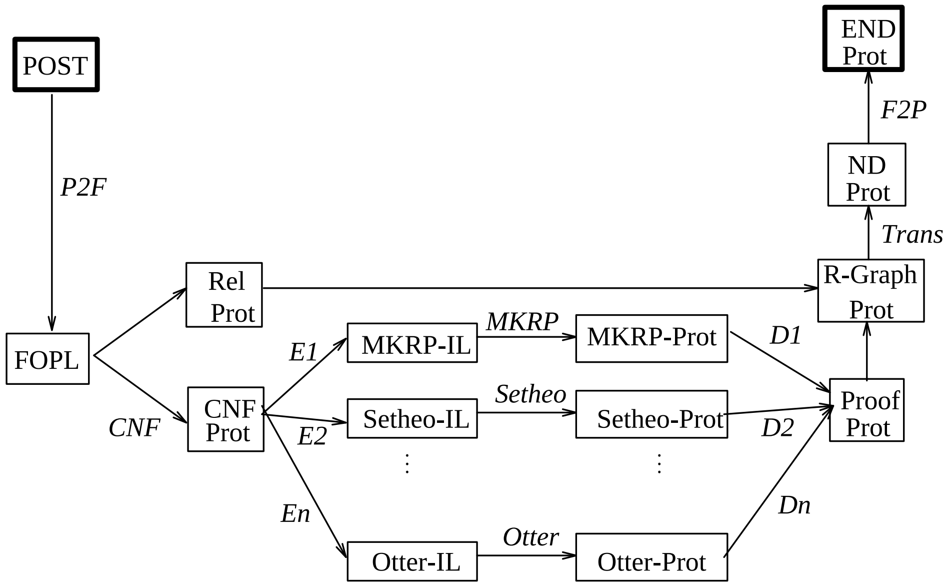

Figure 3: Integration of first-order theorem provers in Ω–MKRP

4.4 A Problem-Solving Cycle in Ω–MKRP

A problem-solving cycle in our system goes as follows: The user formulates his problem in $\mathcal{POST}$. In order to solve the problem, he may load definitions and already-proved theorems from the database, he may invoke a method to split the original problem by hand or let a planner do this job, or he may pass a subproblem to one of the logic engines. When the subproblem is within the capability of the logic engine, a corresponding proof will be found and translated back into the interface format. Then the user can continue with the above cycle.

There is a gap between the problem formulation language $\mathcal{POST}$ and the proof format, on the one hand, and the input and output languages of the underlying first-order theorem provers, on the other hand. To integrate first-order theorem provers, we have to bridge this gap by transforming $\mathcal{POST}$ into the input languages of the underlying provers and translating the output proofs (such as a clause proof of a resolution-based system) back into natural deduction proofs. An overview of the problem solving cycle with a first-order theorem prover in Ω–MKRP is illustrated in figure 3.

In order to pass a subproblem to a first-order theorem prover the problem must be first translated to first-order logic [25], then normalized into clausal normal form and after a successful run of the prover, the clause proof is normalized into a so-called refutation graph [13]. Out of this graph a first-order natural deduction proof can be generated [31], which in turn can be retranslated into a higher-order natural deduction proof.

To illustrate the problem solving cycle, let us return to the example introduced in section 3. With the partial proof given in figure 2 as the current state, the user may now choose to load the definition of reflexivity, which results in the insertion of line 6 in figure 4. Now the user believes that the plan line 97 can be proved directly by calling the logic

No	S;D		Formula	Reason
1.	1;	$\vdash$	$\mathrm{Eqrel}(\sigma)$	(Hyp)
2.	2;	$\vdash$	$\mathrm{Eqrel}(\rho)$	(Hyp)
3.	1;	$\vdash$	$\mathrm{ref}(\sigma) \wedge \mathrm{symm}(\sigma) \wedge \mathrm{trans}(\sigma)$	(Def-Eqrel 1)
4.	2;	$\vdash$	$\mathrm{ref}(\rho) \wedge \mathrm{symm}(\rho) \wedge \mathrm{trans}(\rho)$	(Def-Eqrel 2)
5.	5;	$\vdash$	$\forall \tau_{\blacksquare} \forall \mu_{\blacksquare} \forall x_{\blacksquare} (\tau \cup \mu)(x) \iff (\tau(x) \vee \mu(x))$	(Def-Union)
6.	6;	$\vdash$	$\forall \tau_{\blacksquare} \mathrm{ref}(\rho) \iff (\forall a_{\blacksquare} \tau(a, a))$	(Def-ref)
97.	1,2,5,6,...;	$\vdash$	$\mathrm{ref}((\sigma \cup \rho)^t)$	(MKRP)
98.	98;	$\vdash$	$\mathrm{symm}((\sigma \cup \rho)^t)$	(PLAN)
99.	99;	$\vdash$	$\mathrm{trans}((\sigma \cup \rho)^t)$	(PLAN)
Thm.	1,2;5	$\vdash$	$\mathrm{Eqrel}((\sigma \cup \rho)^t)$	(Def-Eqrel 97 98 99)

Figure 4: Extended example of a partial proof

engine MKRP. After a successful MKRP run, plan line 97 will be converted into the proved line 97 with MKRP as justification. At this point the user has the option to call a program PTrans that generates a natural deduction proof from the clausal proof found by MKRP. In this case line 97 will be replaced by the subproof thus generated. After the completion of the entire proof, the system restructures and substantially shortens (i.e., abstracts) the natural deduction proof thus found and the final presentation is in natural language [21].

5 Outlook

Ω–MKRP is currently under development. While the overall framework is set up and implemented, we are gradually integrating the other components. The transformation mechanism and the first logic engine, (the MKRP-system), are already incorporated into Ω–MKRP. Other systems will follow, in particular logic engines for higher-order logic and mathematical induction (INKA [4]). A database with many of the mathematical facts of [12] has been developed on top of a frame-like mathematical knowledge base (see [41] for details). Two of the most challenging problems of the whole project will be the realization of the planning framework and the declarative approach to methods.

With the Ω–MKRP-system we are orienting our project in the direction outlined above. We believe that this will result in a system that finally measures up to the standards of a mathematical assistant, the goal toward which we have worked for the last decade.

Acknowledgement

The ideas for Ω–MKRP developed out of our common sense of frustration with a standard first-order system such as MKRP and developed in a series of discussions, in which many people took part. We wish to thank all of them, in particular Peter Andrews, Bill Farmer, Detlef Fehrer, Dieter Hutter, Christoph Lingenfelder, Andreas Nonnengart, Hans Jürgen Ohlbach, Axel Präcklein, and Christoph Weidenbach.

References

[1] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic Press, Orlando, Florida, USA, 1986.

[2] Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *Handbook of Mathematical Logic*, chapter A.1, pages 5–46. North-Holland Publishing Company, Amsterdam, The Netherlands, 1977.

[3] Paul Bernays. A system of axiomatic set-theory. *Journal of Symbolic Logic*, **6**:1–17, 1941.

[4] Susanne Biundo, Birgit Hummel, Dieter Hutter, and Christoph Walther. The Karlsruhe induction theorem proving system. In Jörg H. Siekmann, editor, *Proceedings of the 8th CADE*, pages 672–674, Oxford, United Kingdom, July 1986. Springer Verlag, Berlin, Germany.

[5] Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, **2**:287–327, 1986.

[6] Robert S. Boyer and J Strother Moore. *A Computational Logic.* Academic Press, New York, USA, 1979.

[7] Nicolaas Govert de Bruijn. A survey of the project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, London, United Kingdom, 1980.

[8] Alan Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, London, United Kingdom, 1983.

[9] Alan Bundy. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th CADE*, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany. LNCS 310.

[10] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, **5**:56–68, 1940.

[11] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.

[12] Peter Deussen. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher.* Springer Verlag, Berlin, Germany, 1971.

[13] Norbert Eisinger. *Completeness, Confluence, and Related Properties of Clause Graph Resolution.* Pitman, London, United Kingdom, 1991.

[14] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 653–654, Kaiserslautern, Germany, July 1990. Springer Verlag, Berlin, Germany. LNAI 449.

[15] Adolf Abraham Fraenkel. Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre. *Mathematische Annalen*, **86**:230–237, 1922.

[16] Gottlieb Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879. reprint in: Begriffsschrift und andere Aufsätze, J. Angelelli, editor, Hildesheim. See also in Logiktexte, Karel Berka, Lothar Kreiser, editors, pages 82–112.

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

[18] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, **37**:349–360, 1930.

[19] Kurt Gödel. *The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, New Jersey; eighth printing 1970, 1940.

[20] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Sci. Lett. Varsovie, Classe III sci. math. phys.*, **33**, 1930.

[21] Xiaorong Huang. Reference choices in mathematical proofs. In Luigia Carlucci Aiello, editor, *Proceedings of the 9th ECAI*, pages 720–725, Stockholm, Sweden, 6-10th August 1990. Pitman, London, Great Britain.

[22] Xiaorong Huang. An explanatory framework for human theorem proving. In Hans Jürgen Ohlbach, editor, *Proceedings of GWAI-92*. Springer Verlag, Berlin, Germany, 1992.

[23] Xiaorong Huang, Manfred Kerber, and Michael Kohlhase. Methods – the basic units for planning and verifying proofs. SEKI Report SR-92-20, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1992.

[24] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System*, volume 83 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, Netherlands, 1979.

[25] Manfred Kerber. How to prove higher order theorems in first order logic. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th IJCAI*, pages 137–142, Sydney, 1991. Morgan Kaufman, San Mateo, California, USA.

[26] Manfred Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.

[27] Michael Kohlhase. Order-sorted type theory I: Unification. SEKI Report SR-91-18 (SFB), Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1991.

[28] Michael Kohlhase. Unification in order-sorted type theory. In A. Voronkov, editor, *Proceedings of LPAR*, pages 421–432, Berlin, Germany, 1992. Springer Verlag. LNAI 624.

[29] Gottfried Wilhelm Leibniz. Projet et essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer. In Karel Berka and Lothar Kreiser, editors, *Logiktexte*, chapter I.2, pages 16–18. Akademie-Verlag, german translation, 1983, Berlin, Germany, 1686.

[30] Gottfried Wilhelm Leibniz. Historia et commendatio linguae characteristicae universalis quae simul sit ars inveniendi et judicandi. German translation in Gottfried Wilhelm Leibniz: Gott – Geist – Güte: Eine Auswahl aus seinen Werken, pages 163–173, Bertelsmann Verlag, Gütersloh, Germany, 1947, not dated.

[31] Christoph Lingenfelder. Structuring computer generated proofs. In N.S. Sridharan, editor, *Proceedings of the 11th IJCAI*, pages 378–383, Detroit, Michigan, USA, 1989. Morgan Kaufman, San Mateo, California, USA.

[32] Erica Melis. Metamethods in analogical theorem proving. SEKI Report in preparation, 1993.

[33] Dan Nesmith, editor. KEIM-Manual. Version 0, 1992. Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany.

[34] John von Neumann. Die Axiomatisierung der Mengenlehre. *Mathematische Zeitschrift*, **27**:669–752, 1928.

[35] Allen Newell, Cliff Shaw, and Herbert Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings of the 1957 Western Joint Computer Conference*, New York, USA, 1957. McGraw-Hill. reprinted in Computers and Thought, Edward A. Feigenbaum, Julian Feldman, editors, New York, USA, 1963.

[36] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl Refutation Procedure. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, chapter 2, pages 41–112. MIT Press, Cambridge, Massachusetts, 1991. also as SEKI Report SR-89-19, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1989.

[37] Dominique Pastre. Muscadet: An automated theorem proving system using knowledge and metaknowledge in mathematics. *Artificial Intelligence*, **38**(3):257–318, 1989.

[38] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, pages 361–386, 1990.

[39] Axel Präcklein, editor. The MKRP User-Manual. SEKI Working Paper SWP-92-03, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1992. Second edition.

[40] Art Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, **8**(1):91–146, 1992.

[41] Barbara Schütt and Manfred Kerber. A mathematical knowledge base for proving theorems in semigroup and automata theory, part I. SEKI Working Paper in preparation, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1993.

[42] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, Great Britain; second edition, 1910.

[43] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning – Introduction and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1984.

[44] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre. I. *Mathematische Annalen*, **65**:261–281, 1908.