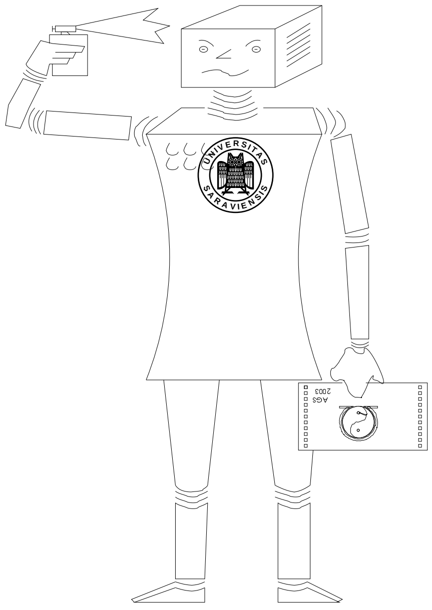
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/



SEKI Report

ISSN 1437-4447

Systems for Integrated Computation and Deduction – Interim Report of the CALCULEMUS Network

Christoph Benzmüller (Ed.)

`chris@ags.uni-sb.de`

FR 6.2 Informatik, Universität des Saarlandes

Saarbrücken, Germany

SEKI Report SR-03-05

Systems for Integrated Computation and Deduction – Interim Report of the CALCULEMUS Network.

Christoph Benzmüller (Ed.)

FR 6.2 Informatik
Universität des Saarlandes, Saarbrücken, Germany
`chris@ags.uni-sb.de`

Abstract

This document reports on the research progress made in all work task of the CALCULEMUS IHP Training Network HPRN-CT-2000-00102 after the first half of the 48 months funding period.

The objectives of the CALCULEMUS Network are:

1. outline the design of a new generation of mathematical software systems and computer-aided verification tools;
2. the training of young researchers in the broad field of mechanical reasoning and formal methods;
3. the dissemination of the results both in industry and in academia; and
4. the cross-fertilisation and amalgamation of the automated theorem proving (ATP/DS), computer algebra (CAS), term rewriting systems (TRS) interactive proof development systems (ITP) and software engineering (SE) research communities.

The work tasks of the Network are:

Task 1.1: Mathematical Frameworks

Task 1.2: Definition of Mathematical Service

Task 2.1: Integration of CASs and DSs via Protocols

Task 2.2: Enhancing the Reasoning Power of Computer Algebra Systems

Task 2.3: Enhancing the Computation Power of Deductions Systems

Task 3.1: Automated Support to Writing Mathematical Publications

Task 3.2: Support to the Development of an Industrial-Strength Application of Formal Methods to Program Verification

Task 3.3: Support to the Solution of Undergraduate Exam in Calculus and Economics

Task 3.4: Modelling of Existing Systems as Mathematical Services

Task 3.5: Challenge Mathematical Problems

Contents

Contributors of this Documents

This document contains a collection of contributions by all CALCULEMUS Network nodes. The network is formed by research teams at

- The Task Leaders, Scientists in Charge and Senior Researchers

  UED Alan Bundy, Ewen MacLean

  UKA Jacques Calmet, Regine Endsuleit

  RISC Bruno Buchberger, Wolfgang Windsteiger, Tudor Jebelean

  TUE Arjeh Cohen, Henk Barendregt, Herman Geuvers, Freek Wiedijk

  ITC-IRST Fausto Giunchiglia, Roberto Sebastiani, Marco Bozzano, Alessandro Cimatti

  UWB Andrzej Trybulec, Czeslaw Bylinski, Grzegorz Bancerek

  UGE Alessandro Armando, Enrico Giunchiglia

  UBIR Manfred Kerber, Volker Sorge

  USAAR Jörg Siekmann, Christoph Benzmüller, Serge Autexier

- The Young Visiting Researchers

  UED Corrado Giromini, Luca Compagna, Jürgen Zimmer, Julien Musset

  UKA Simon Colton, Jesus Maria Aransay Azofra, Julien Musset

  RISC Adrian Craciun

  TUE Scott Murray, Markus Rosenkranz, Mariusz Giero

  ITC-IRST Artur Kornilowicz, Gilles Audemard, Daniel Sheridan

  UWB Josef Urban, Hazel Duncan, Markus Moschner

  UGE Jürgen Zimmer, Sorin Stratulat, Pierre Ganty

  UBIR Martin Pollet

  USAAR Andrew Adams, Corrado Giromini, Pasquale De Lucia, Markus Moschner, Silvio Ranise

Introduction

The main research objective of the CALCULEMUS Network is to foster the integration of deduction systems (DS) and computer algebra systems (CAS) both at a conceptual and at a practical level. The point of origin for this kind of research is a landscape of heterogeneous approaches and systems on both sides of the spectrum, where the diversity on the DSs side is probably greater than on the side of CASs.

Since its start in September 2000 the CALCULEMUS Network has contributed to the convergence of DSs and CASs through its research on unifying frameworks for encoding and combining computation and deduction, the identification of the architectural requirements for a new generation of reasoning systems with combined reasoning and computational power, and the prototypical implementation and application of the improved systems. However, a single predominant theoretical framework is currently not possible. Such an approach would particularly involve predominant solutions to the still rather diverging systems at both sides of the spectrum between DSs and CASs.[1] Therefore a strong line of research in the Network focuses on the modelling and integration of CASs and DSs at the systems layer. In this research direction, significant progress has been made and several systems of project partners and other research institutes have been connected in order to form networks of cooperating mathematical service systems. The benefits and impacts of such integrations have been investigated in prototypical case studies.

The researchers of the CALCULEMUS Network also fostered the Mathematical Knowledge Management (MKM, EU MKMNET IST-2001-37057) research initiative; see [73, 19]. This relatively young line of research adopts a broader perspective on the future of mathematics (research and publication practice, education, and knowledge maintenance) in the 21st century. A significant amount of CALCULEMUS research is MKM relevant and is currently being taken up in this community in order to adopt and integrate it into the broader MKM perspective.

The extensive research activities of the CALCULEMUS Network are furthermore shown inter alia by three special issues of the Journal of Symbolic Computation [226, 9, 176] and the following international events: CALCULEMUS Symposium 2000 in St. Andrews, Scotland [150, 226], CALCULEMUS Symposium 2001 in Siena, Italy [176], CALCULEMUS Symposium 2002 in Marseilles, France [87, 99], CALCULEMUS Autumn School 2002 in Pisa, Italy [38, 39, 40, 277]

In the following paragraphs we sketch the highlights of our research in the different work tasks; for more detailed reports to all tasks we refer to [37].

Task 1.1: Mathematical Frameworks TUE and Nijmegen University investigated type theory for the purpose of formalising mathematics: Barendregt and Geuvers [34] give an overview of type theory, how it is used to represent logic and mathematics and what issues and choices come up. Type theory (encoded in OPENMATH) as a way for communicating mathematics is proposed in [33] and in [96] it is shown how a proof presentation can be generated from a formalised proof in type theory. This paper argues that 'formal contexts' in Coq can be used as a basis for interactive mathematical documents. This topic is also treated in [214]. An in-depth discussion of the various ways to treat computations in theorem provers is given in [32] and further related work is presented in [54].

The Network has also studied other approaches to theorem proving and their capacities to integrate computations (see also [265]). This includes proof planning, as developed and employed by the nodes USAAR and UED. In the ΩMEGA system [238], at USAAR, symbolic calculations can be integrated into proof planning in two ways: (i) to guide the proof planner and to prune the search space by computing hints with control rules and (ii) to shorten and simplify the proofs by calling a CAS within the application of a method to solve equations. As a side-effect both cases can restrict possible instantiations of meta-variables. These approaches are discussed in [105, 243, 194, 239].

An investigation into the use of deduction for the implementation of correct computations within computer

[1]The Network is therefore also striving towards the definition of a uniform theoretical framework for DSs; see, for instance, [24] for some preliminary work.

algebra system was considered at UGE and is presented in [3].

The THEOREMA system, developed at RISC, aims at providing one mathematical framework encompassing all aspects of algorithmic mathematics, notably the aspects of *proving*, *computing*, and *solving*; see [72, 67, 68].

In [153, 154] it is critically argued by UBIR that aspects of mathematical concepts, including procedural knowledge, are hard to reconstruct from the formalisation in deduction systems. This work points to limitations of the flexibility of mathematical representations which apply to all our current approaches.

Task 1.2: Definition of Mathematical Service The primary goal of this Task is the enhancement of existing computer algebra systems and deductive systems by turning them into *open* systems capable of using and/or providing mathematical services. After a preliminary analysis of the state-of-the-art of reasoning systems, it was decided to tackle the problem, in parallel, by a top-down and a bottom-up approach.

In the top-down approach, new infrastructures (both at the conceptual, specification, and architectural level) for the seamless integration of mathematical services have been investigated. This was intended not only for current systems, but also and in particular for future implementations. To this extent particular emphasis was on the definition of frameworks (languages, protocols, semantic specifications, architectural schemata) suitable for making mathematical services accessible over the web. The relevant top-down approaches are: OMRS (Open Mechanised Reasoning Systems) developed by UGE and ITC-IRST [5], LBA (Logic Broker Architecture) developed by UGE [17, 18], MathWeb-SB (MathWeb Software Bus) developed by USAAR [278], MathBroker developed by RISC [186]. These networks can themselves be coupled again as, for instance, exemplarily investigated in [276].

In the bottom-up approach, we have investigated how complex mathematical services can be built out of simpler ones. A particular emphasis has been devoted to decision procedures, and in particular to the integration of procedures specific for solving mathematical problems with deductive procedures. Examples for bottom up approaches are CCR (Constraint Contextual Rewriting) developed by UGE and MathSat [129, 22, 21, 20, 23], developed by ITC-IRST.

In Task 1.2 the CALCULEMUS network also closely cooperates with the EU project MONET (project number IST-2001-34145) and a joint workshop[2] has been organised by O. Caprotti in November 2002 at RISC. In MONET special ontologies comprising mathematical problems, queries and services have been defined and investigated.

Task 2.1: Integration of CASs and DSs via Protocols Cooperation among several software systems can be achieved with indirect, unidirectional and bidirectional communication. The goal of this task is to investigate how protocols can be defined to provide a semantics as well as soundness results for systems exchanging mathematical information. This definition hints at several other tasks in the Network dealing with very similar problems. This is for example true when defining a context for a computation and is partly covered in Task 1. Unidirectional and bidirectional communication protocols are designed when coupling directly different modules. Although there are no direct links between the services with indirect communication, interaction is possible when systems can communicate with a common user interface, central unit, mediator or evaluator. This approach, which is partly based on a joint work with ITC-IRST on OMSCS (Open Mechanised Symbolic Computation Systems), has been investigated within the KOMET system at UKA see [86, 164, 119, 88].

A semantics can be provided by at least three approaches: (a) define a mathematical software bus, (b) define a context from which a semantic can be derived, (c) formulate the problem as a knowledge representation paradigm.

These approaches are shared by several of the partners. Indeed, they lead to introduce multi-agent systems, contexts, and ontologies to just quote a few features (see for instance the LBA and the MathWeb-SB).

[2]See `www.esblurock.com/~ocaprott/mathbrokerWS.html`.

Task 2.2: Enhancing the Reasoning Power of Computer Algebra Systems Enhancement of CAS with reasoning power can be attempted at different levels: (a) enhancement of CAS on the System Level, (b) enhancement of CAS on the Theory Level, and (c) enhancement of CAS on the User Level.

Direction (a) can be achieved by adding additional reasoning capabilities, i.e., logical inference systems, to algorithms built into the CAS. The Constraint Contextual Rewriting (CCR) framework developed by UGE can be used in order to integrate the evaluation mechanism of the CAS MAPLE with an appropriate decision procedure for checking side-conditions, see [3] and [14].

Direction (b) can be achieved by adding proven knowledge about CAS functions to the CAS knowledge base. The HR system, developed at UED, has been used to conjecture properties of functions available in the MAPLE algorithm library from empirical patterns detected in computational data produced by the CAS [108].

Direction (c) can be achieved by giving the CAS user the possibility to prove mathematical statements using proof techniques from logic within the CAS in addition to the computing facilities that each CAS offers. In the framework of the CALCULEMUS Network, the work of RISC represents this aspect of CAS enhancement: The THEOREMA system, see [75], is an add-on package for the widespread and popular CAS *Mathematica* where the user formulates mathematical theorems and proves them entirely within the *Mathematica* environment.

Task 2.3: Enhancing the Computation Power of Deductions Systems UED investigated the combination of the proof-planner $\lambda Clam$ [228] with other systems for computationally costly tasks. This includes (a) an implementation of the GS flexible decision procedure system framework in (Teyjus) LambdaProlog and within the $\lambda Clam$ proof planning system [83] and (b) the integration of the $\lambda Clam$ proof-planner into the MathWeb-SB system [114].

UED also investigated the combination of systems to discover attacks to security protocols [244, 245]. This work makes use of computational power in that it generates a large number of clauses in its processing.

Further relevant work has been done in the $\lambda Clam$ proof-planner to construct very large and modular proof-plans for complicated real analysis theorems [131, 179, 180].

The ΩMEGA proof planner at USAAR has been coupled with different CASs via MathWeb-SB, see [243, 194, 239]. The ΩANTS approach to integrate CASs into mathematical assistant systems is sketched in [44, 43, 49, 50]. This work proposes an agent-based modelling of inference rules and external systems at a very basic level within theorem provers.

Finally, work done at UBIR and UGE which render techniques from automated reasoning highly efficient by using enhanced computational power are presented in [139, 140, 141] and [20, 23, 6]. Further relevant work is given in [225].

Task 3.1: Automated Support to Writing Mathematical Publications Typically, a mathematical publication contains the following ingredients: natural language text, mathematical formulae, formal text (i.e. definitions and theorems), proofs, examples (typically with computations), and graphics (tables, drawings, sketches, etc.). In the optimal case, a software system for supporting mathematical publications would support all these facets of mathematical publications. Several systems and languages have been used for case studies in this area:

(a) The MIZAR approach (at UWB) is based on two kinds of software which automate the process of writing formal mathematical papers: (i) software used to prepare an article as a formal text whose correctness is computer verified and (ii) the software for automatic (or semi-automatic) translation into natural language (particularly English); this includes also the software for translation into XML-based formats. The cooperation with other CALCULEMUS sites includes development of the MIZAR Mathematical Library (MML) and also the above mentioned translation into XML formats. Relevant publications are [199, 126, 29, 30, 31].

(b) THEOREMA is a prototypical software system designed to give computer-support to the working mathematician during all phases of mathematical activity. Several features qualify THEOREMA as a powerful system for creating mathematical publications entirely inside the system. “Classical” mathematical documents can be written that are intended mainly for printout, as for instance the thesis [268] or the conference papers [266], [267], and [269]. In the case studies, however, emphasis has been put on using the THEOREMA system for developing interactive lecture notes for university mathematics courses. Mostly since the THEOREMA language is very similar to the language used in “ordinary mathematics” the system is highly suitable for this approach, both in illustrating computation-based courses as well as in supporting proof-oriented courses.

(c) The OMDOC [157] content markup scheme which has been developed at USAAR, supports authors with writing formal mathematical documents including articles, textbooks, interactive books and courses. OMDOC allows to capture the semantics and structure of these documents. Various tools are available to transform OMDOC documents into other formats for presentation purposes (using, e.g., MathML) or to support inter-system communication (e.g., by transformation into the logic of a theorem prover).

(d) TUE has developed the MATHDOX tool supporting interactive mathematical documents. MATHDOX is based on DOCBOOK but also has similarities to OMDOC.

Task 3.2: Support to the Development of an Industrial-Strength Application of Formal Methods to Program Verification In addition to formal methods, which is undoubtedly the most important application area for our research, we have identified the education sector as another interesting application for DSs and CASs. Actually the systems THEOREMA (RISC) and ACTIVEMATH [197] (USAAR), which make use of tools and approaches developed in the CALCULEMUS Network, are already employed in education practice. Another example is the MATHDOX tool developed at TUE since the next version of the interactive textbook *Algebra Interactive!* [104] will appear in this format.

Formal method applications currently pursued in the Network include (a) an approach to support the verification of hybrid systems with the help of mathematical services in MathWeb-SB [42, 41] — cooperation of UGE, USAAR, UED, (b) the investigation whether specialised reasoning tools within the MathWeb-SB can fruitfully support the formal verification of information flow properties and error detection in security protocols [23] — cooperation of UGE, USAAR, UED, ITC-IRST, and (c) the application of proof planning in first-order linear temporal logic (FOLTL) to feature interactions as they arise in large telephone networks [100] — at UED.

Task 3.3: Support to the Solution of Undergraduate Exam in Calculus and Economics In this Task we focus on simple, mathematics education oriented problems with a strong emphasis on the particular way the problems are solved, how interaction with the user is supported and how the solution is presented. We analyse whether our systems can be employed in a user friendly and adequate way and whether the interaction and maths presentation capabilities of the systems are appropriate.

A task relevant case pursued at Nijmegen University compares how the problem of proving the irrationality of $\sqrt{2}$, which involves computations, can be proved in fifteen different theorem proving environments (including systems of the CALCULEMUS Network) [265, 263, 240, 48, 239].

Among the case studies that are currently being started at USAAR are exercises from the German *Bundeswettbewerb Mathematik* and Calculus exercises being encoded and investigated in the ACTIVEMATH project. Empirical studies at USAAR investigates the phenomena of natural language dialog with mathematical assistant systems on proof exercises in naive set theory.

Task 3.4: Modelling of Existing Systems as Mathematical Services The work in this Task so far has concentrated both on developing the required infrastructure (languages, protocols, semantic specifications, architectural schemata) for making existing systems inter-operate, and on studying extensions and enhancements of the reasoning capabilities of some existing tools. The relevant contributions are: (i) MathSat framework developed at ITC-IRST [22, 21], (ii) the RDL (Rewrite and Decision procedure Laboratory), (iii) the LBA [17, 18, 276] developed by UGE, (iv) the modelling of existing systems, for instance, $\lambda Clam$ developed at UED [228], as mathematical services in MathWeb-SB developed at USAAR [114].

Further work at USAAR concentrates on the mediation of mathematical knowledge between the mathematical knowledge base MBASE, which has been integrated to the MathWeb-SB, and mathematical assistant systems such as ΩMEGA [122, 48, 47].

Task 3.5: Challenge Mathematical Problems During the work on the above tasks some challenging mathematical problems had to be tackled already, in order to have non-trivial working examples. Some of the examples were done either by single partner nodes or in collaboration between some of the nodes. The examples include: (i) Fundamental Theorem of Algebra [125, 124], (ii) Involutive Bases [89, 85], (iii) Exploration in Finite Algebra, (iv) The Residue Class Domain [192, 195, 193, 194], (v) Proving with Invariants [196], (vi) The Jordan curve theorem for special polygons, (vii) Continuous lattices [163], (viii) Order sorted algebras [257, 253, 256], (ix) Proofs in Homological Algebra, (x) Proofs in Graph Theory, (xi) Exploration in Zariski Spaces. Further related work is given in [45, 46].

Task 1.1: Mathematical Frameworks

TASK LEADER: TUE
SCIENTISTS IN CHARGE: ARJEH COHEN, HENK BARENDREGT, HERMAN GEUVERS, FREEK WIEDEJK
RESEARCH TEAM: USAAR, UKA, RISC, TUE, UBIR

1.1.a Overview

Theorem provers are notably good at reasoning and less appropriate for computation. The reason is that to preserve the soundness of the logic of the theorem prover, one can only allow computations that are "correct": simplifying $\sqrt{x^2}$ to x in a real number expression may speed up computation, but combined with the reasoning facilities of a theorem prover, it also allows to derive $1 = -1$, which is obviously undesirable. In a computer algebra system, the user is left responsible for checking the side conditions under which the output is valid, but for a theorem prover system, this is not good enough: the whole point of theorem provers is that they prevent the deriving of invalid statements. So it seems that (fast) computation and (correct) reasoning are antipodes and to a certain extent that is true: if one holds no responsibility for correctness, it will in general be easier to write fast algorithms that suffice for most cases. On the other hand, if one disallows fancy computations and restricts oneself to simple (user-guided) equational reasoning, it is easier to preserve correctness. But of course there is room for improvement on both sides of this spectrum. Computer algebra systems can be made "aware" of side conditions under which certain algorithms are correct and theorem provers may be enhanced by (user-defined, proven correct) computation facilities.

In Section 1.1.b type theory is introduced as a formalism which is expressive enough to include computations via inductive data types and reflection within the formalism itself. The computations provided by external systems can be used to justify proof steps and for the introduction of witness terms in proof planning; this is described in Section 1.1.c. The use of deduction for the implementation of correct computations within computer algebra systems and the THEOREMA system, developed at RISC are presented in Section 1.1.d.

In Section 1.1.e it is argued that aspects of mathematical concepts, including procedural knowledge, are hard to reconstruct from the formalization in deduction systems.

1.1.b Computation using Type Theory

The Type Theory of Coq

We have made several investigations into the use of type theory as a basis for formalizing mathematics. This work has been carried out at Nijmegen University (NL) which is a sub-site of EUT. These investigations have been of a theoretical nature, but we have also done practical experiments by doing large formalizations in the type theoretic theorem prover Coq (this is part of Task 3.5) and by improving the computation capacities of type theoretic theorem provers like Coq (this is part of Task 2.3). In this Section we focus on the theoretical investigations we have made into the application of type theory for formalizing mathematics. An important focus is on the way type theory deals with computations, which will be discussed in more detail in the Section 1.1.b. Here, we first explain the basic relevant ideas of type theory.

In type theory one interprets formulas and proofs via the well-known 'formulas-as-types' and 'proofs-as-terms' embedding, originally due to Curry, Howard and De Bruijn. Under this interpretation, a formula is viewed as the type of its proofs. Hence, a statement in type theory of the form

$$M : A$$

can be read in the following two ways:

- M is an *element of the set* denoted by A,
- M is a *proof of the formula* denoted by A.

In the case that M denotes a proof, it is actually a term notation for a natural deduction style derivation. The main consequences of this approach towards theorem proving are that

- Proof checking is Type checking,
- Interactive Theorem Proving is the interactive construction of a term of a given type.

The Proof Assistant Coq is an interactive theorem prover based on type theory: the implemented typed λ-calculus is the *Calculus of Inductive Constructions*, CIC: a version of constructive higher order logic with powerful inductive types. The system Coq provides the user with powerful tactics to interactively construct a proof term. In this construction process, the system guarantees the type correctness. An important distinction to be made – which is a basic philosophy behind type theoretic provers like Coq – is the one between

- Checking an alleged proof: this is easy, comparable with checking the *syntactic correctness* of a computer program,
- Constructing a proof for a given formula: this is hard (undecidable in general), comparable with constructing a program that satisfies a given specification.

In type theoretic provers, the first task is performed by a *type checking algorithm*, the second task is performed interactively with the user.

An important issue in (automated) theorem proving in general is the question of *correctness* of the implemented system. Or, phrased differently: how can we be sure that a formula that has been proved (interactively) by the Proof Assistant (PA) is really true? We may sometimes not be convinced that all the powerful tactics that a PA provides are sound and it occasionally turns out that a PA contains a bug. In type theoretic PAs, this issue of reliability is solved to some extent, because the PA also provides a *proof term* that can be *type checked* by the user, using his own – relatively easy to write – type checking algorithm. The feature of having proof terms that can be checked *independently* by a relatively *small* and *easy* algorithm, is known as the *De Bruijn criterion*, named after the founding father of the Automath project. In this project the first PAs based on type theory were implemented (in fact they were proof checkers instead of proof assistants).

Another important feature of type theoretic theorem provers is the so-called *Poincaré's principle*, which states that propositions which can be verified by a computation are easy; i.e., no proof is required. This principle is incorporated in CIC through the so-called conversion rule: types (and propositions) that are computationally equal (convertible) are not distinguished. This means that if we have an algorithm (inside type theory) that computes a function, say plus that computes addition, then $\mathrm{plus}(1, 1) = 2$ does not require a proof, because this is just *computationally equal* to $2 = 2$, which holds by reflection. We come back to this below.

The actual formalization of mathematics in type theory proceeds by building up a *context* of results. Such a context consists of the following items. (x is a variable and a and A are general expressions)

- $x : A$ with a data type A. This denotes a *declaration* of the variable x to be of type A.
- $x : A$ with a propositional type A. This denotes the *assumption* ('named' x) of A.
- $x := a : A$ with a data type A. This introduces a *definition* of x as a, where a is of type A.
- $x := a : A$ with a propositional type A. This introduces a reference 'named' x to the *lemma* A. Here a is a *proof* of A.

The first two are called variable declarations and the second two are called definitions. Note that a reference to a lemma is made by introducing an abbreviation (definition) of the proof term. These declarations and definitions can also be made 'locally' (e.g., under the scope of other binders).

In the reporting period we have made several investigations into type theory for the purpose of formalizing mathematics. These have been laid down in the following publications. [34] gives an overview of type theory, how it is used to represent logic and mathematics and what issues and choices come up. [33] proposes type theory (encoded in OPENMATH) as a way for communicating mathematics. [96] shows how from a formalized proof in type theory a proof presentation can be generated. It argues that 'formal contexts' in Coq can be used as a basis for interactive mathematical documents. This topic is also treated in [214].

Apart from trying to improve the type theoretic approach, we have also studied other approaches to theorem proving. This was done by the "Fifteen provers of the world" project of Freek Wiedijk, who compares fifteen theorem provers by studying how they (formalize and) prove the irrationality of $\sqrt{2}$. See [265] for a preliminary comparison. It turns out that "proof style" is an important distinguishing feature in theorem provers. A theorem prover like Coq has a "procedural" proof style: the user types in "tactics" that guide the proof engine in constructing the proof. A theorem prover like MIZAR has a "declarative" proof style: the user types in the reasoning pretty much in the style of an ordinary mathematical paper and the system gives a warning if it can't fill the gaps. The second is closer to ordinary mathematics. In [262] the two declarative proof styles of MIZAR and Isar are compared. In [264] it is shown how a declarative proof style can be programmed on top of the procedural proof style system Hol-light.

Computation in Coq

Type theory presents a powerful formal system that not only captures the notion of *proof* (via the so called 'propositions-as-types embedding', where types are viewed as propositions and terms as proofs), but also the notion of *computation*, via the inclusion of functional programs written in typed λ-calculus. There are three notions of computation: β-, ι- and δ-reduction. The first is the well-known β-rule from λ-calculus, $(\lambda x : A.M)N \to_\beta M[N/x]$. The ι-reduction captures primitive and higher order primitive recursion, which arise from the inductive types that are definable in CIC (e.g., natural numbers, lists, trees but also much more expressive types). The δ-reduction deals with unfolding of definitions: if $x := a : A$ then $M(x) \to_\delta M(a)$.

We have already mentioned two important features of type theory (CIC):

- The decidability of type checking ($\supset$ proof checking)
- The Poincaré principle: computations do not require a proof.

These imply that not all computations can be formalized in CIC: If f is a function (algorithm) on the natural numbers, then 'reflexivity' is a proof of $f(0) = 0$ if and only if the computation of f on 0 yields 0. In CIC, all computations terminate and are deterministic, due to the fact that for defining functions there is a fixed scheme that uses syntactic restrictions to prevent non-termination. The 'fixed scheme' for functions forces a user to define all functions by structural recursion, which is often felt as a limitation. Several proposals have been made to alleviate the restrictions posed by the structural recursion (without giving up the decidability of type checking). The 'Bove-Capretta' approach of [54], jointly developed in Nijmegen (Capretta) and Gothenburg (Bove) has been very successful as it succeeds in taking apart the definition of a function, which is done very much in a functional programming style, and the proof that it terminates, which can be postponed until later. It also provides a way of dealing with partial functions.

In computer algebra systems, computation is used to solve problems, not to write down 'executable functions'. In type theory, notably in CIC, we can also use the computing power of the system itself to solve problems. This is done most successfully using the so called *reflection* approach.

Reflection is the method of 'reflecting' part of the meta language in the object language. Then meta theoretic results can be used to prove results from the object language. Reflection is also called *internalization* or the *two level approach*: the *meta language level* is *internalized* in the object language. It should be stressed that reflection does not extend the logic of the theorem prover, so there is no possible consistency problem. It just enhances the reasoning by providing new tactics. The computations that are carried out by these tactics are mainly 'autarkic', i.e., they are carried *within* the system itself. [32] contains an in-depth discussion of the various ways one can treat computations in theorem provers.

The reflection method can be applied quite generally in situations where one has a specific class of problems with a decision function. It is not restricted to the theorem prover Coq. If the theorem prover allows (A) user defined (inductive) data types, (B) writing executable functions over these data types and (C) user defined tactics in the meta language, then the reflection method can be applied. The classes of problems that it can be applied to are those where (1) there is a syntactic encoding of the class of problems as a data type, say via the type $\mathsf{Problem}$, with (2) a decoding function $[-] : \mathsf{Problem} \to \mathsf{Prop}$ (where Prop is the collection of propositions in the language of our theorem prover), (3) there is a decision function $\mathsf{Dec} : \mathsf{Problem} \to \{0, 1\}$ such that (4) one can prove $\forall p : \mathsf{Problem}((\mathsf{Dec}(p) = 1) \to [p])$. Now, if the goal is to verify whether a problem P from the class of problems holds, one has to find a $p : \mathsf{Problem}$ such that $[p] = P$. Then $\mathsf{Dec}(p)$ (together with the proof of (4)) yields either a proof of P (if $\mathsf{Dec}(p) = 1$) or it 'fails' (if $\mathsf{Dec}(p) = 0$ we obtain no information about P). Note that if Dec is complete, i.e., if $\forall p : \mathsf{Problem}((\mathsf{Dec}(p) = 1) \leftrightarrow [p])$, then $\mathsf{Dec}(p) = 0$ yields a proof of $\neg P$. The construction of p (the syntactic encoding) from P (the original problem) can be done in the implementation language of the theorem prover. Therefore it is convenient that the user has access to this implementation language; this is condition (C) above. If the user has no access to the meta language, the reflection method still works, but the user has to construct the encoding p himself, which is very cumbersome.

The reflection method turns out very useful in practice. We list the use we have made of it, also in large formalizations.

- In the proof of the Fundamental Theorem of Algebra (FTA), form alized in Coq, we have implemented and used a tactic called "Rational", which solves equations between rational expressions, like $\frac{x}{y} * y = \frac{1}{\frac{1}{x}}$. To implement it, we have defined a syntactic type of rational expressions and an interpretation function to any field. An extra complication here is that the interpretation function is partial (the syntactic expression $\frac{1}{0}$ does not have a value).
- Based on the FTA work, Luis Cruz-Filipe has proved the Fundamental Theorem of Calculus FTC. In the formalization, a tactic for computing derivatives and a tactic for checking continuity have been implemented, both using the reflection method. See [111].
- We are working on a tactic that proves statements from primitive recursive arithmetic by replacing them with a computation (of the associated primitive recursive function), using the reflection method.

1.1.c Symbolic Computations in Proof Planning

In the context of this work package we developed a method that enables the use of symbolic computations in deduction, more precisely in proof planning, without sacrificing the correctness of the overall proofs that are constructed. The work was carried out by researchers at nodes in Saarbrücken and Birmingham using the ΩMEGA system [238, 239].

We will first give a brief introduction to proof planning and its particularities in the ΩMEGA system, and then explain the two different methods we have developed to integrate computer algebra.

Multi-Strategy Proof Planning

Proof planning, developed in Edinburgh by Alan Bundy [81], considers mathematical theorems as planning problems where an *initial partial plan* is composed of the proof *assumptions* and the theorem as *open goal*. A proof plan is then constructed with the help of abstract planning steps, called *methods*, that are essentially partial specifications of tactics known from tactical theorem proving. In order to ensure correctness, proof plans have to be executed to generate a sound calculus level proof.

In the ΩMEGA system [238], the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see [198] for details). That is, methods can encode general proving steps as well as knowledge particular to a mathematical domain. Moreover, *control rules* provide the possibility to introduce mathematical knowledge on how to proceed in the proof planning process by specifying how to traverse the search space. Depending on the mathematical domain or proof situation they can influence the planners behavior at choice points (e.g., which goal to tackle next or which method to apply next).

Symbolic calculations can be integrated into proof planning in two ways: (1) To guide the proof planner and to prune the search space by computing hints with control rules. (2) To shorten and simplify the proofs by calling a CAS within the application of a method to solve equations. As side-effect both cases can restrict possible instantiations of meta-variables[3].

Employing Computer Algebra in Control Rules

Computations of a CAS can be employed in control rules to influence the course of the planning process by prefering applicable methods or to compute a proper substitution for a needed witness term. In the latter case a control rule is triggered after the decomposition of an existentially quantified goal which results in the introduction of a meta-variable as substitute for the actual witness term. After an existential quantifier is eliminated, the control rule computes a hint with respect to the remaining goal that is used as a restriction for the introduced meta-variable. If hints can be computed, the meta-variables are instantiated before the proof planning proceeds. However, the instantiations suggested by `select-instance` are treated as a hint by the proof planner; that is, they have to be verified during the subsequent proof planning process. In case the proving attempt fails for a particular instantiation, the proof planner backtracks and tries to find an appropriate instantiation by crude search.

Examples of its use are given in the case study for proof planning in the residue class domain [194], which is described in more detail in the report for Task 3.5. There, for instance, it is necessary to show the existence of a unit element e in a given algebraic structure $(S, \circ)$. The control rule supplies a hint as to what e might be. To obtain suitable hints, the control rule sends corresponding queries to the CAS GAP and MAPLE.

[3]Meta-variables are place-holders for terms whose actual form is computed at a later stage in the proof search.

Employing Computer Algebra in Methods

The way we use CAS computation in methods is an extension of previous work, in particular [152], that presents the integration of computer algebra into proof planning, and [243], that exemplifies how the correctness of certain limited computations of a large-scale CAS can be guaranteed within the proof planning framework. It is based on the idea to separate computation and verification and can thereby exploit the fact that many elaborate symbolic computations are trivially checked. In proof planning the separation is realized by using a powerful computer algebra system during the planning process to do non-trivial symbolic computations. Results of these computations are checked during the refinement of a proof plan to a calculus level proof using a small, self-tailored system that gives us protocol information on its calculation. This protocol can be easily expanded into a checkable low-level calculus proof ensuring the correctness of the computation.

An example of the use of calculations, is realized within the `Solve-Equation` method in the residue class case study. Its purpose is to justify an equational goal using MAPLE and, if necessary, to instantiate meta-variables. In detail, it works as follows: If an open goal is an equation, MAPLE's function `solve` is applied to check whether the equality actually holds. Any meta-variables contained in the equation are considered as the variables the equation is to be solved for and they are supplied as additional arguments for `solve`. In case the equation involves modulo functions with the same factor on both sides, MAPLE's function `msolve` is used instead of `solve`. If MAPLE can solve the equation, the method is applied and possible meta-variables are instantiated accordingly. The computation is then considered correct for the rest of the proof planning process. However, once the proof plan is executed MAPLE's computation is expanded into low level logic derivations to check its correctness. This is done with the help of a small, self-tailored CAS that provides detailed information on its computations in order to construct the expansion [243].

1.1.d Extending Symbolic Computation with Deduction

Theorema

The THEOREMA system, developed at RISC under the direction of B. Buchberger, see [75, 70, 72, 67, 68], aims at providing *one mathematical framework* encompassing all aspects of algorithmic mathematics, notably the aspects of *proving*, *computing*, and *solving*. A detailed description of the THEOREMA system will be given in report for Task 2.2.

The system architecture bases on an existing Computer Algebra System (CAS), in the concrete case *Mathematica*, see [188], and extends it with proving facilities implemented in the native programming language of the CAS. In principle, any CAS offering programming facilities could be used as starting point. The main advantages of *Mathematica* lie in the elegant pattern-matching oriented programming style and in the powerful programmable user front-end. The *Mathematica* user front-end is—though highly sophisticated—in its nature command-line oriented, i.e. the user enters a command to the system and the system displays the result of evaluating the command. *Mathematica* comes as a huge library of algorithms mainly for computer algebra related areas based on exact representation of integers, rational numbers, and algebraic numbers, and on polynomial and rational function arithmetic.

The philosophy in *Theorema* is to provide a uniform mathematical environment offering essentially three commands reflecting the three central mathematical activities:

- `Prove` for proving formulae w.r.t. to some knowledge base,
- `Compute` for computing normal forms of given expression w.r.t. to some knowledge base, and
- `Solve` for finding terms satisfying certain properties w.r.t. to some knowledge base.

For the Prove command, methods from automated deduction need to be implemented from the scratch. We mainly emphasize on automated proving methods that generate proofs in a human readable and—more importantly—human understandable style. For a detailed description of the current proving capabilities of THEOREMA we refer to the report on Task 2.2. The Compute command mainly bases on the underlying rewrite engine provided by the *Mathematica* system. It can also be seen as one main interface, through which the powerful collection of computational algorithms in the *Mathematica* kernel can be made available in THEOREMA. The Solve command is not yet far investigated. Some algorithms for solving special types of equations are available in *Mathematica*, but other methods, notably constraint solvers for various domains available among the Calculemus nodes, shall be incorporated at this stage.

Proving, computing, and solving will be organized in such a way, that not only these three commands are available at the top-level, but an *interaction* between them can seamlessly be integrated into the system. It can be observed e.g. in proving, that an alternation of phases of proving, computing, and solving is a fruitful strategy for automated proof generation. This paradigm, called "PCS", was invented in [60] for proofs in elementary analysis. It is, however, the design guideline, which many of the proving methods developed in the frame of THEOREMA follow, notably the set theory prover described in [268]. Analogously, computing will employ phases of proving and solving and solving will integrate phases of proving and computing. It is a challenging task for the system design to setup the components so that all these interactions are easily possible on a correct and sound logical basis.

Constraint Contextual Rewriting in MAPLE

UGE and UKA have jointly worked at the reconstruction of MAPLE's symbolic evaluator and its assume facility, introduced [261] to solve inconsistencies arising from rules like $\sqrt{x^2} = x$. This rule is wrong unless x denotes a real number and $x \geq 0$. Removing the rule makes the simplifier correct, but also less powerful. The assume facility provides a way out of the dilemma: it maintains a context which enables the user to specify properties of terms, and the rule is applied to an expression $\sqrt{a^2}$ only if $a \geq 0$ can be derived from the context. Thus, due to the addition of the assume facility, MAPLE's symbolic evaluator is a complex mathematical service resulting from the combination of specialized reasoning modules: the evaluator, the property reasoner, a solver for linear programming problems, and a general solver.

The notion of context plays also a key role in Constraint Contextual Rewriting (CCR, for short) [14] (cf. Section 1.2.f). CCR is a powerful form of conditional rewriting which incorporates the services provided by a decision procedure. In CCR contextual information is stored and manipulated by the decision procedure whose interface functionalities are neatly specified in an abstract way.

The generality of the integration schema employed in CCR promotes its reuse. Indeed, we have shown that MAPLE's evaluation process can be recast in CCR as a set of cooperating reasoning specialists with neatly specified interfaces. This is not just an academic exercise:

- A fault that causes MAPLE to return wrong results with some contexts was discovered during the analysis of the assume facility. The reason for this is that the facility is based on the assumptions that one of its modules, namely the solver, is complete in the sense that it uses all the available assumptions in the context. This is not the case.

- CCR provides a solution to this problem, at least if the context contains only linear equalities and inequalities, by integrating linear arithmetic more tightly with simplification than the present implementation in the assume facility does.

- Only a certain class of lemmas about functions is amenable to the assume facility. It has been shown that augmentation can be used to extend this class.

This leads also to the observation that properties of user-defined functions should be declared rather than programmed. Last but not least, Weibel and Gonnet's property reasoner has been made available to the automated reasoning community.

The results of this work have been published in [3].

1.1.e Design of Mathematical Concepts

It is one of the deep mathematical insights that foundational systems like first-order logic or set theory can be used to construct large parts of existing mathematics and formal reasoning. Unfortunately, this insight has been used in the field of automated theorem proving as an argument to disregard the need for a diverse variety of representations. While design issues play a major role in the formation of mathematical concepts, the theorem proving community has largely neglected them. We argue that this leads not only to problems in human computer interaction, but that it causes severe problems at the core of reasoning systems, namely at their representation and reasoning capabilities. In order to improve applicability, theorem proving systems need to take care about the representations used by mathematicians.

Donald Norman gives a fascinating introduction into “The Design of Everyday Things.” His insights are of a very general nature and we argue that principles for good design hold in mathematics as well. The design of concepts in mathematics takes a lot of the burden on getting things right from the human user by the use of appropriate representations. The different representations are used to keep information together, hide unimportant details, and allow to concentrate on the important parts. Sometimes the right representation is the key step in the process of problem solving. If one were to use a foundational system directly, however, everything would have to be expressed explicitly in a uniform representation, which offers no or only little structural support.

To exemplify this, we will take a closer look at multiplication tables.

$$\begin{array}{c|ccc}
\circ & d_1 & \cdots & d_n \\
\hline
d_1 & c_{11} & \cdots & c_{1n} \\
\vdots & \vdots & \ddots & \vdots \\
d_n & c_{n1} & \cdots & c_{nn}
\end{array}$$

The information accessible from the table is that it is a binary operation, it is discrete and defined on a finite domain. Domain and range are directly given. The table has its own notion of well-formedness, that is, all d_i have to occur and have to be different, the table must be fully filled. In the design we find natural and cultural constraints. Multiplication tables are designed in a way that their structure puts “information in the world” that makes it difficult to violate well-formedness. An under-specification would leave a hole in the structure, it is impossible to enter more than one entry per field. Furthermore, although the order of the d_i in the columns and rows could in principle be different, cultural conventions prevent that. This in turn makes particular reasoning methods possible which are connected to the representation. For instance, the commutativity of $\circ$ is checked by verifying that the table is symmetric with respect to the diagonal.

This work has been carried out in collaboration of the nodes in Birmingham and Saarbrücken. It particularly involved the YVR Martin Pollet. The results where published in [153, 154].

Task 1.2: Definition of Mathematical Service

TASK LEADER: ITC-IRST
SCIENTISTS IN CHARGE: FAUSTO GIUNCHIGLIA, ROBERTO SEBASTIANI, MARCO BOZZANO, ALESSANDRO CIMATTI
RESEARCH TEAM: USAAR, UED, UKA, RISC, ITC-IRST, UGE

1.2.a Overview

Broadly speaking, a *mathematical service* is a set of implementations running on a particular machine which accomplishes some mathematical task. An implementation is a particular realization of an algorithm as executable software, possibly with additional constraints on the input and additional possibilities for the output. Mathematical services are traditionally subdivided into those providing *proving*, *solving* or *computing* capabilities [58]. E.g., Deduction Systems (DSs) mostly provide proving services, whilst Computer Algebra Systems (CASs) mostly provide solving and/or computing services. [4]

The primary goal of this task is the enhancement of existing computer algebra systems and deductive systems, with the aim of turning them into *open* systems capable of using and/or providing mathematical services. Since we aim at providing a definition encompassing the complexity of state-of-the-art systems (with issues ranging over from the very theoretical ones at the mathematical level to the very technological ones at the communication level) we found it convenient to pursue a variety of different approaches (classified in bottom-up and top-down, as stated below).

In particular, the support of communication and interaction between these categories of mathematical services is required for tackling real problems in mathematics. Thus, a key goal of this task is to find common frameworks for defining mathematical services and description formats that abstract from the particularities of an implementation and focus on the actual problem being solved. To this aim, it is also important to identify the architectural and functional requirements (e.g, communication protocols for networked mathematical services) for turning existing systems, like CASs and DSs, into *open* systems capable of using and/or delivering mathematical services.

We summarize below the contributions relevant for task 1.2. The contributions are detailed in the remainder of this report. After a preliminary analysis of the state-of-the-art reasoning systems, it was decided to tackle the problem in parallel by a top-down and a bottom-up approach. In the top-down approach, new infrastructures (both at the conceptual, specification, and architectural level) for the seamless integration of mathematical services have been investigated. This was intended not only for current systems, but also and in particular for future implementations. To this extent, a particular emphasis has been devoted to the

[4]Notice that the meaning of the terms "proving", "solving" and "computing" is not universally stated and may be different among the different scientific communities. E.g., in the SAT community SAT tools are commonly called "solvers", whilst in our sense they provide proving services.

definition of frameworks (languages, protocols, semantic specifications, architectural schemata) suitable for making mathematical services accessible over the web.

The relevant contributions are:

- **OMRS** (Open Mechanized Reasoning Systems), developed by UGE and ITC-IRST, is a specification framework for logical services. An OMRS specification consists of three layers: the *logic* layer (specifying the assertions manipulated by the system and the elementary deductions upon them), the *control* layer (specifying the inference strategies), and the *interaction* layer (specifying the interaction of the system with the environment). Notice that this layering allows for an additional and complementary way to structure the specifications with respect to the standard approach based on modularity. As a consequence, OMRS specifications are therefore more structured than conventional specifications. This domain-specific feature of the OMRS specification framework is fundamental to cope with the complexity of functionalities provided by state-of-the-art implementations.
- **LBA** (Logic Broker Architecture), developed by UNIGE, is an architecture which provides the required infrastructure for making mechanized reasoning systems interoperate. In the LBA each mechanized reasoning system is seen as an entity providing and/or requiring a set of mathematical services. The LBA provides location transparency, a way to forward requests for logical services to appropriate reasoning systems via a simple registration/subscription mechanism, and a translation mechanism ensuring the transparent and provably sound exchange of logical services.
- **MathWeb-SB** (MathWeb Software Bus) [278] connects a wide range of reasoning systems (*mathematical services*), such as ATPs, (semi-) automated proof assistants, Computer Algebra Systems (CASs), model generators (MGs), constraint solvers (CSs), human interaction units, and automated concept formation systems, by a common *mathematical software bus*. Reasoning systems integrated in the MathWeb-SB can therefore offer new services to the pool of services, and can in turn use all services offered by other systems.
- **MathBroker**, developed by RISC, is a software framework for brokering mathematical services that are distributed among networked servers. The foundation of this framework is a language for describing the mathematical problems solved by the services. Servers register their problem solving capabilities with a ”semantic broker” to which clients submit corresponding task descriptions.

In the bottom-up approach, we have investigated how complex mathematical services can be built out of simpler ones. A particular emphasis has been devoted to decision procedures, and in particular to the integration of procedures specific for solving mathematical problems with procedures with deductive power. We provided formal modeling of the following integration schemata:

- **CCR** (Constraint Contextual Rewriting), developed by UNIGE, is a generalized form of rewriting that allows for the effective and plug & play integration of decision procedures in formula simplification. CCR is a generalization of (contextual) rewriting that incorporates the functionalities provided by a decision procedure. The services of the decision procedure are characterized *abstractly* (i.e., independently of the theory decided by the decision procedure) and the notation CCR(X) (by analogy with the CLP(X) notation) is used to stress this fact. By using CCR(X) as a *reference model*, the problem of the integration of decision procedures in formula simplification is reduced to the implementation of a decision procedure for the fragment of choice.
- **MathSat** [129, 22, 21], developed by ITC-IRST, introduces a formal framework, a generalized algorithm and architecture for integrating boolean reasoners and mathematical solvers so that they can efficiently solve boolean combinations of boolean and mathematical propositions. Many techniques are described to optimize this integration. Moreover, the MathSAT framework evidences the main requirements boolean reasoners and mathematical solvers must fulfill in order to be integrated correctly and to achieve the maximum benefits from their integration. The MathSat procedure [20, 23] is ITC-IRST implementation of an integrated procedure based on the MathSat framework.

- $\lambda Clam$ is a proof planning system which has been integrated into the MathWeb-SB framework. As a result, $\lambda Clam$ can now use reasoning services provided by existing systems in MathWeb-SB, and, in turn, provide new reasoning services to them.

In Task 1.2 the CALCULEMUS network also closely cooperates with the EU project MONET (project number IST-2001-34145) and a joint workshop[5] has been organised by O. Caprotti in November 2002 at RISC. In MONET special ontologies comprising mathematical problems, queries and services have been defined and investigated.

1.2.b The OMRS Specification Framework

The Open Mechanized Reasoning Systems (OMRS) project has the objective of designing a formal framework for the specification of state-of-the-art provers. The starting point of the OMRS approach is to structure the specification of a system in a logic component, a control component, and an interaction component, thereby suggesting the following equation:

OMRS = LOGIC + CONTROL + INTERACTION

Preliminary but significant results have been obtained in the application of the OMRS framework for supporting (i) the definition and the development of provers as open architectures usable in a ”plug-and-play” fashion, and (ii) the design and development of proof-checkable and customizable reasoning systems.

Starting from the consideration that any reasoning system, as such, performs deductions within some logic(s), guided by some (more or less complex) heuristics, and exhibits some interaction capabilities, an OMRS specification of a reasoning system is structured in a logic component, a control component, and an interaction component. The logic component provides a description of the assertions manipulated by the system and the elementary deductions upon them; the control component allows for the specification of the strategies guiding the construction of complex deductions out of the elementary ones; finally the interaction component specifies how the system interacts with the external world (including human users and other provers). Crisply separating the concerns of the three layers, results in clearer and better specifications. This is an important issue as it allows us to cope with the complexity of existing systems.

UNIGE (together with ITC-IRST) has contributed to the definition of the control layer of the OMRS specification framework [5]. This key idea of the approach is to specify the control layer by

(i) adding control knowledge to the data structures representing the logic by means of *annotations*; this leads naturally to an extended notion of inference which accounts for the simultaneous manipulation of logic and control information;

(ii) specifying proof strategies via *tactics*, i.e., expressions denoting sets of admissible derivations.

1.2.c The LBA Architecture

UNIGE has both designed the conceptual model of the LBA [17, 18] and developed two prototype implementations of the LBA: one based on CORBA and – recently – one based on XML. Moreover, a bridge between LBA and MathWeb has been defined [276].

The Logic Broker Architecture (LBA) addresses the problems arising from the integration of different reasoning systems. In particular, interconnection of two different reasoners can lead to unsound results, because of differences in the underlying semantics. The LBA architecture addresses this problem by means

[5]See `www.esblurock.com/~ocaprott/mathbrokerWS.html`.

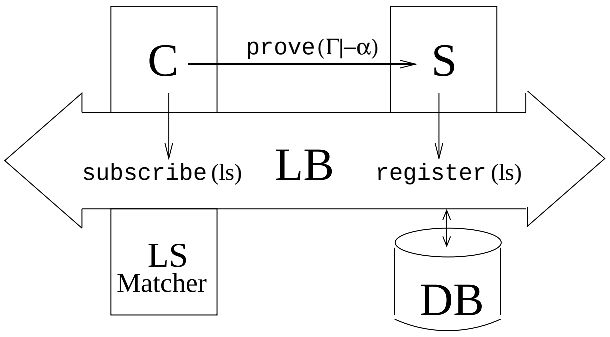


Figure 1: The Logic Broker Architecture. A client C wants to prove a formula; it subscribes its query to the Logic Broker (LB), waiting for a result. LB tries to find in the database (DB) a server matching the requested service and provide to C the service pointer.

of a diversification between the logic layer and the communication layer. A *reasoning theory* can be thought of as composed of a *sequent system* and some *inference rules*, which respectively model assertions and inference steps. Before instantiating a communication, a client sends to the Logic Broker a pair containing its set of inference rules and the service requested. The broker then tries to make a match between client query and services registered in its own database. If there is a positive result, then the Logic Broker provides the connection between the objects. The architecture can be seen from the point of view of the client like a library of functions, which can be easily integrated into the local environment, without any overhead coming from network connections. Note that the client doesn't ask about a specific server, but calls a service like `simplify` an equation, `solve` a set of constraints, etc. As a result, the same client can receive many solutions coming from different servers and it can apply some policies to decide which is the best for its computation. This level of decision can be shifted to the broker, asking for the first solution, or for the complete list of them.

One of the main goals of LBA is to use only consolidate standards, which can be easily implemented in the most common development environments. Due to this, the new version of LBA supports two main technologies, namely CORBA and XML. CORBA comes out from the tradition of LBA, guarantees the possibility to share and distribute not only results, but also parts of the proof or parts of the strategy, when this is possible. XML ensures the possibility to communicate with a large variety of web services and to interpolate data very fast. LBA also uses a standard for sharing mathematical documents, namely OPENMATH. Thanks to the extremely open design, OPENMATH can be used to describe a huge variety of mathematical knowledge. Everything is regulated by the agreement of the Content Dictionaries, which contain the classifications of mathematical symbols. Each client/server has its own phrase-book that translates the local language into the common layer OPENMATH.

1.2.d The MathWeb Software Bus

Specification of Automated Theorem Provers

In [10], Armando, Kohlhase, and Ranise presented a first taxonomy of possible states for automated reasoning systems. J. Zimmer adapted this taxonomy to the special case of Automated Theorem Provers (ATPs) and extended it by states which describe errors, timeouts and situations where the search is exhausted for some reason. First results of this work are described in an informal note [275]. The current state of our taxonomy can also be seen in Figure 2.

J. Zimmer extended all first order ATP services in the MathWeb-SB such that the `prove` service always returns one of the valid ATP states. Furthermore, all first-order ATPs in the MathWeb-SB now accept

state	For input formula F ATP has
`proof`	found a proof for F
`counter-proof`	found a proof for $\neg F$.
`valid`	determined that F is valid by some method
`unsatisfiable`	determined that F is unsatisfiable ($:\Longleftrightarrow \neg F$ valid)
`satisfiable`	determined that F is satisfiable (has a model)
`counter-sat`	determined $\neg F$ is satisfiable (has a model)
`successful`	successfully terminated but couldn't determine one of the above
`error`	detected an error (e.g., incorrect problem description)
`syntax-error`	detected a syntax error in F
`timeout`	used up a given time resource and is not yet `determined`
`incomplete`	the prover could not go on with the search (e.g. SoS empty)
`unsuccessful`	been unsuccessful for some reason
`determined`	determined one of the above (this will never be used)
`undetermined`	not determined any state

Figure 2: Valid states of MathWeb ATPs, where the formula F is logically equivalent to the given problem description (set of assumptions + conclusion).

`Name`	IntegerSort
`Context`	Sort
`Types`	
`Input`	xs: `ListOf Integer`;
`Output`	ys: `ListOf Integer`;
`InConstraints`	le(length(xs), 100);

`OutConstraints`	before(x, y, ys) $\leftarrow$ ge(x,y);
	in(x, ys) $\leftarrow$ in(x, xs);

`ConcDescriptions`	
`TextDescriptions`	sort list of at most 100 integers

Figure 3: A LARKS description of a sorting service for integers.

problem descriptions in the standard languages TPTP and OMDOC. The described work forms a crucial step towards the definition of a uniform first-order theorem proving service that is independent of concrete implementations.

Capability Description Languages for Mathematical Services

In the context of a further agentification of the MathWeb-SB, J. Zimmer investigated the possible use of existing frameworks for the description of mathematical services. MathWeb agents should offer abstract mathematical services that are described in a service description language. Service descriptions should describe the valid input and output parameters of a service, as well as the semantics, i.e., the mathematical task a service performs (e.g. proof of a theorem or factorization of a polynomial). Since the latter is closely related to the definition of web services, we studied a possible use of frameworks developed in the semantic web community, such as RDF [55], UDDI, and WSDL [103], and of languages developed in the information agents community, such as the capability description language LARKS [247] (a Language for Advertisement and Request for Knowledge Sharing).

LARKS is expressive, easy to use, and capable of supporting inferences in capability descriptions. It also incorporates application domain knowledge in agent advertisements and requests. Domain-specific knowledge is specified as local ontologies in the concept language ITL. Figure 3 shows the LARKS description of a sorting service for lists of integers. The slots of the description in Figure 3 have the following meaning:

`Name` The name of capability description for human consumption.

`Context` of keywords denoting the domain of the description.

`Types` slot allows to define type abbreviations using the existing primitive types.

`Input` (`Output`) slot contains the specification of the input (output) parameters of the service.

`InConstraints` and `OutConstraints` slots contain constraints on the input and output variables. These constraints are expressed in Horn clauses and describe the actual functionality of the service. In the example, the constraints say that the output list ys contains at least the elements of xs in a sorted order.

`ConcDescriptions` can contain optional descriptions of concepts in the ontology description language ITL.

`TextDescription` describes the capability and the meaning of the content of the other slots in natural language.

Using this and the taxonomy of states described above, we can, for instance, specify the first-order theorem proving service as it is offered by the MathWeb-SB.

`Name`	proveFOP
`Context`	ProofTheory
`Types`	
`Input`	1: `FOConjecture`; format: `String`; replyWith: (state: `ATPState`, host: `String`, time: (user: `Real`, real: `Real`, sys: `Real`));
`Output`	atp_result: (state: `ATPState`, host: `String`, time: (user: `Real`, real: `Real`, sys: `Real`));

`InConstraints`	default(format,"tptp");
	default(replyWith, record(pair(state, "")));
	default(time, 120);

`OutConstraints`	
`ConcDescriptions`	
`TextDescriptions`	Try to determine the logical status of a first order problem.

This description does not contain much explicit knowledge about the semantics of the service. This is due to the fact that most of the semantics is hidden in the definition of the `ATPState`s in section 1.2.d. This raises a crucial question that has to be answered in the context of the definition of mathematical service, namely what knowledge should be located in the definition of the ontology and what knowledge should explicitly be given in the service descriptions themselves.

1.2.e The MathBroker Web Framework

At RISC the work on Task 1.2 has mainly been devoted to the definition of a language that describes the mathematical problems solved by the services [186]. This language has to serve as a foundation for a framework for brokering mathematical services distributed among networked servers, as such it extends WSDL, the web service description language. Servers register their problem solving capabilities expressed in this language with a "semantic broker" to which clients submit corresponding problem descriptions. The broker (possibly in cooperation with a deduction system) determines the suitable services and returns them to the client for invocation. This mechanism thus hides from the client the actual implementation of mathematical services and focuses on the semantical aspects carried out. The overall design of the description language is structured in layers which proceed from an abstract view, namely the problem addressed by the mathematical service, all the way down to the hostname-port address of the service, as depicted in Figure 4.

On top of this, a run-time system accepts the descriptions of compound tasks and coordinates the invocation of the services offered by the broker. This mechanism thus hides from the client the coordination of mathematical services. Embedded into XML-documents and interpreted by browser applets, such descriptions may act as interactive hypermedia interfaces for distributed mathematical applications.

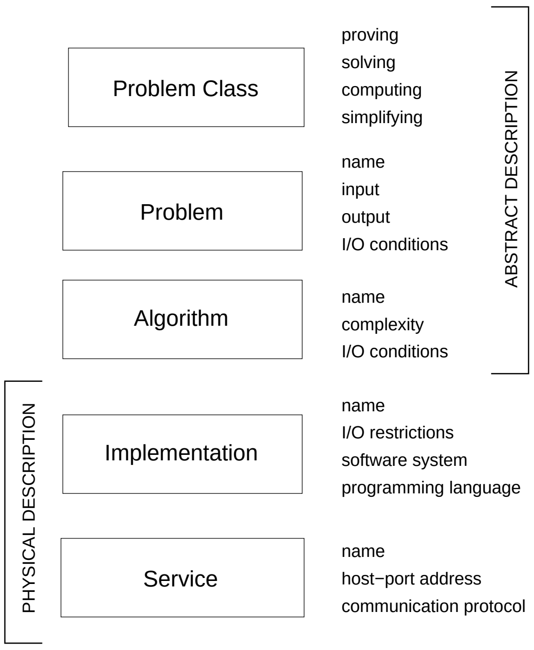


Figure 4: Mathematical service description language layers

The mathematical service description language has been presented at the International Congress of Mathematical Software, in Beijing [98] and at the MathML International Conference 2002, in Chicago [97]. Since application areas arising from the work of the Calculemus partners [87] are the prime candidates examples where we intend to apply the mathematical service description language, we organized a workshop at RISC [187], on November 11-12, 2002 on the topic of *Mathematical Web Services* to follow up on the informal meetings that took place in Marseille (July 2002) and in Pisa (September 2002).

1.2.f The CCR Rewriting Framework

The effective integration of decision procedures in formula simplification constitutes one of the key ingredients in many state-of-the-art automated reasoning systems. In many cases the interplay between the reasoning modules is so tight and complicated that providing an accurate description of the integration is a challenge. As a matter of fact, the descriptions available in literature are given by examples or in informal ways with design decision often intermixed with implementation details. As a consequence, most of the existing integration schemas are difficult to grasp, making it very difficult any attempt to modify, extend, reuse, and reason about. The goal of the CCR Project is the integration of reasoning specialists in simplifiers.

This goal can be achieved in two ways. Firstly, (in many cases) the integration of decision procedure in verification systems is performed by means of a tight cooperation between a rewriting engine and a decision procedure. Making this cooperation effective is a daunting task and it requires sophisticated techniques. The difficulties lie in both formalizing and proving the desired properties (e.g., termination) of the designed integration schema. Second, the decision procedure can be extended by means of a lemma speculation mechanism in such a way that it is capable of tackling problems falling outside the scope it has been originally designed for. Devising sound, terminating, complete (at least for certain subclasses of formulae), and efficient mechanisms to extend decision procedure is a very difficult task.

Constraint Contextual Rewriting [6, 15] is a new form of contextual rewriting, based on the integration of a linear arithmetic decision procedure in the rewriting activity of the Boyer and Moore prover, where the decision procedure is allowed to access and modify the rewriting context. Under the assumption that certain interface functionalities (satisfying some abstract properties) are provided by the decision procedure DP, CCR enjoys the following properties (which have been formally stated and proved): it is parametric in DP, it is sound, and it is terminating. The formalization of CCR relies on the notion of Contextual Reduction System (CRS), which generalizes the standard notion of an abstract reduction system. Informally, a CRS is defined by a set of ternary relations together with a set of inference rules that define the relations by (mutual) induction. The concept of CRS is the starting point for formally specifying and reasoning about integration schemas between rewriting and decision procedures.

CCR is at the core of the simplifier of RDL [6] (cf. Task 3.4), a fully automatic theorem prover for the quantifier-free fragment of first-order logic with equality.

1.2.g The MathSat Framework

Motivations and Goals

Many real-world problems require the ability of reasoning efficiently on formulae which are boolean combinations of boolean and unquantified mathematical propositions, on integer or real variables. (Noteworthy examples come from the domains of compilers design [224], temporal reasoning [4], resource planning [272], automated verification of systems with numerical data [102] or of timed and hybrid systems [200, 23], software and protocol design and verification [121, 246].) This ability requires an efficient combination of *boolean reasoning* and *mathematical solving* capabilities.

From the viewpoint of boolean reasoning (SAT), in the last years we have witnessed an impressive advance in the efficiency of SAT techniques, which has allowed to solve previously intractable problems. Unfortunately, simple boolean expressions are not expressive enough for representing most of the real-world domains listed above.

From the viewpoint of mathematical solving, in the last years also mathematical solvers like computer-algebra systems and constraint solvers have very much improved both in expressivity and in efficiency, being able to solve classes of problems which were previously unsolvable or intractable. Unfortunately, mathematical solvers cannot handle efficiently problems involving heavy boolean search —or do not handle them at all— so that most of the real-world domains above are out of their reach too.

MathSat

MathSat [129, 22, 21] is a general framework proposed by ITC-IRST for efficiently integrating boolean reasoning and mathematical solving tools. MathSat consists of a formal framework, a generalized algorithm and architecture for integrating boolean reasoners and mathematical solvers so that they can efficiently solve boolean combinations of boolean and mathematical propositions. The basic idea underlying the MathSat approach is to decompose the search into two orthogonal components, namely, one purely propositional component and one ”boolean-free” domain-specific component (e.g., a mathematical component). The fist component is based on a SAT solver, typically a (modified) Davis Putnam Longemann Loveland (DPLL) procedure. (The DPLL procedure has been shown to be preferable to alternative approaches proposed in the literature, such as , e.g., the ones based on Disjunctive Normal Form (DNF), Tableau Search Graph, or (Ordered) Binary Decision Diagrams (OBDDs).)

MathSat describes many techniques to optimize the integration, and highlights the main requirements SAT tools and mathematical solvers must fulfill in order to be integrated correctly and to achieve the maximum benefits from their integration. Specifically, a suitable SAT tool complies with the following crucial requirements:

- generation of *complete* collections of boolean assignments;
- generation of *non-redundant* collections of boolean assignments;
- *lazy* (i.e., one at a time) generation of boolean assignments;
- optimal space performance;
- good integration with the mathematical solver component (e.g., it allows for several run-time search optimizations).

A suitable mathematical solver complies with the following crucial requirements:

- be *incremental*, that is, be able of reusing the computation of previous calls on sub-problems without restarting from scratch;
- be able to provide *failure information* which can drive the boolean reasoner in pruning its search.

The ultimate goal of the MathSat framework is to develop tools able to handle real-world problems in complex domains like those described above. From the viewpoint of boolean reasoning, SAT tools can be extended in such a way they can handle also mathematical concepts and operators. From the viewpoint of mathematical solving, computer algebra systems and constraint solvers can be enriched by very efficient boolean reasoning capabilities.

Modeling Systems within the MathSat Framework

As evidenced in [22, 21], a significant amount of existing procedure used in various application domains can be modeled within the MathSat framework. These procedures either are purely symbolic or combine symbolic and numeric techniques. This will be discussed with more details in Task 3.4.

The MathSat Procedure

The MathSat solver [20, 23], developed by ITC-IRST, is a state-of-the-art solver based on the MathSat framework, able to reason on boolean combinations of linear arithmetic formulas. The efficiency of MathSat is due both to the tight integration of the boolean and mathematical solving subroutines, and to the layered structure of the mathematical decider, which is organized into levels dealing with subclasses of formulas of increasing complexity. This will be discussed with more details in Task 2.3.

Applications

The MathSat solver has been used to address verification problems in different domains, e.g., in temporal reasoning [20] and timed systems [23]. In particular, the verification of timed systems is a very important and challenging problem, in that it combines the challenge of handling finite-state variables, which is typically encoded as a boolean deduction problem, with the problems related to time elapsing, which are encoded into mathematical constraints on real variables representing absolute time and clocks. This will be discussed with more details in Task 3.2.

1.2.h The $\lambda Clam$ Proof Planning System

During his stay as a young visiting researcher (YVR) in Edinburgh, J. Zimmer integrated the proof planning system $\lambda Clam$ into the MathWeb-SB [114]. Due to this integration, $\lambda Clam$ can not only use the services

Name	ripple
Context	Rewrite Theory
Types	
Input	omdoc: OMDoc;
Output	result: OMDoc
InConstraints	elements(omdoc, Elements), lemmas(Elements, RewRules) member(sequent(_), Elements), not(RewRules = nil).
OutConstraints	elements(result, [Sequent]), not(member(Rule, RewRules), applicable(Rule, goal(Sequent)).
TextDescriptions	Tries to reduce the difference between the goal of the given OMDoc sequent and one of its hypotheses using $\lambda Clam$s step_case method and the rewrites given as lemmas in the OMDoc.

Figure 5: The rippling service offered by $\lambda Clam$

of any reasoning specialist already integrated in the MathWeb-SB, such as the CAS Maple, but can also offer its theorem proving expertise to other systems in the MathWeb-SB. First, $\lambda Clam$ offers an inductive theorem proving service to the MathWeb-SB which takes a problem description formulated in OMDoc as an input and runs $\lambda Clam$ on the given problem. Second, the rippling heuristics of $\lambda Clam$ [241] is offered as a service that takes a set of rewrite rules and a proof planning sequent as an input and applies the rippling method of $\lambda Clam$ with the given rewrites. The two services offered by $\lambda Clam$ are new examples for mathematical services offered by the MathWeb-SB that have not been described formally until now.

However, we tried to use the description language LARKS described in Section 1.2.d to give a first characterization of the rippling service offered by $\lambda Clam$ (see figure 5).

1.2.i Discussion

There is a growing interest in combining the reasoning and computational capabilities of heterogeneous systems, like deductive systems and computer algebra systems. In fact, state-of-the-art tools are the result of many man-years of careful development and engineering, and usually they provide a high degree of sophistication in their respective domain. However, they often perform poorly when applied outside the domain they have been designed for.

We have investigated two complementary approaches to pursue the above goal. On the one hand, we have explored the possibility of enhancing the capabilities of existing systems (see Sections 1.2.f, 3.4.b and 1.2.h). On the other hand, we have studied the possibility of integrating existing systems by means of a suitable infrastructure providing service exchange (see Sections 1.2.b, 3.4.d, 1.2.d and 1.2.e). A particular emphasis has been devoted to the definition of frameworks suitable for making mathematical services accessible over the web.

Unfortunately service integration between different systems is not an easy task. The main difficulty is that most of the existing reasoning systems are conceived and built as stand-alone systems to be used by humans users. Moreover, if the logical services provided by the component reasoning systems are not interfaced in a proper way, then the logical services provided by the compound systems may be unsound. This is something which must be carefully avoided, particularly in all the application domains where the correctness is of paramount importance as, e.g., in the formal verification of safety or security critical systems.

For this purpose, a reasonable amount of work has been devoted to the identification of the infrastructure (i.e., languages, protocols, semantic specifications and architectural schemata) needed to implement service exchange, and to the investigation of the requirements that the infrastructure should satisfy. During the workshop on *Mathematical Web Services* [187] organized by RISC in November 2002, a joint discussion has developed between partners of the CALCULEMUS project and partners of the MONET [201] project. As a result of the discussion, the following preliminary list of requirements for web-based mathematical services has been singled out:

- mathematical services are similar to web services, in that: each service is described by XML-based meta-information; information is published, it can be discovered and queried; clients use information to find appropriate services and connect to them;
- this process should be based on existing web technologies and standards as much as possible;
- each service description can be decomposed into multiple parts describing different facets of the service, each facet may have multiple formalization levels, from informal text for human readers up to formal statements that are machine-understandable and thus have a precise semantics;
- descriptions and facets may be organized into different (multiple) taxonomies such that a coarse pruning of the search process is possible in the discovery process;
- when querying, the client specifies whatever information is available (e.g., simple or multiple taxonomy concepts, formal behavior description); the broker can return results with different degrees of confidence.

This list will be furtherly discussed and taken as a starting point for future work on this task.

Task 2.1: Integration of CASs and DSs via Protocols

TASK LEADER: UKA
SCIENTISTS IN CHARGE: JACQUES CALMET
RESEARCH TEAM: USAAR, UKA, TUE, ITC-IRST

2.1.a Overview

The main goal of this task is to provide a semantics to interacting mathematical services. A first trivial remark is that this problem has been and still is investigated through several approaches. It is enough to query the web with the two key-words: semantics and mathamatics to be flooded by informations. One may quote SGML, XML, OPENMATH, MathML, SMPS and many others. Some references on such works are provided elsewhere in this report.

To provide interfaces between existing mathematical services requires an open software architecture. Among the aspects that must be further investigated are messaging and communication facilities, information modeling, infrastructure, mathematical service modeling application and knowledge base integration, development and management tools. Some of these requirements are not specific to mathematical knowledge and have been studied in very different applications, for example an Information Bus and Enterprise Toolkit in manufacturing, construction, and banking sectors [249].

When this project was initialized, there was a lack of software environments, languages and standards for interfaces between systems for mathematical computation. The reasons are manyfold:

1. CAS and TPS are designed, implemented and validated as stand-alone systems,
2. many systems are copyrighted and allow neither communication nor external access to internal methods,
3. they do not provide interfacing,
4. a CAS is never semantically sound. Thus, to provide a semantics is a required goal when interfacing any computing modules.

The solution has been to link mathematical services through networking methodologies such as communication languages, information exchange, and common knowledge representation.

A communication language defines how mathematical information can be exchanged among services. It must be recognized by each system in order to translate the information into their internal representation. Appropriate languages can either be the input language or internal encoding of one of the involved systems or standardized communication languages. Along these lines several communication languages for interfaces between software systems exchanging mathematical information have been developed. The better known is most probably OPENMATH.

Cooperation among several software systems can be achieved with indirect, unidirectional and bidirectional communication. The goal of this task is to investigate how protocols can be defined to provide a semantic meaning and soundness to systems exchanging mathematical information. This definition ought to give a hint why several other tasks in the network are in fact dealing with very similar problems. This is for example true when defining a context for a computation and is partly covered in task 1. Unidirectional and bidirectional communication protocols are designed when coupling directly different modules. Although there are no direct links between the services with indirect communication, interaction is possible when systems can communicate with a common user interface, central unit, mediator or evaluator. This approach has been investigated within the KOMET system.

A simplifying introduction is to say that a semantic can be provided through at least 3 approaches that are shortly described here under.

1. Define a mathematical software bus,
2. define a context from which a semantic can be derived,
3. formulate the problem as a knowledge representation paradigm.

These approaches are shared by several of the partners. Indeed, they lead to introduce multi-agent systems, contexts, ontologies to just quote a few features. References on Logic Broker, MathWeb-SB, OPENMATH and similar works are given throughout this midterm report.

2.1.b Open Mechanized Symbolic Computation Systems

A preliminary introduction to our work is given in [86]. The mathematical software bus and the context approaches are based on a joint work ([51]) with IRST on OMSCS (Open Mechanized Symbolic Computation Systems) which is a generalization of OMRS introduced by Giunchiglia and Talcott. We outline first OMSCS.

OMSCS A symbolic mathematical service is a software able to conduct useful and semantically meaningful two-way interactions with the environment. A symbolic mathematical service should be structurally organized as an OPEN ARCHITECTURE able to provide services like, e.g., proving that a formula is a theorem, or computing a definite symbolic integral, and to be able, if and when necessary, to rely on similar services provided by other tools. The Open Mechanized Reasoning System (OMRS) architecture was introduced [128] as a means to specify and implement reasoning systems (e.g., theorem provers) as logical services. In [90], this approach has been recast for the domain of symbolic computer algebra systems. OMSCS is the result of recasting together the two approaches.

The OMSCS Framework The specification of a service must be performed at various levels. At the object level, it is necessary to define formally the objects involved in the service, and the basic operations upon them. E.g., for a theorem prover, one must define the kind of assertions it manipulates, and the basic inference rules that can be applied upon them. Then, the control level provides a means to define the implementation of the computational capabilities defined at the object level, and to combine them. The control level must include some sort of "programming language" which is used to describe a strategy in the applications of modules implementing basic operations, therefore to actually define the behavior of the complex system implementing the service. Finally, the way the service is perceived by the environment, e.g., the naming of services and the protocols implementing them, is defined within the interaction level. This leads to the following architectural structure for reasoning and algorithmic services:

Reasoning Theory	=	Sequents + Rules
Reasoning System	=	Reasoning Theory + Control
Logical Service	=	Reasoning System + Interaction

Computation Theory	=	Objects + Algorithms
Computation System	=	Computation Theory + Control
Algorithmic Service	=	Computation System + Interaction

We synthesize these definitions into that of Symbolic Mathematical Service.

Symbolic Computation Theory	=	Symbolic Entities + Operations
Symbolic Computation System	=	Symbolic Computation Theory + Control
Symbolic Mathematical Service	=	Symbolic Computation System + Interaction

The interaction level of OMRS is for instance illustrated by the Logic Broker Architecture of Alessandro Armando et al. (see reference in Task 1).

Mathematical software bus The concept was introduced a few years ago by Richard Zippel and also in [91]. This work is still in progress. Our approach mostly consists in formalizing further OMSCS. We plan to extend it to interval arithmetic seen as a constraint programming problem.

Schemata and context Task one reports on some works conducted on defining a context, including one in collaboration with UGE (Constraint Contextual Rewriting in MAPLE by Ballarin and Alessandro, reference in task 1). We report here on a slightly different approach inspired directly by OMSCS and a schemata representation of algebraic algorithms. Schemata are nowadays found in several approaches such as MathML. We proposed several years ago to express algorithms as schemata.

The starting point is to consider that we compute with operators defined on given domain (types) that have specific properties (specifications). We call the triplet operator, domain, properties an abstract computational structure (ACS). Setting our approach in the framework of knowledge representation, a mathematical knowledge base consists of type schemata, algorithm schemata, algebraic algorithms, theorems, symbol tables, and normal forms. A schemata is a representation paradigm used in artificial intelligence. We adopt the specification language FORMAL-Σ ([95]) to represent the mathematical knowledge. It is well-suited to specify mathematical domains of computations. An algebraic specification introduces constants, operators and properties in their intended interpretation, and enables the reuse of subspecifications within a specification in accordance with the dependencies between particular specification modules of an ACS. It is based upon category theory. The next step is to define types, equations and algorithms through schemata's. A more detailed presentation and suitable references are found in [92].

A type schema represents such a module and consists of:

- *Name*, a unique identifier
- *Based-on*, a list of inherited ACS
- *Parameters*, a list of ACS which are parameters
- *Sorts*, a list of new sorts
- *Operators*, declarations of new operators
- *InitialProps*, initial properties.

These definitions build a based-on hierarchy of the mathematical domains of computation. One defines also an equation schemata Algorithms are also represented in terms of schemata. They allow the representation of meta-knowledge like:

- *Name*, a unique identifier of the schema with variable bindings
- *Signature*, describes the types of input and output
- *Constraints*, imposed on domain and range
- *Definition*, mathematical description of the output
- *Subalgs*, list of subalgorithms describing the embedded subtasks
- *Theorems*, describing properties of the algorithm
- *Function*, name of the corresponding executable algebraic function to compute the output.

Similarly to type and equation schemata, algorithm schemata build a hierarchy of specialized versions, and specializations inherit definitions and theorems from more general algorithms. New properties of algorithms can be derived then by a possibly coupled theorem prover. The concepts of specification language and schemata representation are at the of the CALVIN system.

A consequence of such an approach is to enable to define a context for a computation. This methodology is valid both when a CAS stands alone and when it is coupled to a TP. A context aims at making available the mathematical knowledge hidden in algebraic algorithms and in computational procedures. It is a methodology to improve the semantical soundness of symbolic computations. Although this is not straightforward to see, both FORMAL-Σ and OMSCS are closely related to the formulation of specifications through category theory. To stay as close as possible to OMSCS, we subdivide a context into three levels.

Similarly to type and equation schemata, algorithm schemata build a hierarchy of specialized versions, and specializations inherit definitions and theorems from more general algorithms. New properties of algorithms can be derived then by a possibly coupled theorem prover. The concepts of specification language and schemata representation are at the origin of an experimental CAS called CALVIN that was designed by students.

A consequence of such an approach is to enable to define a context for a computation. A context aims at making available the mathematical knowledge hidden in algebraic algorithms and in computational procedures. Although this is not straightforward to see, both FORMAL-Σ and OMSCS are closely related to the formulation of specifications through category theory. To stay as close as possible to OMSCS, we subdivide a context into three levels.

- The object level context collects the set of specifications linked to an operator and to its domain of definition. More generally, the goal is to access all of the information and knowledge that is either explicitly or implicitly available in the schemata representation of algebraic algorithms.
- At the control level the context is partly static and partly dynamic. The static part arises from the hierarchical organization of the object level schemata into equational schemata. At the root of this graphical hierarchy lies the "simplify" function. This generates thus a dynamical component that is associated to the simplification process.
- To define the interaction level part of context is still an open problem. A possible track is that the schemata approach leads to a concept of protocol to exchange mathematical knowledge and to check its soundness.

Mathematical Knowledge Our approach is based upon KOMET (Karlsruhe Open Mediator Technology), a system under development since 1994 [93]. Integrating data and knowledge from multiple heterogeneous sources (each one possibly with a different underlying data model) is not only an important aspect of automated reasoning but also of retrieval systems, in the widest sense, whose queries can span such multiple sources. One such source can be a CAS or a DS. A mediator integrates different sources on a semantic level by providing an integrated view spanning heterogeneous information sources. Different languages for building mediatory information systems have been proposed. We have selected (and extended) the Generalized Annotated Logic of M. Kifer and V.S. Subrahmanian [155]. We focus on some implementational details involving the embedding of Mathematica into our Mediator Architecture. First, the basic underlying mediator architecture is described. Then, some steps of the integration process are sketched.

The Mediator Architecture The basic architecture consists of MEDIATORS converting queries from a common format into more specialized queries, which are subsequently converted by TRANSLATORS (WRAPPERS) into the query language of the requested knowledge sources.

Such a translator must exist for each mediator–knowledge source combination. A translator also includes other functionalities for utilizing the knowledge source for the mediator such as caching extracted information or managing remote procedure calls.

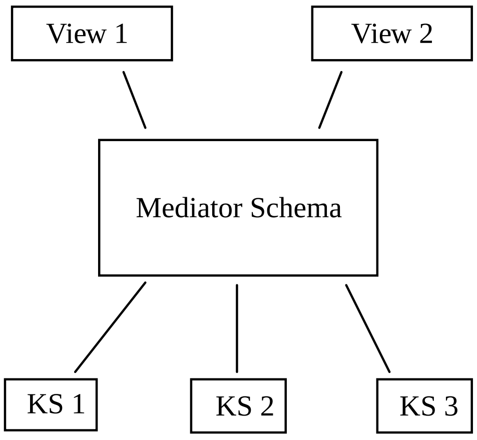


Figure 6: Bottom-up Integration of Heterogeneous Information Sources

The KOMET approach differs from other approaches in that the mediator is knowledge-based, i.e., a declarative rule based language for expressing the mediatory knowledge is being used.

Syntax and Semantics We sketch here the basic theory behind our approach to mediated systems. More detailed accounts are available in [93].

A *domain* D is an abstraction of databases and software packages and consists of three components:

1. a set Σ whose elements may be thought of as the data-objects that are being manipulated by the package in question,
2. a set { of functions on Σ — these functions take objects in Σ as input, and return, as output, objects from their range (which needs to be specified). The functions in { may be thought of as the predefined functions that have been implemented in the software package being considered,
3. a set of relations on the data-objects in Σ — intuitively, these relations may be thought of as the predefined relations in the domain, D.

A constraint Ξ over D is a first order formula where the symbols are interpreted over D. Ξ is either true or false in D, in which case it Ξ is sait to be solvable, or respectively unsolvable in D, where the reference to D will be eliminated if it is clear from context. The key idea behind a mediated system is that constraints provide the link to external sources, whether they are databases, object bases, or other knowledge sources. In particular, the full-fledged language involves annotations of predicate symbols according to GAP. Basically, an annotation corresponds to a multi-valued truth value from a complete lattice of truth values. A more detailed description is given in [155]. The annotations play an important role in resolving attribute value inconsistencies.

Embedding Mathematica into a Mediator Architecture MathLink is used to link Mathematica to KOMET. In contrast to the usual formalization of constraint domains (consisting of relations and functions), our approach relies on the more powerful concept of representing the functionality of the information source only as a set of relations, but where for each relation one or more *modes* are given. Intuitively, a mode describes the permitted binding patterns for the evaluation of a given relation. Note, that this is not a limitation, since functions can be represented as relations with appropriate modes. A relation mode is a tuple of argument modes, which specify the binding type of each argument required for the evaluation. The possible argument types are listed in the following table.

Argument mode	before	after
+	ground	ground
-	arbitrary	ground
?	arbitrary	arbitrary

"$+$" means that the argument must be ground before testing the constraint predicates, "$-$" means that the argument must be ground after calling the external function, and "?" means that the variable instantiation is arbitrary.

The use of modes implicitly imposes a certain order of evaluation on the constraint set and thus controls the data flow during the evaluation. With this approach, specific functionality of the CAS can be adequately introduced to the mediator. The proper modes ensure valid usage of the CAS functions. Consider as an illustrating example the interoperation between a relational database containing the coefficients of polynomials $a_1X2 + a_0$ over integers and Mathematica providing useful routines such as factoring polynomials. In spite of the task itself being rather simple, if not trivial, the example demonstrates how the mediator language can be used to interface the mediator to a CAS in a declarative manner:

$$\begin{array}{rcl}
\text{Coeff}(A_0, A_1) & \leftarrow & \text{Oracle::Polynomials}(A_0, A_1) \\
\text{Factorized\_Poly}(X) & \leftarrow & \text{Mathematica::Factor}(X), \\
 & & \text{Mathematica::Plus}(X, A_0, Z), \\
 & & \text{Mathematica::Times}(Z, A_1, Z), \\
 & & \text{Mathematica::Power}(Z, X, 2), \\
 & & \text{Coeff}(A_0, A_1)
\end{array}$$

with modes Polynomials$(+,+)$, Factor$(+)$, Plus$(-,+,+)$ and Times$(-,+,+)$. When issuing a query *Factorized_poly(X)*, the translator receives repeatedly an expression

$$\text{Factor}(\text{Plus}(A_0, \text{Times}(A_1, \text{Power}(X, 2))))$$

with tuples (A_0,A_1) from the Oracle database, which will result in the following set of MathLink function calls:

```
link = MLStart('math -noinit -mathlink');
MLPutFunction(link,'Factor',1);
MLPutFunction(link,'Plus',2);
MLPutFunction(link,'Times',2);
MLPutFunction(link,'Power,2);
MLPutSymbol(link,A0);
MLPutSymbol(link,A1);
MLPutSymbol(link,X);
MLPutInteger(link,2);
MLEndPacket(link);
```

It is exactly the purpose of the translator to generate a sequence of those MathLink function calls. We could support the OPENMATH interface as well.

This is only a sketch of how we can use a general purpose multiagent query system to interface a CAS to any other system.

An important feature of KOMET is that it introduces, after wrapping the query, a semantically sound representation of the information. Recent work has been devoted to extend such capabilities New operations on lattices where truth valued are defined have been defined. Composite distributive lattices as annotation domains for mediators provide such an extension. [94].

Another piece of work deals with the validation of queries. In this context, a query is a mathematical ”request”. But, very generally, we investigate whether a query is syntactically valid ([181]). It was straightforward to apply this validation into KOMET and the result is a system to validate web queries described in this paper. The dissertation thesis of Peter Kullmann “Wissensrepraesentation und Anfragebearbeitung in einer logikbasierten Mediatorumgebung” ([164]) looks at the optimization of queries in the context of KOMET and of logic programming. This leads to the general question of security in multiagent systems and in distributed computing ([119]).

Finally, KOMET enables to test some ideas related to ontologies. Indeed, ontologies arise directly from the definition of a context. Structuring ontologies by defining them as clusters of classes of knowledge led to a demonstration system called MASTER-Web ([88]).

Task 2.2: Enhancing the Reasoning Power of Computer Algebra Systems

TASK LEADER: RISC
SCIENTISTS IN CHARGE: BRUNO BUCHBERGER, WOLFGANG WINDSTEIGER, TUDOR JEBELAN
RESEARCH TEAM: UED, UKA, RISC

2.2.a Overview

Computer Algebra Systems (CAS) and Deduction Systems (DS) as they are available these days are typically designed as *stand-alone systems*, i.e. they expect the user to solve problems *entirely inside* one system. Mathematical practice, however, shows that "doing mathematics" is characterized many times by an interplay between *computing*, *proving*, and also *solving*, which neither CAS nor DS in todays state encompass.

State-of-the-art CAS can perform impressive algebraic or numeric computations with comparably low effort, they can simplify algebraic expressions, solve huge classes of equations, visualize mathematical objects, and they offer text-processing capabilities to different extents, which makes them heavily used in engineering and maths education. The question of *correctness* of the manipulations carried out by a CAS is, however, left without answer because the reasoning power of typical CAS is still very limited.

Unfortunately, this prohibits CAS from being used by a larger community of mathematicians for many more application areas. The limitations regarding the reasoning power of CAS are manifold:

- A typical CAS consists of a collection of algorithms that perform computations involving mathematical objects and an interface that allows easy access to these algorithms. The user manual of the CAS usually contains a (semi-) formal specification of each algorithm that gives the user an idea of which properties the output of an algorithm is supposed to fulfill with respect to the input. Now, given some algorithm A and appropriate input I, a CAS user would usually expect that the result of calling A with input I fulfills the output property stated in the specification. One characterizing feature that makes CAS very popular and heavily used especially in engineering and education is their capability to work with *parametrized input*. Intuitively, if the input to some algorithm depends on the parameter p, the user would expect the result to fulfill the output property *for all* values of p. There are numerous examples for computations performed by CAS, which are in fact not correct for all values of parameters, one of the easiest examples being the following: Given the equation $ax = b$, find solutions for x with parameters a and b. Most of the CAS available today will give $x \to b/a$ as the unique solution of the equation. In fact, this is not correct for all parameter values, consider e.g. $a = 0$ and $b = 1$.

- The correctness of most mathematical algorithms requires some properties of the input or of intermediate results that appear during the computation. Simple cases of testing such properties may only require e.g. testing numbers to be non-zero, which can easily be maintained again through computation. In some cases, powerful mathematical theorems are available in order to reduce checking a non-trivial property to comparably simple computations, e.g. checking whether a system of linear equations has a solution reduces to checking the determinant of the system to be non-zero. In general, however, checking properties of mathematical objects requires *proving*, e.g. the computation of the definite integral of an arbitrary function f by a certain algorithm might depend on continuity of f in some domain. Proving mathematical properties based on logical inference techniques is still beyond of what can be done in standard CAS.

- One central task of mathematics – in particular the branch of algorithmic mathematics – can be seen in developing algorithms in order to perform computations. The basis for each algorithm is always some mathematical theorem that guarantees the correctness of the algorithm. In many cases, the improvement of algorithms is then based on the development of "stronger" theorems that can be used to make algorithms perform more efficiently. A computer-aided approach to algorithmic mathematics therefore needs computer-support in both computing and also in proving mathematical theorems. Todays CAS provide algorithms for doing computations in various mathematical fields and they usually provide programming languages, which allow the implementation of new algorithms. The development of the mathematical theory behind the algorithms, however, is only supported poorly, if at all, in the CAS.

The aim of Task 2.2 is to enhance CAS with reasoning power, which can be attempted at different levels. We will briefly explain the possible directions into which CAS enhancements can be pursued.

Enhancement of CAS on the System Level

Enhancement on the system level can be achieved by *adding reasoning capabilities to algorithms built into the CAS*. Checking side-conditions on parameters in computer algebra algorithms is a typical example, where current CAS perform comparably poor. In the frame of the Calculemus network, the work of UNIGE contributes to this aspect of CAS enhancement: the Constraint Contextual Rewriting (CCR) framework developed by UNIGE can be used in order to integrate the evaluation mechanism of (the CAS) MAPLE, see [185] with an appropriate decision procedure for checking side-conditions, see [3] and [14].

In the concrete case, MAPLE's assume-facility has been used to link an enhanced decision procedure to the standard evaluation process of MAPLE. The `assume` command allows the MAPLE user to state certain facts about objects occurring as input to some algorithm from the MAPLE algorithm library. In the process of checking conditions during the execution of the algorithm, the evaluation procedure has access to all additional facts stated through `assume`. Of course, *checking a condition* φ using additional facts Φ needs more than just checking whether φ is contained in Φ. It finally requires *reasoning* whether φ can be *inferred* from Φ. In standard MAPLE, only weak reasoning techniques are applied for this purpose. Using CCR, a decision procedure increases the reasoning power when checking conditions during a computation.

This integration of reasoning and computation facilities increases the reliability of computations, because, through the more powerful reasoning engine, formulae of much more complicated nature can be handled when checking underlying conditions during computation. Computations involving parameters should particularly benefit from this improvement. It should be noted, however, that this approach needs access to the internals of the CAS since the evaluation procedure and, thus, the mechanism how to check for side-conditions, is normally hidden from the CAS user. Other CAS, e.g. *Mathematica*, see [188], offer similar mechanisms to pass additional assumptions when calling internal algorithms, but, due to the commercial nature of *Mathematica*, there is no way to get access to the internal evaluation procedure in order to enhance the reasoning capabilities in a similar manner. Another drawback lies in the fact that not all algorithms in MAPLE's library take into account additional knowledge given through `assume`, so the user never knows whether assumed facts were regarded properly during a computation or not.

Enhancement of CAS on the Theory Level

Enhancement on the theory level can be achieved by *adding proved knowledge about CAS functions* to the CAS knowledge base. This knowledge can then be used by the CAS and can lead to simplified computations. In the frame of the Calculemus network, the work of UED represents this aspect of CAS enhancement: The HR system, developed at UED, has been used to conjecture properties of functions available in the MAPLE algorithm library from empirical patterns detected in computational data produced by the CAS. HR has some built-in abilities to prevent the generation of conjectures that are trivially true in the first place. For more sophisticated reasoning it invokes a third party automated theorem prover, in the concrete case OTTER, and tries to prove the conjecture from first principles, i.e. the definitions of the functions involved. For an attempt to disprove a conjecture, the user can supply a set of objects that can be tried as counterexamples. The CAS is then used to calculate function values and HR checks, whether any of the tested values breaks the conjecture. For more advanced counterexample construction in algebraic domains, HR can call the MACE model generator.

This approach has been tested firstly in the area of number theory, see [108]. Given the number theoretic MAPLE functions `isprime(n)`, which checks whether `n` is a prime number, `tau(n)`, which returns the number of divisors of `n`, and `sigma(n)`, which gives the sum of all divisors of `n`, HR conjectures 137 properties involving `isprime`, `tau` and `sigma`. After a first round, only 22 conjectures remained that could neither be proved by OTTER nor disproved by MACE. Among those 22 formulae, one could identify some interesting properties, such as e.g. if `sigma(n)` is prime then `tau(n)` is prime. Different strategies for continuation can be considered at this point:

- As done in [108], one can find a generalized theorem and prove it by hand. Adding the proved theorem as an axiom further reduced the number of unsolved conjectures to 10 in a subsequent round.
- Alternatively, as a challenge for DS, one can try to apply probably more powerful *specialized theorem provers* in order to automatically prove the conjecture.

Any proven knowledge about `tau(n)` and `sigma(n)` can then be added to the knowledge base of the CAS, either by improving the internal algorithms in case they are accessible or by adding the knowledge on the user level. In general, improved performance of the CAS can be expected from adding more knowledge about the functions available in the CAS.

Enhancement of CAS on the User Level

Enhancement on the user level can be achieved by giving the CAS user the possibility to actually *prove mathematical statements using prove techniques from logic within the CAS* in addition to the computing facilities that each CAS offers. In the frame of the Calculemus network, the work of RISC represents this aspect of CAS enhancement: The THEOREMA system, see [75], is an add-on package for the widespread popular CAS *Mathematica* that allows the user to formulate mathematical theorems and prove them entirely within the *Mathematica* environment.

THEOREMA is implemented in *Mathematica*'s native programming language, which is based on *pattern matching* and *rewriting*. Since the release of *Mathematica* 3.0 in 1996, *Mathematica* can handle two-dimensional input and output containing arbitrary characters known from traditional mathematics and its programming language even allows to access and enhance input and output facilities in the *Mathematica* user front end. These are the key features that qualify the *Mathematica* system as a basis for building up a software system providing computer-support through the entire cycle of mathematical activity. The overall design goal of the THEOREMA system is to support the working mathematician in all phases of mathematical activity in a human-like style, both in input and output. THEOREMA allows to

- enter mathematical formulae in traditional mathematical fashion,

- structure formulae into definitions, axioms, lemmata, theorems, etc. which can be organized in nested theories, and, most importantly,
- prove mathematical theorems with respect to a given knowledge base,
- compute (simplify) mathematical terms or formulae with respect to a given knowledge base, and
- solve mathematical formulae with respect to a given knowledge base.

The THEOREMA system is a *multi-method system*, i.e. it provides different (specialized) proving / computing / solving methods for different purposes. Concerning the aspect of proving, it distinguishes two classes of provers, black-box provers and white-box provers. Black-box provers typically transform the original prove problem into some other problem, for which solution algorithms already exist. As an example, the Gröbner basis prover transforms a prove problem given as universally quantified boolean combination of polynomial equations over the complex numbers into the problem of deciding whether a system of polynomial equations has a solution, which can be done by testing the Gröbner basis of the system to be $\{1\}$. Another category of black-box provers is offered in the THEOREMA-system through linking existing external automated provers, e.g. OTTER. Black-box provers usually tell the user whether some mathematical statement is true or false but *do not provide evidence why this is so*.

White-box provers, on the other hand, try to obtain a mathematical proof in a style like a formally well-trained mathematician would write up the proof. Evidence for thruth or falsity of the statement under consideration is given by listing a sequence of logical inference steps, which show *why* the statement follows logically from the assumptions. In this class, THEOREMA provides *general provers* for first order predicate logic, proving by case distinction, proving equalities by simplification or equational proving and *theory specific provers* for induction on natural numbers, tuple induction, or set theory. These general and theory specific provers are designed in a modular structure so that they can be combined to more powerful provers.

For milestone 2.1 we decided to go towards enhancement *on the user level*, i.e. to *embed* reasoning facilities into an existing CAS by implementing various general and special theorem provers that interact with the algebraic algorithms available in the CAS. The THEOREMA system, developed at RISC on top of *Mathematica*, is surely the most advanced system propagating this approach. We will give a more thorough description of this system below.

2.2.b The THEOREMA System

In this section, we will give a brief description of the distinctive features of the THEOREMA system. We will emphasize on THEOREMA's proving capabilities and base the presentation of individual system components on typical examples. During the first example in Section 2.2.b, we will also explain the THEOREMA user interface, which allows input and output of mathematical text in a comfortable and easy-to-read form. In Sections 2.2.b to 2.2.b we will describe latest developments that have been added to the system in the frame of the Calculemus project. In order to make this presentation self-contained, we give an overview on the most important aspects of THEOREMA in Section 2.2.b. In particular, the issue of integrating *proving* capabilities with *computing* capabilities available from the underlying *Mathematica* system will be discussed in this concluding section. For an overview on the system philosophy behind THEOREMA, we refer to [62], [67], [68], [146], [144], [145].

The PCS Paradigm for Automated Theorem Proving

For a quick summary, the level of quality in automated theorem proving which we achieved in the first phase is best documented by our examples from elementary analysis: With our new proof strategies, notably the "PCS" (= "proving, solving, computing") strategy, we manage to generate proofs of the typical theorems in elementary analysis (on the notion of limit, continuity, etc.) completely automatically and with almost no superfluous branches in the search space. Also, the proofs generated have a couple of distinctive quality features:

- The proofs are structurally simple and clear and resemble the proofs generated by humans.
- The proofs contain intermediate explanatory text in natural language. (At the moment, English and Japanese are the choices.)
- When browsing a proof on the screen, the level of proof details shown can be controlled by the user. Also, various auxiliary information, e.g. on the definitions used, can be made visible in pop-up windows.
- The proofs generated show the terms constructed in the proof of existential subgoals explicitly and thus, mathematically, are more telling than the usual proofs shown in math textbooks.

In the automated theorem proving community, the automated generation of proofs in elementary analysis is considered to be an important benchmark problem and, so far, was beyond the capabilities of theorem proving systems. Thus, we think that being able to produce such proofs completely automatically and with distinctive quality features is a good documentation of the progress that has been achieved.

We now show one typical example in all details, namely the proof of the theorem that the limit of the sum of two sequences is the sum of the limits of the individual sequences.

The THEOREMA system is implemented on top of the well-known CAS *Mathematica*, using the high-level programming language available in *Mathematica*. The THEOREMA system can therefore access all components of *Mathematica*, most importantly its powerful user-interface. Since the release of *Mathematica* 3.0 in 1996, the *Mathematica* front-end supports traditional mathematical input and output in two-dimensional form containing also special characters commonly used in mathematical texts. Through its programming language, it allows even customization of the input parser and the typesetting of mathematical expressions, i.e. we can extend the class of expressions recognized by standard *Mathematica* with arbitrary expressions that seem convenient for a software system supporting the entire mathematical problem solving process. As for supporting *reasoning*, one will need the possibility to enter structured mathematical knowledge into the system.

Structured mathematical knowledge is more than just formulae and terms. In usual mathematical texts, mathematical knowledge is presented in the form of definitions, axioms, theorems, lemmata, and the like, which enrich a pure formula with additional information like a description of the symbols occuring in the formula, conditions for the variables, under which the formula holds, labels for referencing the formula later in a proof or a computation. This additional information is commonly given in natural language. The THEOREMA *formal text language* gives a formal frame for structuring mathematical information in this way. Each structural entity is called an *environment*, inside an environment there is the possibility to declare certain symbols as variables, put conditions on variables, or assign names to formulae. For later reference each environment also has a name. Let us now consider the definition of the *convergence of a sequence* and the theorem that the sum of two convergent sequences converges, which would appear in conventional mathematical texts as follows:

Definition 1 (convergence) *For any sequence f and any a, we say f* converges to *a if and only if for all $\epsilon > 0$ there exists an N such that for all $n > N$*

$$|f_n - a| < \epsilon$$

We call f convergent *if and only if there exists an a such that f converges to a.*

Theorem 1 (convergence of sum) *For any two convergent sequences f and g the sum-sequence $f + g$ converges.*

We analyze the ingredients of these two entities and discuss their representation in the THEOREMA formal text language:

- Keywords (**Definition**, **Theorem**) indicate the "type" of mathematical knowledge, labels are introduced for later reference. The THEOREMA formal text language provides various keywords for mathematical environments, such as Definition, Theorem, Lemma, Proposition, Theory, etc.
- In the traditional definition, a phrase such as "For any sequence f and any a" tells that in the subsequent formulae any free occurences of f and a are to be understood *universally quantified*. The THEOREMA formal text language provides the construct `any[...]` for this purpose.
- Conditions may be put on some of the universally quantified variables. The THEOREMA formal text language provides the construct `with[...]` for easy reading.
- Superfluous natural language ingredients, such as "For", "we say", and "we call", do not have a counter-part in THEOREMA since they can be omitted.
- *f converges to a* if and only if ... is a definition for a *new* binary predicate, e.g. converges$[f, a]$. Alternatively, we could invent some infix notation, e.g. $f \to a$, for converges$[f, a]$.
- For the core of a mathematical environment, i.e. the formula itself, THEOREMA provides a concrete natural syntax for higher order predicate logic. Input of formulae in natural syntax (special symbols like $\forall$, $\exists$, sub- and superscripts, under- and overscripts and the like) are supported by the standard *Mathematica* front end, special definitions for the concrete syntax have been added in the frame of THEOREMA.
- Definitions of new predicates can be written in predicate logic in the form $lhs :\Leftrightarrow rhs$.
- the phrase "for all $\epsilon > 0$ there exists an N such that for all $n > N$: $|f_n - a| < \epsilon$" can be written in pure predicate logic in the form

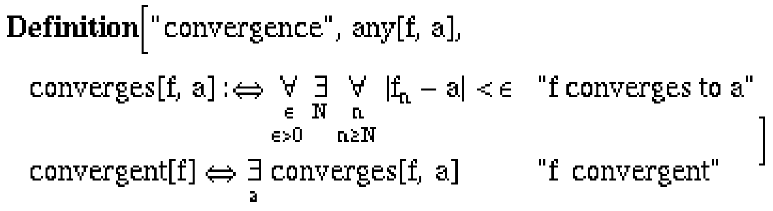
  $$\underset{\epsilon>0}{\forall}\ \underset{N}{\exists}\ \underset{n>N}{\forall}\ |f_n - a| < \epsilon \quad .$$

  Depending on the taste of the author, this form might be used even in traditional text instead of the natural language formulation for quantified formulae.
- Following the same rules, we could write the second part of the definition as

  $$\text{convergent}[f] :\Leftrightarrow \underset{a}{\exists}\ \text{converges}[f, a] \quad .$$

- A THEOREMA environment may contain more than one formula, each formula can optionally be given a label. If no label is given explicitely, formulae are labeled using ascending numbers.
- As soon as an environment `keyword` with label `label` has been entered in a THEOREMA session, its contents can be accessed through `keyword[label]`.

In THEOREMA, Definition 1 and Theorem 1 can be entered as shown below[6]:



Definition["convergence", any[f, a],

converges[f, a] :⇔ $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}\ |f_n - a| < \epsilon$ "f converges to a"

convergent[f] ⇔ $\underset{a}{\exists}$ converges[f, a] "f convergent"]

[6]We will provide screenshots from a THEOREMA session in the first examples to show the real appearance in the *Mathematica* front-end. In later examples, we will show THEOREMA-environments typeset in LaTeX.

$$\textbf{Theorem}\Big[\text{"convergence of sum"},\ \text{any}[f, g],\ \text{with}\Big[\bigwedge\begin{cases}\text{convergent}[f]\\ \text{convergent}[g]\end{cases}\Big],\ \text{convergent}[f+g]\Big]$$

Of course, we cannot prove Theorem "convergence of sum" if nothing is known about the notions appearing in the definition and the theorem. It is interesting to note that the following knowledge is sufficient for the proof:

Definition["sum of sequences", any[f, g, n],
  $(f + g)_n = f_n + g_n$]

Lemma["distance of sum", any[$x, y, z, t, \delta, \epsilon$],
  $|(x + y) - (z + t)| < \delta + \epsilon \Leftarrow |x - z| < \delta \wedge |y - t| < \epsilon$]

Lemma["max greater", any[$m, M1, M2$],
  $m \geq \max[M1, M2] \Rightarrow (m \geq M1 \wedge m \geq M2)$]

The first formula defines the sum of sequences component-wise. The second formula, essentially, formulates continuity of addition on real numbers. The third formula formulates an easy property of the maximum function, which also could be considered as part of the definition of the maximum function.

In THEOREMA, we now have the possibility to combine knowledge to "theories". In fact, the "Theory" construct is recursive and, thus, we can build up arbitrarily nested hierarchies of knowledge bases. In our example, we combine the individual formulae of the knowledge base to one theory called "convergence".

Theory["convergence",
  Definition["convergence"]
  Definition["sum of sequences"]
  Lemma["distance of sum"]
  Lemma["max grater"]]

The only thing we have to do now for the automated generation of a proof of the theorem is to call one of the THEOREMA provers, in this case the "PCS" prover, and ask it to prove the Theorem["convergence of sum"] using the knowledge base Theory["convergence"]:

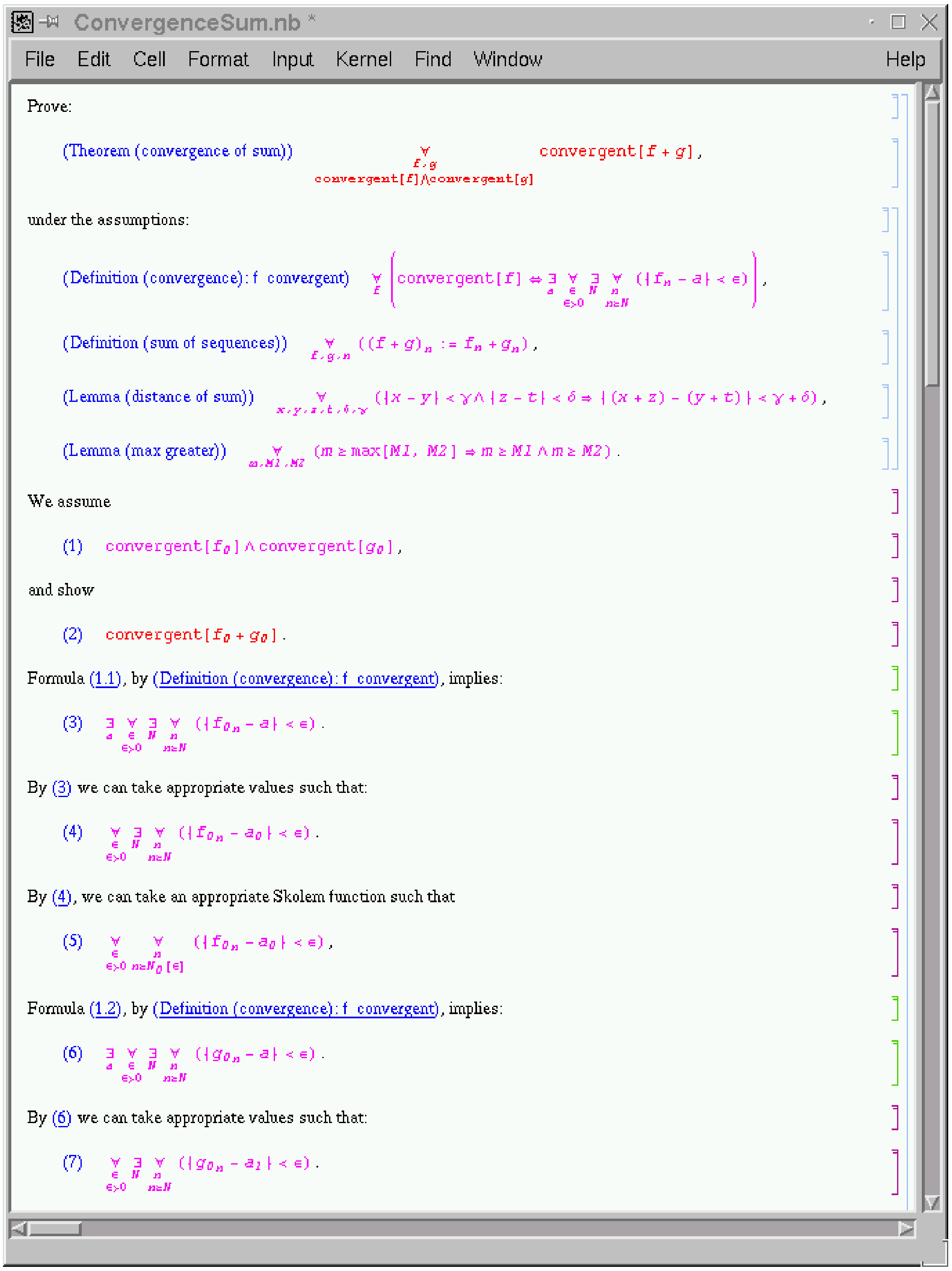
`Prove`[Proposition["convergence of sum"], using→Theory["convergence"], by→PCS]

After a few seconds, the proof shown in Figures 7 to 9 will appear in an extra notebook. Note that the entire proof text, including the explanatory comments in English, is produced fully automatically without any user interaction.

In order to obtain a quick impression about the typical achievements we made in the first phase of the project, it is worthwhile to study this proof in detail[7]. Some comments on the most important phases in the proof are:

- The proof starts with a "Prove" phase, in which THEOREMA structures the proof applying basic inference rules of predicate logic, see (1) and (2).
- Then it applies the definition of convergent in a rewriting style ("Compute" phase), see (3) and (6).
- Then again, basic inference rules of predicate logic are applied, see (4) and (7).
- Now it comes to a crucial phase, in which it applies Skolemization, see (5) and (8). This will be important for being able, later, to "construct" the final solving terms. Note that we apply Skolemization

[7]To shorten the proof we compressed the two parts of Definition["convergence"] into only one formula, which does not essentially change the proof!



Prove:

(Theorem (convergence of sum)) $\underset{\substack{f,g \\ \text{convergent}[f] \wedge \text{convergent}[g]}}{\forall} \text{convergent}[f+g]$,

under the assumptions:

(Definition (convergence): f convergent) $\underset{f}{\forall}\left(\text{convergent}[f] \Leftrightarrow \underset{a}{\exists}\ \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n \geq N}}{\forall} (|f_n - a| < \epsilon)\right)$,

(Definition (sum of sequences)) $\underset{f,g,n}{\forall} ((f+g)_n := f_n + g_n)$,

(Lemma (distance of sum)) $\underset{x,y,z,t,\delta,\gamma}{\forall} (|x-y| < \gamma \wedge |z-t| < \delta \Rightarrow |(x+z)-(y+t)| < \gamma+\delta)$,

(Lemma (max greater)) $\underset{m,M1,M2}{\forall} (m \geq \max[M1, M2] \Rightarrow m \geq M1 \wedge m \geq M2)$.

We assume

(1) $\text{convergent}[f_0] \wedge \text{convergent}[g_0]$,

and show

(2) $\text{convergent}[f_0 + g_0]$.

Formula (1.1), by (Definition (convergence): f convergent), implies:

(3) $\underset{a}{\exists}\ \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n \geq N}}{\forall} (|f_{0_n} - a| < \epsilon)$.

By (3) we can take appropriate values such that:

(4) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n \geq N}}{\forall} (|f_{0_n} - a_0| < \epsilon)$.

By (4), we can take an appropriate Skolem function such that

(5) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n \geq N_0[\epsilon]}}{\forall} (|f_{0_n} - a_0| < \epsilon)$,

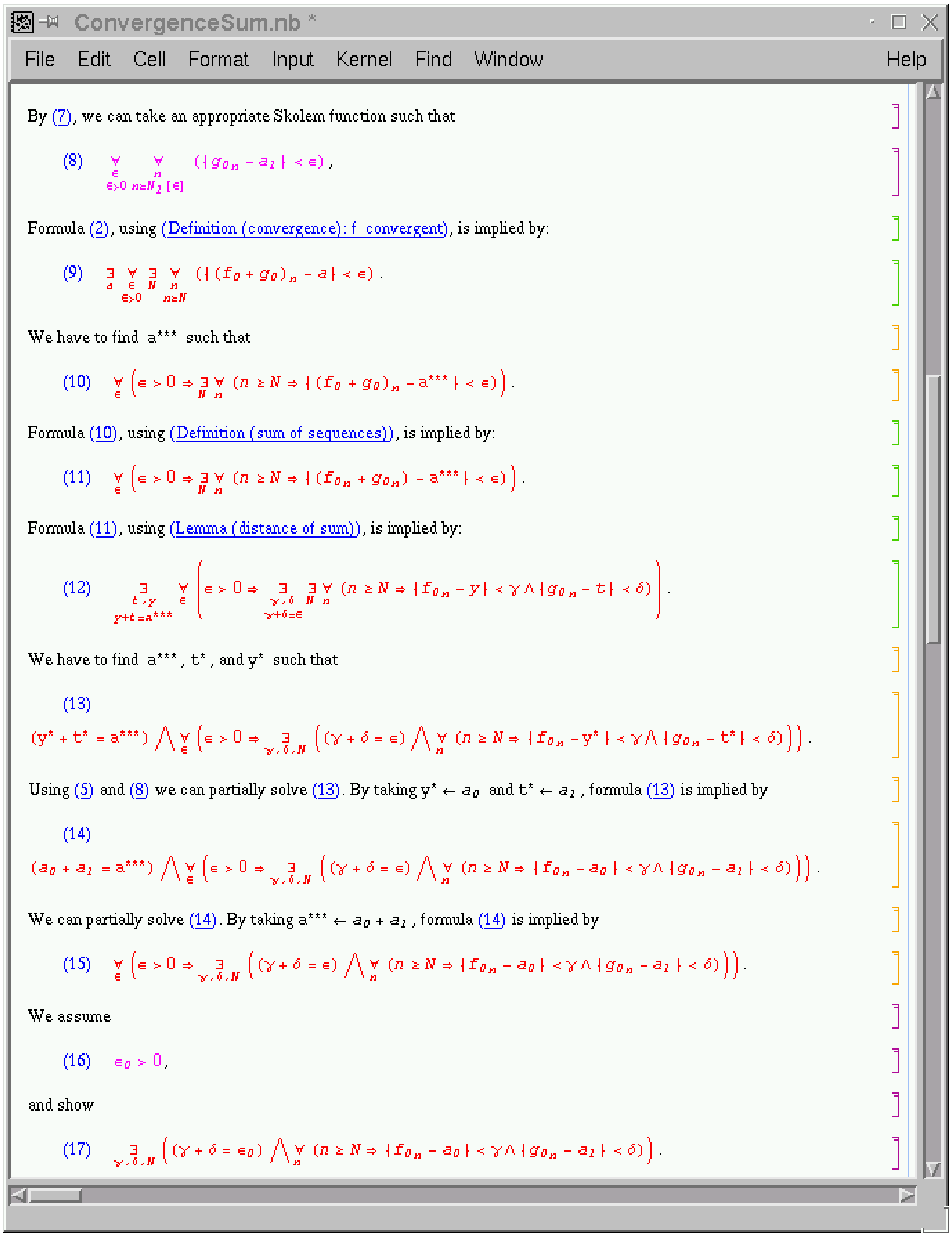
Formula (1.2), by (Definition (convergence): f convergent), implies:

(6) $\underset{a}{\exists}\ \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n \geq N}}{\forall} (|g_{0_n} - a| < \epsilon)$.

By (6) we can take appropriate values such that:

(7) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n \geq N}}{\forall} (|g_{0_n} - a_1| < \epsilon)$.

Figure 7: Proof “Convergence of Sum” (part 1)



By (7), we can take an appropriate Skolem function such that

(8) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \; \underset{\substack{n \\ n\geq N_1[\epsilon]}}{\forall} \left(|g_{0_n} - a_1| < \epsilon \right),$

Formula (2), using (Definition (convergence): f convergent), is implied by:

(9) $\underset{a}{\exists} \; \underset{\substack{\epsilon \\ \epsilon>0}}{\forall} \; \underset{N}{\exists} \; \underset{\substack{n \\ n\geq N}}{\forall} \left(|(f_0 + g_0)_n - a| < \epsilon \right).$

We have to find a^{***} such that

(10) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \; \underset{n}{\forall} \left(n \geq N \Rightarrow |(f_0 + g_0)_n - a^{***}| < \epsilon \right) \right).$

Formula (10), using (Definition (sum of sequences)), is implied by:

(11) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \; \underset{n}{\forall} \left(n \geq N \Rightarrow |(f_{0_n} + g_{0_n}) - a^{***}| < \epsilon \right) \right).$

Formula (11), using (Lemma (distance of sum)), is implied by:

(12) $\underset{\substack{t,y \\ y+t=a^{***}}}{\exists} \; \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{\substack{\gamma,\delta \\ \gamma+\delta=\epsilon}}{\exists} \; \underset{N}{\exists} \; \underset{n}{\forall} \left(n \geq N \Rightarrow |f_{0_n} - y| < \gamma \wedge |g_{0_n} - t| < \delta \right) \right).$

We have to find a^{***}, t^*, and y^* such that

(13)
$(y^* + t^* = a^{***}) \bigwedge \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{\gamma,\delta,N}{\exists} \left((\gamma + \delta = \epsilon) \bigwedge \underset{n}{\forall} \left(n \geq N \Rightarrow |f_{0_n} - y^*| < \gamma \wedge |g_{0_n} - t^*| < \delta \right) \right) \right).$

Using (5) and (8) we can partially solve (13). By taking $y^* \leftarrow a_0$ and $t^* \leftarrow a_1$, formula (13) is implied by

(14)
$(a_0 + a_1 = a^{***}) \bigwedge \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{\gamma,\delta,N}{\exists} \left((\gamma + \delta = \epsilon) \bigwedge \underset{n}{\forall} \left(n \geq N \Rightarrow |f_{0_n} - a_0| < \gamma \wedge |g_{0_n} - a_1| < \delta \right) \right) \right).$

We can partially solve (14). By taking $a^{***} \leftarrow a_0 + a_1$, formula (14) is implied by

(15) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{\gamma,\delta,N}{\exists} \left((\gamma + \delta = \epsilon) \bigwedge \underset{n}{\forall} \left(n \geq N \Rightarrow |f_{0_n} - a_0| < \gamma \wedge |g_{0_n} - a_1| < \delta \right) \right) \right).$

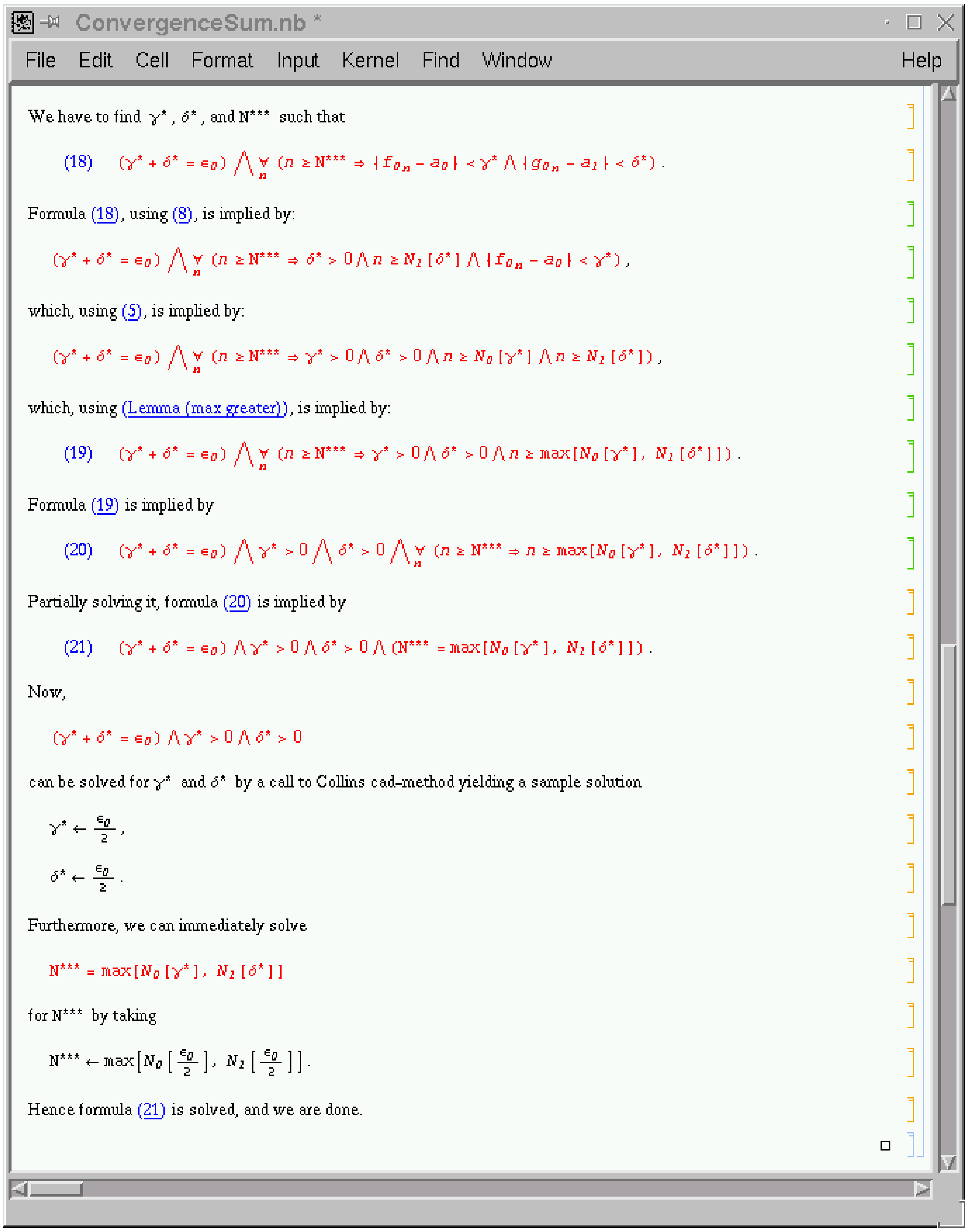
We assume

(16) $\epsilon_0 > 0,$

and show

(17) $\underset{\gamma,\delta,N}{\exists} \left((\gamma + \delta = \epsilon_0) \bigwedge \underset{n}{\forall} \left(n \geq N \Rightarrow |f_{0_n} - a_0| < \gamma \wedge |g_{0_n} - a_1| < \delta \right) \right).$

Figure 8: Proof “Convergence of Sum” (part 2)



We have to find γ^*, δ^*, and N^{***} such that

(18) $(\gamma^* + \delta^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N^{***} \Rightarrow |f_{0_n} - a_0| < \gamma^* \wedge |g_{0_n} - a_1| < \delta^*)$.

Formula (18), using (8), is implied by:

$(\gamma^* + \delta^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N^{***} \Rightarrow \delta^* > 0 \wedge n \geq N_1[\delta^*] \wedge |f_{0_n} - a_0| < \gamma^*)$,

which, using (5), is implied by:

$(\gamma^* + \delta^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N^{***} \Rightarrow \gamma^* > 0 \wedge \delta^* > 0 \wedge n \geq N_0[\gamma^*] \wedge n \geq N_1[\delta^*])$,

which, using (Lemma (max greater)), is implied by:

(19) $(\gamma^* + \delta^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N^{***} \Rightarrow \gamma^* > 0 \wedge \delta^* > 0 \wedge n \geq \max[N_0[\gamma^*], N_1[\delta^*]])$.

Formula (19) is implied by

(20) $(\gamma^* + \delta^* = \epsilon_0) \bigwedge \gamma^* > 0 \bigwedge \delta^* > 0 \bigwedge \underset{n}{\forall} (n \geq N^{***} \Rightarrow n \geq \max[N_0[\gamma^*], N_1[\delta^*]])$.

Partially solving it, formula (20) is implied by

(21) $(\gamma^* + \delta^* = \epsilon_0) \wedge \gamma^* > 0 \wedge \delta^* > 0 \wedge (N^{***} = \max[N_0[\gamma^*], N_1[\delta^*]])$.

Now,

$(\gamma^* + \delta^* = \epsilon_0) \wedge \gamma^* > 0 \wedge \delta^* > 0$

can be solved for γ^* and δ^* by a call to Collins cad-method yielding a sample solution

$\gamma^* \leftarrow \frac{\epsilon_0}{2}$,

$\delta^* \leftarrow \frac{\epsilon_0}{2}$.

Furthermore, we can immediately solve

$N^{***} = \max[N_0[\gamma^*], N_1[\delta^*]]$

for N^{***} by taking

$N^{***} \leftarrow \max\left[N_0\left[\frac{\epsilon_0}{2}\right], N_1\left[\frac{\epsilon_0}{2}\right]\right]$.

Hence formula (21) is solved, and we are done.

Figure 9: Proof “Convergence of Sum” (part 3)

only “in small doses” and not in the general way like this is done in resolution. This is crucial for keeping the proofs “natural”.

- Now the definition of convergent is applied to the goal, see (9). This is a rewriting (“Compute”) step.
- Now we come to a “Solve” phase, i.e. we have to construct a suitable a, see (10). Note that, by the introduction of a “find” constant (constants with asterisks), we can continue to work on the “outermost” symbols of formulae.
- Now we apply the definition of the sum of sequences, see (11), a “Compute” step.
- The next step is again crucial in our method: Matching the goal with the proposition on the continuity of addition is not directly possible. Instead we apply a new proof rule (which we call “semantic matching”) which forces a match to be possible on the expense of introducing new existential quantifiers (for t, y and γ, δ, respectively), see (12). By this, we again arrive at as “Solve” situation, which we again handle by solve constants, see (13).
- By unification using formulae from the knowledge base, we can instantiate some of the solve constants, see (14).
- Part of the goal formula is easy enough to read off from it the instantiation for the remaining solve constant, see (15). Note that, by this, the proof by itself *constructs* the value to which the sum sequence converges!
- After a standard “Proof” step, see (16) and (17), we again enter a “Solve” phase, in which we introduce solve constants for γ, δ, and N, see (18).
- (18) is now transformed by a couple of rewriting steps (“Compute” steps), using the skolemized (!) assumptions into a pure solve problem, see (20).
- This solve problem, in fact, is neatly decomposed into a part which can be handled by pure predicate logic and a part which is a solve problem with all constants over the real numbers (!), see (20). This is essential: Summarizing, what happened in the proof, was a reduction of the proof problem that contained variables over *functions* to a solve problem over the real numbers.
- Now we can call any of the powerful methods of computer algebra for solving constraints over the reals and we are done! (In the particular example, a relatively simple constraint solver would be sufficient, in more complicated examples the full potential of current constraint solvers is helpful.)
- Note that we do not show a trace of the constraint solver because, in an “exploration round” on the notion of convergence, it is absolutely “uninteresting” to see parts of the proof that refer to the earlier “exploration round” of proving and solving over the real numbers! We think it is of utmost importance for understanding the significance of formal proving to formulate the notion of “importance or unimportance of details” *relative* to a given exploration round.
- Note also that the proof does not only stop with saying that the final proof situation (which, actually, is a solve problem) can be handled (in this case, by a call to a black-box constraint solver) but, rather, it exhibits the solving term, which contains mathematically and didactically highly relevant and interesting information: In our case, the final solving term for N^{***} tells us a “method” how, if we know a “method” for finding the appropriate Ns for given ϵs for the input sequences, we also can find an appropriate N for given ϵs for the sum of the input sequences! In other words, the proof is not only a guarantee for the truth of the proposition but can also be considered as a “program synthesis” algorithm for constructing “methods” from “methods”. Note also that the term assigned to N^{***} is an algorithm (!) if, in concrete cases of input sequences f and g we know algorithms N_0 and N_1. All this interesting information on the solving terms, usually, is not produced in math text books although it would have enormous value for “constructive analysis” and also for the didactics of understanding the notion of convergence and its interaction with operations on sequences!

The Set Theory Prover

The THEOREMA set theory prover is a special prover available for proving theorems involving language constructs from set theory. The THEOREMA language provides syntax for various language constructs from set theory and some semantical knowledge for finite sets, which allows doing computations involving finite sets. Before we go into details on the set theory prover, we show some THEOREMA *computations* using semantics for sets from the THEOREMA language. Consider the definitions

Definition[“reflexivity”, any[$A, \sim$],
is-reflexive$_A[\sim]$:$\Leftrightarrow \underset{x \in A}{\forall}\ x \sim x$]

Definition[“relation sets”, any[$x, A, \sim$],
class$_{A,\sim}[x] := \{a \in A \mid a \sim x\}$ “class”
factor-set$_\sim[A] := \{\text{class}_{A,\sim}[x] \underset{x \in A}{|}\}$ “factor-set”]

introducing the notions of *reflexivity* on A for some binary relation $\sim$, the notion of the *class* (w.r.t. A and $\sim$) of x, and the notion of the *factor set* (with respect to $\sim$) of some set A, respectively. Using built-in knowledge about numbers available through *Mathematica* and built-in semantics for quantifiers and sets available from THEOREMA we can compute e.g. the class of 6 (with respect to $\{1, 2, \ldots, 7\}$ and the divisibility relation “|”) and the factor set of $\{1, 2, \ldots, 7\}$ (with respect to the divisibility relation “|”). In a THEOREMA session, these computations appear as

`Compute`[class$_{\{i \underset{i=1,\ldots,7}{|}\},|}$[6], using→Definition[“relation sets”],
built-in→ ⟨ Built-in[“Numbers”], Built-in[“Quantifiers”], Built-in[“Sets”]⟩]

$$\{1, 2, 3, 6\}$$

`Compute`[factor-set$_|$[$\{i \underset{i=1,\ldots,7}{|}\}$], using→Definition[“relation sets”],
built-in→ ⟨Built-in[“Numbers”], Built-in[“Quantifiers”], Built-in[“Sets”]⟩]

$$\{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 5\}, \{1, 7\}, \{1, 2, 4\}, \{1, 2, 3, 6\}\}$$

After some more computations, one might conjecture the following lemma:

Lemma[“factor set covers”, any[$A, \sim$], with[is-reflexive$_A[\sim]$]]
$\bigcup$ factor-set$_\sim[A] \supseteq A$]

The set theory prover can generate a proof of Lemma[“factor set covers”] fully automatically[8].

Prove:

(Lemma (factor set covers)) $\underset{A,\sim}{\forall} \left(\text{is-reflexive}_A[\sim] \Rightarrow \cup\text{factor-set}_\sim[A] \supseteq A\right)$,

under the assumptions:

(Definition (reflexivity)) $\underset{A,\sim}{\forall} \left(\text{is-reflexive}_A[\sim] :\Leftrightarrow \underset{x}{\forall} (x \in A \Rightarrow x \sim x)\right)$,

(Definition (relation sets): class) $\underset{A,x,\sim}{\forall} \left(\text{class}_{A,\sim}[x] := \left\{a \underset{a}{|}\ a \in A \wedge a \sim x\right\}\right)$,

(Def.(relation sets):factor-set) $\underset{A,\sim}{\forall} \left(\text{factor-set}_\sim[A] := \left\{\text{class}_{A,\sim}[x] \underset{x}{|}\ x \in A\right\}\right)$.

[8]In a THEOREMA session, the proof would appear in a separate window as already shown in Figures 7 to 9. We try to imitate its appearance as closely as possible in LaTeX.

We assume

(1) is–reflexive$_{A_0}[\sim_0]$,

and show

(2) $\bigcup$ factor–set$_{\sim_0}[A_0] \supseteq A_0$.

Formula (2), using (Definition (relation sets): factor-set), is implied by:

$\bigcup \left\{ \text{class}_{A_0,\sim_0}[x] \underset{x}{|}\ x \in A_0 \right\} \supseteq A_0$,

which, using (Definition (relation sets): class), is implied by:

(3) $\bigcup \left\{ \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x \right\} \underset{x}{|}\ x \in A_0 \right\} \supseteq A_0$.

Formula (1), by (Definition (reflexivity)), implies:

(4) $\underset{x}{\forall} (x \in A_0 \Rightarrow x \sim_0 x)$.

For proving (3) we choose

(5) $x1_0 \in A_0$,

and show:

(6) $x1_0 \in \bigcup \left\{ \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x \right\} \underset{x}{|}\ x \in A_0 \right\}$.

In order to show (6) we have to show

(9) $\underset{x2}{\exists}\ x1_0 \in x2 \wedge x2 \in \left\{ \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x \right\} \underset{x}{|}\ x \in A_0 \right\}$.

In order to solve (9) we have to find $x2^*$ such that

(10) $x1_0 \in x2^* \wedge \underset{x}{\exists} \left(x \in A_0 \wedge \left(x2^* = \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x \right\} \right) \right)$.

Since (5) matches a part of (10) we try to instantiate, i.e. let know $x := x1_0$.

Thus, by (10), we choose $x2^* := \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x1_0 \right\}$.

Now, it suffices to show

(12) $x1_0 \in A_0 \wedge x1_0 \in \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x1_0 \right\}$.

We prove the individual conjunctive parts of (12):

Proof of (12.1) $x1_0 \in A_0$:

Formula (12.1) is true because it is identical to (5).

Proof of (12.2) $x1_0 \in \left\{ a \underset{a}{|}\ a \in A_0 \wedge a \sim_0 x1_0 \right\}$:

In order to prove (12.2) we have to show:

(13) $x1_0 \in A_0 \wedge x1_0 \sim_0 x1_0$.

Formula (13), using (4), is implied by:

(14) $x1_0 \in A_0$.

Formula (14) is true because it is identical to (5). □

We briefly comment on the essential steps in the proof:

- The proof starts with a "Prove" phase, in which the universally quantified implication in the proof goal is reduced by natural deduction inference rules, see (1) and (2).
- In a "Compute" phase, the goal and the knowledge base are rewritten using the definitions in the knowledge base, see (3) and (4).
- The prover switches back again to a "Proof" phase, but now *special inference rules for set theory* are applied in order to eliminate set theoretic operators ("$\supseteq$" and "$\bigcup$") in the goal, see (5), (6), and (9).
- The existential goal (9) has the special structure $\underset{x2}{\exists}\ x1_0 \in x2 \wedge x2 \in \{\underset{x}{T_x \mid}\ P_x\}$, which is always the case after reducing a goal of the form $x1_0 \in \bigcup\{\underset{x}{T_x \mid}\ P_x\}$. Therefore we enter a set theory specific "Solve" phase, in which the existential quantifier is eliminated by introducing the solve constant $x2^*$, and the expansion of the inner membership $x2^* \in \{\underset{x}{T_x \mid}\ P_x\}$ introduces another existential quantifier (for x), see (10).
- The existential subformula in (10) in solved for x by unification with formulae in the knowledge base. In fact, in this example *matching* is sufficient, but we provide unification in this step for the general case. Having solved for x, the solve constant $x2^*$ can be instantiated from the equational subformula $x2^* = \ldots$ in (10), reducing the solve problem (10) again to a proof problem, see (12).
- The goal (12) is split using general predicate logic, subgoal (12.1) is trivially true, and subgoal (12.2) is handled first by a set theory specific prove rule, see (13).
- Finally, the goal (13) is proved by simple rewriting using implications from the knowledge base in a final "Compute" phase, see (14).

Another feature worth mentioning, though not appearing in the above proof, is the set theory specific "Compute" phase. In this phase, semantics for set theory specific language constructs available from the THEOREMA language is applied for rewriting. In particular, finite sets are computed to a canonic representation in exactly the same way as it is done in computations on the top-level when using `Compute` as shown at the beginning of this section. Moreover, built-in knowledge about certain numbers and number sets can be applied for simplification purposes, e.g. the fact that $6 \in \mathbb{N}$ holds when interpreting '6' as the built-in natural number six and '$\mathbb{N}$' as the built-in set of natural numbers. For details on the set theory prover, we refer to [268], [271], [270], and [267], an extensive case study of using the prover is given in [269].

The Equational Prover

The Theorema equational prover is a prover for unit equality problems. It consists of two parts - the prover kernel and the proof presenter. The kernel is an implementation of the unfailing completion procedure, see [26], with extensions to handle existential goals and using various simplifiers. The prover can be run on problems which are in purely equational form. All the equalities in the knowledge base of the problem are universally closed and each variable in the goal is either universally or existentially quantified. Handling of existential goals in the "Solve" phase is based on unification. The unification procedure has been extended in order to support special language constructs (*flexible arity symbols*, i.e. function and predicate symbols that can be applied to a flexible number of arguments, and *sequence variables*, i.e. variables that can be substituted by zero or an arbitrary number of terms) available in THEOREMA. It has been shown that unfailing completion remains a refutationally complete proving method if sequence variables occur in terms only in the last argument position, see [169]. The proof presenter is based on the Proof Communication Language – PCL, see [115]. We give again one example to illustrate the method (variables with overbar are sequence variables).

Prove:

(Proposition (goal)) $\underset{\bar{x}}{\exists}\,(\text{sort}[\langle 1,3,2,4\rangle] = \langle \bar{x}\rangle)$,

under the assumptions:

(Axiom 1) $\text{sort}[\langle\rangle] = \langle\rangle$,

(Axiom 2) $\underset{n}{\forall}\,(\text{insert}[n,\langle\rangle] = \langle n\rangle)$,

(Axiom 3) $\underset{x,\bar{y}}{\forall}\,(\text{prepend}[x,\langle\bar{y}\rangle] = \langle x,\bar{y}\rangle)$.

(Axiom 4) $\underset{x,\bar{y}}{\forall}\,(\text{sort}[\langle x,\bar{y}\rangle] = \text{insert}[x,\text{sort}[\langle\bar{y}\rangle]])$,

(Axiom 5) $\underset{n,m,\bar{x}}{\forall}\,(\text{insert}[n,\langle m,\bar{x}\rangle] = \text{prepend}[\max[m,n],\text{insert}[\min[n,m],\langle\bar{x}\rangle]])$,

To prove (Proposition (goal)), we have to find $\bar{x}^*$ such that

(1) $\text{sort}[\langle 1,3,2,4\rangle] = \langle\bar{x}^*\rangle$.

We choose

$$\bar{x}^* = \text{Sequence}[4,3,2,1]$$

and show that the equality (1) holds for this value (assuming that the built-in simplification/decomposition is sound):

(Theorem) $\text{sort}[\langle 1,3,2,4\rangle] = \langle 4,3,2,1\rangle$.

Proof.

$$\text{sort}[\langle 1,3,2,4\rangle] = \langle 4,3,2,1\rangle$$

if and only if (by (Axiom 4) LR)

$$\text{insert}[1,\text{insert}[3,\text{insert}[2,\text{insert}[4,\text{sort}[\langle\rangle]]]]] = \langle 4,3,2,1\rangle$$

if and only if (by (Axiom 1) LR, (Axiom 2) LR)

$$\text{insert}[1,\text{insert}[3,\text{insert}[2,\langle 4\rangle]]] = \langle 4,3,2,1\rangle$$

if and only if (by (Axiom 5) LR)

$$\text{insert}[1,\text{insert}[3,\text{prepend}[\max[4,2],\text{insert}[\min[2,4],\langle\rangle]]]] = \langle 4,3,2,1\rangle$$

if and only if (by (Axiom 2) LR, (Axiom 3) LR)

$$\text{insert}[1,\text{insert}[3,\langle\max[4,2],\min[2,4]\rangle]] = \langle 4,3,2,1\rangle$$

if and only if (by (Axiom 5) LR, (Axiom 5) LR, (Axiom 2) LR, (Axiom 3) LR, (Axiom 3) LR, (Axiom 5) LR, (Axiom 5) LR, (Axiom 5) LR, (Axiom 2) LR, (Axiom 3) LR, (Axiom 3) LR, (Axiom 3) LR)

$$\langle\max[\max[\max[4,2],3],1],\max[\max[\min[2,4],\min[3,\max[4,2]]],$$

$$\min[1,\max[\max[4,2],3]]],\max[\min[\min[3,\max[4,2]],\min[2,4]],$$

$$\mathrm{min}[\mathrm{min}[1,\mathrm{max}[\mathrm{max}[4,2],3]],\mathrm{max}[\mathrm{min}[2,4],\mathrm{min}[3,\mathrm{max}[4,2]]]]],\mathrm{min}[\mathrm{min}[\mathrm{min}[1,$$
$$\mathrm{max}[\mathrm{max}[4,2],3]],\mathrm{max}[\mathrm{min}[2,4],\mathrm{min}[3,\mathrm{max}[4,2]]]],\mathrm{min}[\mathrm{min}[3,\mathrm{max}[4,2]],\mathrm{min}[2,4]]]\rangle$$
$$= \langle 4,3,2,1\rangle$$

if and only if (by the built-in simplification/decomposition)

$$\langle 4,3,2,1\rangle = \langle 4,3,2,1\rangle$$

which, by reflexivity of equality, concludes the proof. □

- The proof has an existential goal, thus, it starts with a "Solve" phase, where a solve constant $\bar{x}^*$ is introduced for the sequence variable $\bar{x}$, see (1).
- The solve constant is instantiated by unification. Note, that in this case sequence unification is applied. The solve problem is reduced to an equational proof problem, see (Theorem).
- The prover enters a "Compute" phase, in which it tries to prove the equality by rewriting the goal equality using equalities from the knowledge base.
- After having applied all possible rewrite steps, the remaining equality is simplified using built-in semantic knowledge. In the concrete case, the prover has been given access to available *Mathematica* functions `Min` and `Max` in order to simplify terms containing `min` and `max`. This is achieved through a **Built-in** environment

  Built-in["Mathematica MinMax",
  min → Min
  max → Max]

  in the prover's knowledge base, which tells the prover to interpret `min` as *Mathematica*'s `Min` and `max` as *Mathematica*'s `Max`. By this, $\mathrm{max}[\mathrm{max}[\mathrm{max}[4,2],3],1]$ simplifies to 4 etc.
- The goal is reduced to a simple equality with identical left hand side and right hand side, thus the proof is finished.

For details on the equational prover, we refer to [169], [170], [168], [171], and [167].

Logicographic Symbols

We present the idea of logicographic symbols by using the theory showing the correctness of merge-sort.

Algorithm["stmg", any[X],

$$\mathrm{stmg}[X] := \begin{cases} X & \Leftarrow |X| \leq 1 \\ \mathrm{mg}[\mathrm{stmg}[\mathrm{lsp}[X]],\mathrm{stmg}[\mathrm{rsp}[X]]] & \Leftarrow \text{otherwise} \end{cases} \quad]$$

Algorithm["mg", any[$X, Y, a, b, \bar{x}, \bar{y}$],

$$\mathrm{mg}[\langle\rangle, Y] := Y$$
$$\mathrm{mg}[X, \langle\rangle] := X$$
$$\mathrm{mg}[\langle a,\bar{x}\rangle,\langle b,\bar{y}\rangle] := \begin{cases} a \smile \mathrm{mg}[\langle\bar{x}\rangle,\langle b,\bar{y}\rangle] \Leftarrow a \geq b \\ b \smile \mathrm{mg}[\langle a,\bar{x}\rangle,\langle\bar{y}\rangle] \Leftarrow \text{otherwise} \end{cases} \quad]$$

Definition["istv", any[X, Y],
$\mathrm{istv}[X,Y] \Leftrightarrow \mathrm{ist}[X] \wedge \mathrm{ipm}[X,Y]$]

Lemma["mg", any[A, B],

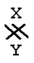
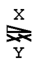
	lsp[X]	left split of X
	rsp[X]	right split of X
	mg[X,Y]	the result of merging two tuples X and Y
	stmg[X]	the result of sorting X by merging
	ipm[X,Y]	X is a permuted version of Y
	ist[X]	X is a sorted tuple
	istv[X,Y]	X is a sorted version of Y

Figure 10: Logicographic Symbols for the Merge-Sort Theory

ist$[A] \wedge$ ist$[B] \Rightarrow$ ist[mg$[A, B]$]]

Lemma[“mg2”, any$[A, B]$,
 ipm[mg$[A, B], A \asymp B$]]

In the THEOREMA notation, ‘$\langle\rangle$’, ‘$\langle x, \bar{X}\rangle$’, ‘$x \smile X$’, ‘$X \asymp Y$’ stand for ‘empty tuple’, ‘a tuple with the first element x and a finite sequence $\bar{X}$ of elements’, ‘tuple X with element x prepended’, ‘the concatenation of tuple X and tuple Y’ respectively. With the additional explanation in Figure 10 (for the moment ignore the leftmost column), the meaning of the above formal text should be self-explanatory. For example, the definition of ‘stmg’ describes the algorithm of merge-sort: If the length of the argument tuple ‘X’ is less than or equal to 1, then the result is ‘X’. Otherwise, ‘X’ splits into ‘lsp$[X]$’ and ‘rsp$[X]$’, then each of these tuples is sorted by a recursive call of ‘stmg’ and, finally, the two sorted parts are merged by ‘mg’.

With the definitions above, the correctness of merge-sort can be formalized as follows:

Proposition[“Correctness of Merge Sort”, any$[A]$,
 istv[stmg$[A], A$]]

This proposition states that for any tuple ‘A’, after application of the algorithm ‘sorted by merging’, the resulting tuple ‘stmg$[A]$’ is a sorted version of ‘A’. It will be possible to prove this theorem automatically by one of the THEOREMA provers.

Of course, one could be happy with the above formal text from a strictly formal point of view. However, it is difficult to grasp the intuition behind the algorithm in the formal way. So we will now demonstrate how, by the introduction of new “logicographic” symbols in two-dimensional notation, the above formulae become easier to understand.

Figure 10 shows a possible choice of logicographic symbols for the merge-sort theory. Of course the user has complete freedom in designing new symbols for the various notions. With these logicographic symbols, the above formal text can now be written in the way shown in Figure 11. The expressions are represented in a nested 2-dimensional syntax with dark gray and light gray coloring for indicating the syntactical structure. (The users can change the coloring by writing an appropriate $Mathematica$ function).

Using Logicographic Symbols

Since logicographic symbols can be evaluated and they are just a different way of writing the corresponding THEOREMA (function or predicate) constants, they can be used in all contexts in which function and predicate constants appear in THEOREMA (e.g. in Definition[. . .], Prove[. . .], etc.) Namely, when formulae containing logicographic symbols are evaluated, this causes the same effect as executing the formula with all logicographic symbols replaced by their internal constants. As we saw in Figure 11, we can compose

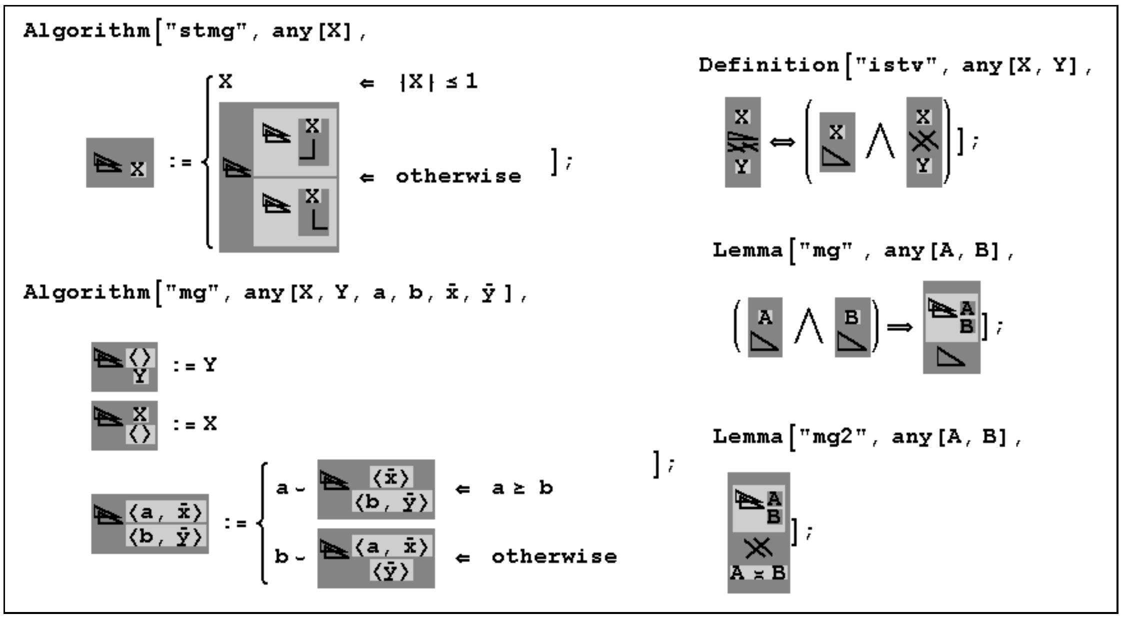


Figure 11: Formalized Merge-Sort Theory with Logicographic Symbols

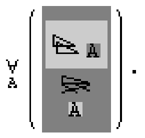
arbitrary knowledge bases using logicographic symbols.

The logicographic presentation can be used for displaying formal proofs. Note that the above logicographic symbols may appear with various different argument terms at different places within the text. One could control which expressions should be displayed with logicographic symbols by specifying the option `Notation` in the option `ShowOptions` of the THEOREMA `Prove` command. For example,

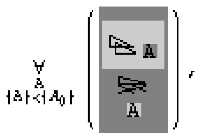
`Prove`[Proposition[“Correctness of Merge Sort”],
  using → Theory[“Merge Sort”], by → CourseOfValueProver,
  ShowOptions → {Notation → LogicographicNotation[“Merge Sort”]}]

The following proof sketch shows the correctness of merge-sort which demonstrates the positive effect of logicographic symbols on making proofs easier to understand.

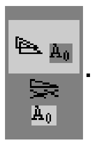
Prove:



We use course of value induction on A. Let now A_0 be arbitrary but fixed and assume

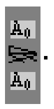
(ind-hyp) 

and show

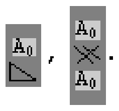
(G) 

We prove (G) by case distinction using (Algorithm: stmg).

Case $|A_0| \leq 1$: We have to prove

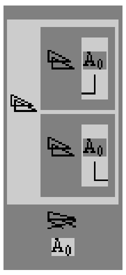


By (Definition: istv), we have to prove



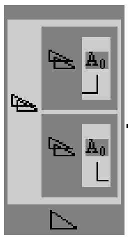
These are true, because (properties of sorted tuples) and (reflexivity of perm).

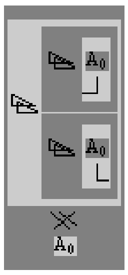
Case $|A_0| \not\leq 1$: We have to prove



.

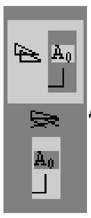
By (Definition: istv), we have to prove

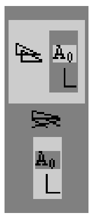
(G1) 

, (G2) 

.

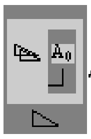
By (ind-hyp), we know

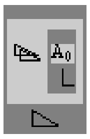
(K) 

, 

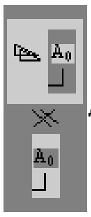
.

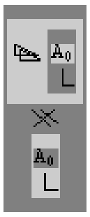
And hence, by (Definition: istv) we also know

(K1) 

, 

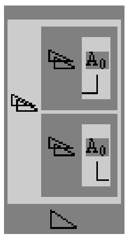
,

(K2) 

, 

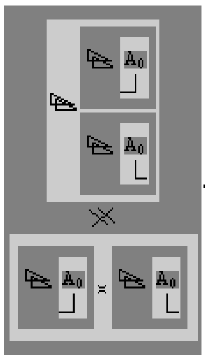
.

We prove (G1): By (Lemma: mg) and (K1),

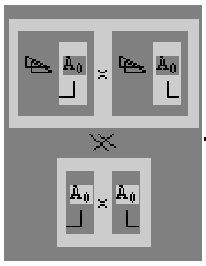


.

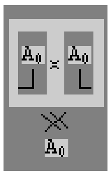
We prove (G2): We know (C1) by (Lemma:mg2), (C2) by (properties of permutation), (C3) by (properties of splitting),

(C1) 

,

(C2) 

,

(C3) 

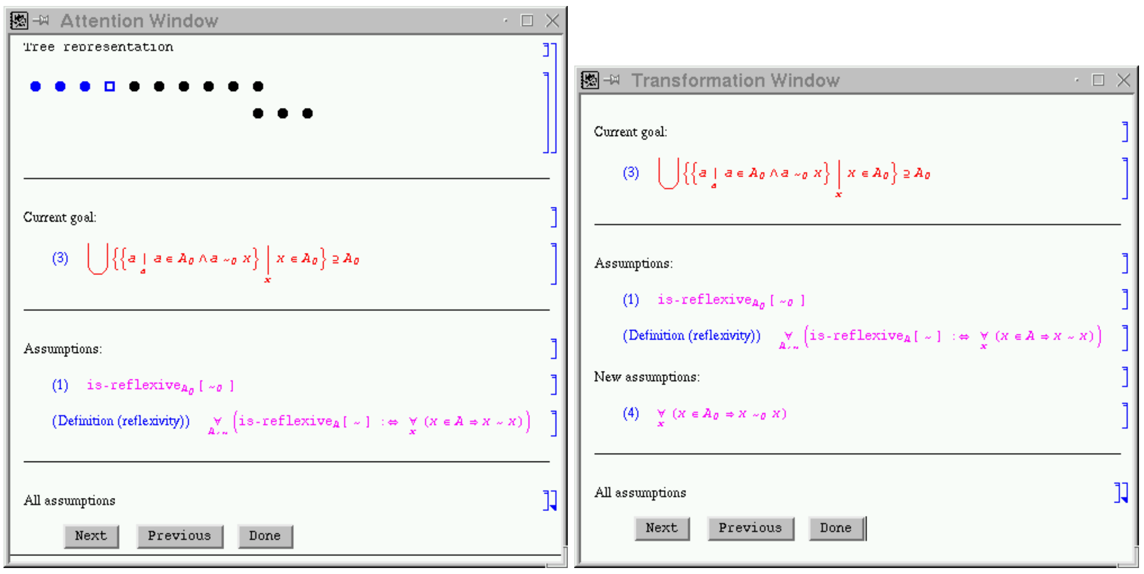
.

Hence, by (C1), (C2), (C3) and (transitivity of permutation), (G2) is proved.

For details on logicographic symbols, we refer to [204], [205], [207], [206], and [65].

Focus Windows

The "focus windows presentation" a new technique for presenting proofs, in particular proofs generated by automated theorem proving systems like THEOREMA. We call this technique "focus windows" technique because with this technique, in each proof step, all the relevant formulae are collected in one window (the "focus window") so that the reader can focus on them. The sequence of focus windows alternates between



Current goal:

(3) $\bigcup\left\{\left\{\underset{a}{a \mid} a \in A_0 \wedge a \sim_0 x\right\} \underset{x}{\Big|} x \in A_0\right\} \supseteq A_0$

Assumptions:

(1) $\text{is-reflexive}_{A_0}[\sim_0]$

(Definition (reflexivity)) $\underset{A,\sim}{\forall}\left(\text{is-reflexive}_{A}[\sim] :\Leftrightarrow \underset{x}{\forall}(x \in A \Rightarrow x \sim x)\right)$

All assumptions

Current goal:

(3) $\bigcup\left\{\left\{\underset{a}{a \mid} a \in A_0 \wedge a \sim_0 x\right\} \underset{x}{\Big|} x \in A_0\right\} \supseteq A_0$

Assumptions:

(1) $\text{is-reflexive}_{A_0}[\sim_0]$

(Definition (reflexivity)) $\underset{A,\sim}{\forall}\left(\text{is-reflexive}_{A}[\sim] :\Leftrightarrow \underset{x}{\forall}(x \in A \Rightarrow x \sim x)\right)$

New assumptions:

(4) $\underset{x}{\forall}(x \in A_0 \Rightarrow x \sim_0 x)$

All assumptions

Figure 12: Attention window and transformation window (part 1)

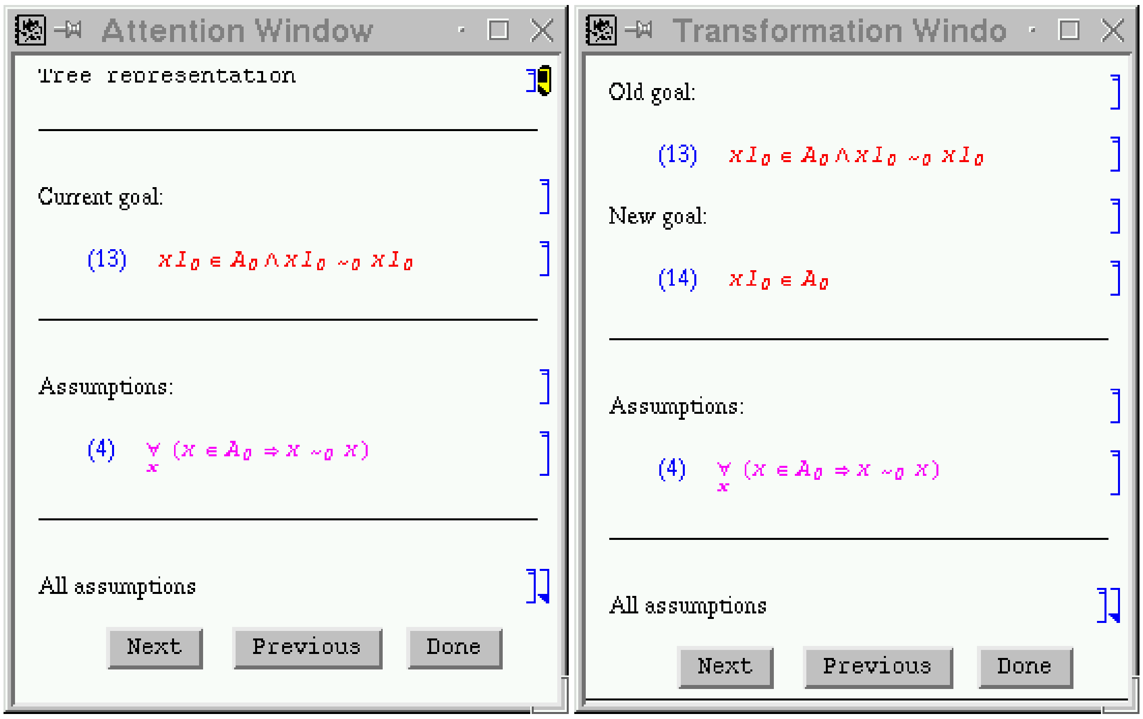
“attention windows” and “transformation windows”. In an attention window, exactly those formulae are displayed – and highlighted – that are relevant for the next proof step. In the subsequent transformation window, in addition to the highlighted formulae, the formulae are displayed that are added as a goal or additional knowledge. Also, a standard natural language text is presented that briefly characterizes the proof technique used.

Focus windows presentations are intended for *interactive* presentation of proofs on a computer screen. Their benefit is based on facilities available in today’s windowing systems such as “multiple windows” and “mouse clicking”. Therefore, it is difficult to demonstrate the advantages of this type of proof presentation on paper. We show parts of the proof of Lemma[“factor set covers”] from Section 2.2.b using the focus windows presentation. Note that the proof object used is the same as for linear proof presentation, meaning the formulae and their labels are the same as in Section 2.2.b.

The presentation starts with a transformation window showing the proof goal and the complete knowledge base. After clicking the “Next” button in the navigation panel on the bottom of the window, an attention window shows only the proof goal, because the proof will proceed by reducing the goal. Clicking “Next” several times presents the attention window shown in Figure 12. The top of the window shows a tree representation of the proof with a “□” indicating the current position[9]. Then the current goal (3) is displayed, followed by two assumptions (1) and (Definition (reflexivity)) from the current knowledge base, i.e. attention for the next proof step has to be paid just to these formulae. Clicking “Next” brings up the transformation window shown in Figure 12, which contains the same items as the preceding attention window plus the new assumption (4) obtained by rewriting (3) using (Definition (reflexivity)).

In the final phase of the proof, we need to reduce the goal using an implication from the knowledge base. The attention and transformation window for this step are shown in Figure 13. In the attention window only the goal (13) and the known implication (4) are shown, clicking “Next” brings up the reduced goal (14) in the transformation window. In particular in longer proofs, this type of presentation is advantageous, because it always keeps the focus on just those parts of the proof that are relevant for just the current or the next step. For details on the focus window presentation technique see [220], [218], and [221].

[9]The cell brackets at the right margin are a feature offered by the *Mathematica* notebook front end. Clicking cell brackets opens/closes cells so that their contents is displayed/hidden. By default, the cell showing the tree representation is closed.



Tree representation

Current goal:

(13) $xI_0 \in A_0 \wedge xI_0 \sim_0 xI_0$

Assumptions:

(4) $\underset{x}{\forall} (x \in A_0 \Rightarrow x \sim_0 x)$

All assumptions

Old goal:

(13) $xI_0 \in A_0 \wedge xI_0 \sim_0 xI_0$

New goal:

(14) $xI_0 \in A_0$

Assumptions:

(4) $\underset{x}{\forall} (x \in A_0 \Rightarrow x \sim_0 x)$

All assumptions

Figure 13: Attention window and transformation window (part 2)

Theorema: An Introduction to the System

Integrating Proving and Computing

The integration of proving and computing in one frame is a challenge for the logic and the language of a system. Pure computation systems, such as *Mathematica* MAPLE or others, contain a huge database of mathematical knowledge, which is applied *implicitly* in each computation. In a proving system, on the other hand, the user needs *explicit* control over the knowledge base that is used to obtain a proof. A system that combines proving and computing needs to resolve this conflict between implicit and explicit knowledge, see also [61], [62] for more background on this issue.

Thus, in the THEOREMA system, we decided to design the language in such a way, that there is a clear separation between the implementation of syntax and semantics of the language. The syntax of expressions is pre-defined by the THEOREMA expression language but there is no hidden implicit semantics of THEOREMA expressions. Of course, we offer the usual semantics, even in the form of algorithms, wherever possible but it must always be applied explicitly!

We try to illustrate the critical issue in the system design again in one example: Consider the expression $2 + 7$. For serving as a *computation system*, the system must have knowledge about the symbols involved in the expression, i.e. "2", "+", and "7". Since THEOREMA is built on top of Mathematica, it would be convenient to use the knowledge built into Mathematica, that is "2" and "7" are natural numbers and "+" is the well-known addition. On the other hand, we want to *prove* formulae such as $\underset{x,y}{\forall}\, x + y = y + x$ (commutativity of "+"), where, of course, we do not want to use the built-in interpretation of "+", because the built-in addition is already assumed to be commutative. The same kind of conflict arises with all semantics that we implement for the THEOREMA language. Thus, we decided to *hide all semantics* on the top level and instead force the user to enter `Compute`[2+7, built-in $\rightarrow$ Built-in["Operators"][+]] in order to compute "2+7" using the built-in operator "+". A detailed explanation of this issue is given in [80] and

[74].

Built-in["Operators"] is a THEOREMA pre-defined collection of translations of frequently used operator symbols ('+', '$*$', '$\leq$', etc.) to available *Mathematica* operations ('Plus', 'Times', 'LessEqual', etc.). Various pre-defined Built-in[$\ldots$] collections are available, moreover, there is the possibility for user-defined translations, as it has already been demonstrated in the example of the equational prover in Section 2.2.b, when translating 'min' and 'max' to the *Mathematica* functions 'Min' and 'Max'. Through this mechanism, the entire algorithm library of *Mathematica* is available for doing computations in THEOREMA.

Prove Strategies

All the provers of the THEOREMA system work on *proof situations*. A proof situation $\langle\{\mathrm{assumptions}\}, \mathrm{goal}\rangle$ consists of a collection of assumptions and a goal formula. In one proof step a prover carries out a proof deduction step that reduces a given proof situation to hopefully simpler proof situations. For instance, the rule "For proving $A \wedge B$ we prove A and we prove B" is represented as a rewrite rule which transforms the proof situation $\langle\{\mathrm{assumptions}\}, A \wedge B\rangle$ into the two proof situations $\langle\{\mathrm{assumptions}\}, A\rangle$ and $\langle\{\mathrm{assumptions}\}, B\rangle$. Most of the rules will, in fact, produce only one proof situation, however some of the rules produce several proof situations connected by *AND* (as above), and some rules will produce proof alternatives connected by *OR*. Some rules are terminal – like "the goal is among the assumptions" – and produce final proof steps.

The proof is represented internally as an *AND-OR* tree, called *proof-object*, whose nodes contain annotations documenting the proof steps. These annotations are used later to produce a human readable proof. Final proof steps are the leaves of the tree. The management of the proof-object is done by a control mechanism which also allows to combine several provers – e.g. simplification and induction –, for the details we refer to [251], [252], [250], and [258].

As a meta-strategy, THEOREMA provides the *cascade*. Intuitively, the idea is that, given a goal G and a knowledge base K, we let a given prover P try to find a proof. If P succeeds, we stop and present the proof. If not, we let a "failure analyzer" analyze the proof attempt and conjecture a lemma L, which could be strong enough to allow P to prove G from $K \bigcup L$. Now we let P try to prove L from K. If this succeeds we let P try, again, to prove G but this time under the assumption $K \bigcup L$. Otherwise we let the failure analyzer work on the failing proof. In other words, given a prover P and a "conjecture from failure generator" C, the recursive "cascade" may result in a much stronger prover that, in fact, does not only prove more theorems than P but, on the way of proving a goal from a knowledge base, gradually extends the knowledge base by "useful" lemmas. For details see [59], [8], and [72].

Proof Presentation and Proof Simplification

One of the goals of the THEOREMA system is to produce proofs in a natural style, i.e. in a style that would typically be used by a human mathematician. In Figures 7 to 9 we already showed an example of a proof generated by one of the THEOREMA provers. Many more examples can be found in the various publications on THEOREMA see for example [74], [70], [76], or [72], see also `http://www.theorema.org`.

For achieving the goal of producing natural-style proofs, the generation of a proof is split into two phases:

- the generation of an *abstract proof object* and
- the generation of the *written presentation of the proof.*

In the first phase, as already described in Section 2.2.b, a proof is generated by application of inference rules. The process of searching for a successful proof is stored in an internal tree structure, the proof object. Each node in the proof object represents one deduction step, with which a textual representation is associated. Once the proof object is generated, the written proof presentation is generated by processing the individual nodes of the proof object. Note that the textual presentation of the proof is *not* part of the proof object. Thus, proofs can be produced in different languages from one and the same proof object. See for instance [250], [8], and [59].

Depending on the purpose, the proof object can be presented in different ways:

- the entire proof object including all failing branches,
- only the succeeding branch,
- only steps that actually contribute to the successful proof,
- or only the information, whether the proof succeeded or not.

This process of *proof simplification* is a natural strategy, also applied by human provers: In a first attempt one tries to find a crude version of a proof, which might still be unnecessarily complicated. Then, in a second step, one works on the proof found and tries to simplify it in various ways.

In [258] the currently available simplification mechanisms are explained. We also started to work on more sophisticated techniques for proof simplification, for example extracting similar proof parts from various parallel branches of the proof. Such proof simplification strategies that work, as post-processors, on the proof objects generated by our provers, are currently implemented in the ongoing PhD thesis [219], for first results see [222].

General and Special Internal Provers

In the first phase of the project, several general and special prove methods have been implemented:

Propositional Logic Prover for proving formulae in propositional logic using inference rules in natural deduction style. This prover is based on a set of rules which are similar to the ones used in sequent calculus, some examples are below:

$\langle\{A, \neg A, \ldots\}, G\rangle \longrightarrow \text{Proved}$

$\langle\{A, A \Rightarrow B, \ldots\}, G\rangle \longrightarrow \langle\{A, B, \ldots\}, G\rangle$

$\langle\{\ldots\}, G_0 \vee G_1 \vee \ldots \vee G_n\rangle \longrightarrow \langle\{\neg\, G_1, \ldots, \neg\, G_n, \ldots\}, G_0\rangle$

$\langle\{A \vee B, \ldots\}, G\rangle \longrightarrow \langle\{A, \ldots\}, G\rangle$ *AND* $\langle\{B, \ldots\}, G\rangle$

$\langle\{A \Rightarrow G, \ldots\}, G\rangle \longrightarrow \langle\{\ldots\}, A\rangle$ *OR* $\langle\{\neg A, \ldots\}, G\rangle$

As a general strategy, the prover will try to move the negation symbol inside the formulae and to split the assumptions and the goal until a final proof situation is found. Some examples of the propositional proofs are given in [74].

Predicate Logic Prover for proving formulae in first order predicate logic using inference rules in natural deduction style. This prover uses the rules of the propositional prover intermixed in a convenient way with new rules for predicate logic. Some example of such rules are listed below:

$\langle\{\ldots\}, \underset{x}{\forall}\, P[x]\rangle \longrightarrow \langle\{\ldots\}, P[x_0]\rangle$

$\langle\{\underset{x}{\exists}\, P[x], \ldots\}, G\rangle \longrightarrow \langle\{P[x_0], \ldots\}, G\rangle$

$\langle\{P[a] \Longrightarrow Q[a], \underset{x}{\forall}\, P[x], \ldots\}, G\rangle \longrightarrow \langle\{Q[a], \underset{x}{\forall}\, P[x], \ldots\}, G\rangle$

Rules as above are "simplifying rules" because they simplify the proof situation. Such rules are used also in the PCS prover (described in the sequel) in the "proving" phase.

However, one also needs rules, which "complicate" the proof situation either by adding new assumptions, either by creating several branches. An example of an applications of such rules is:

$\langle\{\underset{x}{\forall} P[x] \Longrightarrow Q[x], P[a], \ldots\}, G\rangle \longrightarrow \langle\{Q[a], \underset{x}{\forall} P[x] \Longrightarrow Q[x], P[a], \ldots\}, G\rangle$

which corresponds to *forward reasoning*. By *backward reasoning* one replaces the current goal by using an universal assumption, but in order to insure completeness of the prover one usually has to follow several alternatives.

The usage of such rules has to be done following a certain proof search strategy in order to insure completeness and efficiency of the prover. One such strategy is defined by the PCS prover (described later), which uses special universal assumptions (implications, equivalences, equalities) as rewrite rules.

In order to approach proof problems having universal assumptions, we have also implemented a *level saturation* strategy for predicate logic proving, which is applicable both in *forward* mode and in *backward* mode or combined, see [158]. For instance, in forward mode, after all the "simplifying rules" have been exhausted, in one proof step all possible new assumptions are produced based on existing ground literals and universal assumptions, using a general forward inference rule. Then the simplifying rules are applied again, etc. This strategy produces proofs for proof problems that can be solved using ground assumptions.

In dual fashion, the backward mode consists in replacing the goal at each cycle with all possible alternatives that are determined by the universal assumptions. Combining the two strategies efficiently produces proofs for problems having ground goals or simple existential goals.

However, this is not sufficient, for instance, for problems containing formulae with alternating quantifiers. Currently we are implementing a strategy, which uses the analysis of the relationship between the existential goals and the universal assumptions in order to make the appropriate instantiations that allow then the application of simpler rules. Preliminary experiments show that this method can produce proofs for a large class of problems, especially when it is combined with the use of metavariables, see [147], [159], [142], [143].

Induction Provers for various domains, e.g. natural numbers or tuples, proving universally quantified formulae in the respective domain by induction. The induction provers implement the induction scheme for the respective domain depending on the inductive structure of the domain. General induction provers, which infer the inductive structure of the underlying domain from the functor definition (see [266] and [8]) of the domain will be implemented in future versions of the system. Some first experiments have been done in the frame of the PhD Thesis [250], for examples see also [74]. Recently, extensive case studies in the domain of tuples have been carried out by a Calculemus YVR, see [110], [109].

Simplifier Prover for proving equalities by simplification. This prover applies term-simplification on both sides of equalities. Various options can be used to adjust the behavior of the prover with respect to properties of the operators involved (commutative operators, associative operators, etc.).

Case Distinction Prover for proving formulae involving predicates or functions defined by case distinction. This prover is rarely used as stand-alone prover, but is normally used in combination with the other provers, see [258] and [252].

All provers mentioned above are so-called *White-Box-Provers* in the sense that they produce proofs in human readable style that can be checked easily by a human mathematician by applying basic inference rules. However, many powerful proof methods have been developed over the years, which are based on applying powerful algorithms from computer algebra as "black boxes" that, however, can be used only in special theories applying special mathematical knowledge to a transformed prove problem. We put some effort into integrating these sophisticated, known methods into the current version of THEOREMA. By now, the following methods have been implemented:

Gröbner Bases Prover for proving boolean combinations of multivariate polynomial equations using the Gröbner bases method. The class of formulae decidable by this prover includes also the important class of geometrical theorems in cartesian coordinates formulation, see [63] and [64]. In the thesis [230] this prover is also supplied with an interactive graphical input tool. Also, in this thesis, other algebraic methods for geometrical theorem proving, in particular Wu's method and the area method, are re-implemented in the frame of THEOREMA with a perspective to study the interaction with these method with our predicate logic provers, see [231], [229].

Paule-Schorn Prover for proving combinatorial identities based on a method developed by P. Paule and M. Schorn. This method was completely invented and implemented in Mathematica already in [215], and was integrated into the language and logic frame of THEOREMA, see [70], as a model for the potential of THEOREMA to integrate smoothly various algebraic methods developed by other groups in the Calculemus community.

Resolution Prover for proving formulae in first order predicate logic by resolution. Highly sophisticated resolution provers are available from various research groups around the world. Therefore, at first sight, it seems to be appropriate not to devote effort into developing a new resolution prover in the frame of THEOREMA. In fact, we established links to existing resolution provers, see next section. However, we decided to embark also on the implementation of resolution within THEOREMA, see [169], because we want to combine the essential components of resoulution, notably unification and equational paramodulation, with our natural deduction style theorem provers. Fundamental research work in related topics has been done in [166] and [165].

The PCS Prove Strategy

Since, of course, most proofs can not be done by applying just one method, each THEOREMA-prover is a certain combination of the above components. The system provides a mechanism to combine prover components to a new prover in a simple way by just describing, which method to be applied under what circumstances. For instance, the *PCS-provers* combine **P**rove, **C**ompute, and **S**olve phases, which results in very natural proofs for theorems of type "for all ϵ exists δ" as they appear often, for example, in analysis. See [79], [66], and [60] for an overview and [258] for a detailed description of the PCS approach, which is applied, in a similar way, also in [268]. The PCS approach resulted in the automatic generation of natural-style proofs for many theorems of elementary analysis, which so far has still been a challenge for theorem provers.

In the case of the analysis proofs, the "Prove" and the "Compute" phase reduce the proof problem to a solve problem over the real numbers, which then can be solved by calling one of the existing computer algebra methods for constraint solving, e.g. Collins' algorithm. This opens the possibility to establish a strong link between theorem proving and computer algebra.

User Interaction

The feasibility of computer-supported theorem proving depends drastically on the possibility of reasonable user interaction. We believe that the most important type of interaction is by composing the appropriate knowledge base for a particular prove problem. The importance of this possibility is often underestimated. In THEOREMA, we put particular emphasis on convenient knowledge base management, which is possible by our formal text tools, see [69]. More research will be done in the ongoing PhD thesis [259].

However, we also investigated and implemented possibilities for user interaction during the generation of proofs, which is particularly useful also for applications of THEOREMA in didactics, see [162], [161], [8], and [223].

External Provers

The provers described in Section 2.2.b implemented in the frame of THEOREMA in the Mathematica programming language are the so-called *internal provers*. Since THEOREMA wants to serve as an integrative tool for all phases of mathematical work, the system should offer a big variety of proving methods from different fields of mathematics. Due to the fact that some of those proving methods are already available, the system also provides interfaces to existing *external provers* developed outside THEOREMA. The *MathLink* protocol provided by Mathematica to communicate with external programs is used to exchange data with external provers running as external programs. At the moment THEOREMA provides interfaces to OTTER, EQP, Gandalf, BLIKSEM, and a model searcher Mace. Details on interfacing to external provers are given in [172].

Moreover, an interface to the quantifier elimination package *QEPCAD*, originally developed by Hoon Hong, has been implemented, which offers a sophisticated method to solve constraints over the real numbers. Using this interface, THEOREMA provides quantifier elimination as a special "solving-technique" on the top-level (i.e. in a user-request to solve a set of constraints over the reals) as well as it can also be applied as a black box during the solve-phase in the PCS provers, see Section 2.2.b, see [63].

2.2.c Discussion

Mathematical software systems that should give computer-support in the entire spectrum of mathematical activity must combine the strengths of both Computer Algebra Systems (CAS) and Deduction Systems (DS). Most of today's CAS provide huge algorithm libraries for performing heavy computations involving complicated mathematical data. Moreover, most CAS provide facilities for graphical representation of data, a programming language for implementing user extensions, and a comfortable user interface. DS, on the other hand, put their emphasis on formal rigor and on the logical correctness of manipulations they perform.

Different approaches can be taken in order to combine the powers of CAS and DS. In this section, we report on the possibilities to enhance CAS with reasoning power, i.e. to embed DS into CAS. In the frame of the Calculemus project, several levels of enhancement have been explored. Mainly, we distinguish enhancement on the *system level*, enhancement on the *theory level*, and enhancement on the *user level*. For milestone 2.1 and for future development in Task 2.2, we decided to focus on user level enhancement, which aims at an implementation of general and special theorem proving facilities on top of an existing CAS using the programming language provided by the CAS. THEOREMA developed at RISC, is an example of such a system. The THEOREMA system is presented in detail as a prototype of a CAS enhanced with deductive power representing milestone 2.1 of the project.

Task 2.3: Enhancing the Computation Power of Deductions Systems

TASK LEADER: UED
SCIENTISTS IN CHARGE: ALAN BUNDY, SIMON COLTON, EWEN MACLEAN
RESEARCH TEAM: UED, ITC-IRST, UGE, UBIR

2.3.a Overview

This report introduces six pieces of research which make use of enhanced computational power. We describe two systems which combine the proof-planner $\lambda Clam$ [228] with other systems and perform computationally costly tasks. We give an overview of work done in combining systems to discover attacks to security protocols. This work makes use of computational power in that it generates a large number of clauses in its process. We give a brief description of work done in the $\lambda Clam$ proof-planner to construct very large and modular proof-plans for complicated real analysis theorems. Finally we introduce two pieces of work done outside Edinburgh which render techniques from automated reasoning highly efficient by using enhanced computational power.

2.3.b Integrating MathWeb and $\lambda Clam$

The work of Jürgen Zimmer and Louise Dennis incorporates the $\lambda Clam$ proof-planner into the MathWeb system. Here the services offered by $\lambda Clam$ to MathWeb are described, and the services used by $\lambda Clam$ that MathWeb offers. $\lambda Clam$ offers two services to MathWeb:

`planProblem` This service takes an OMDOC document, containing a single conjecture, as an argument. The service starts the $\lambda Clam$ proof planning mechanism on the conjecture. In the current implementation, the service expects the conjecture to be about natural number arithmetic. A proposed extension of the service allows clients to also provide the theory in which the conjecture is defined. Client applications using the `planProblem` service can use optional arguments to determine which proof strategy (compound method) $\lambda Clam$ should use for the planning attempt, and to give a time limit in seconds. In the current implementation, the service simply returns the OPENMATH symbol *true* if $\lambda Clam$ could find a proof plan within the given time limit, and *false* if no proof plan could be found.

`ripple` $\lambda Clam$ offers its rippling mechanism as a separate service to MathWeb. The service is given a single input formatted using the OMDOC standard. The OMDOC must contain a non-empty set of rewrite rules, formalised as lemmas, and a goal sequent $H \vdash \phi$ as a conjecture. The `ripple` service tries to reduce the difference between ϕ and the best suitable hypothesis in H using the rewrite rules.

The `ripple` service also tries to apply fertilisation to reduce the goal ϕ to the trivial goal *true*. As a result, the service `ripple` returns an OMDoc which contains the resulting proof planning goal as a sequent $H \vdash \phi'$.

Experiments have also been done in $\lambda Clam$ using MathWeb services. In particular experiments have been done using MAPLE through the MathWeb interface. The results of the combined system have been compared against the CLAM-Lite system as developed by Toby Walsh. Interestingly one theorem for which the combined system found a proof-plan, could not be planned in Clam-Lite. The theorem is:

$$\Sigma_{i=0}^{n} Fib(i) = Fib(n+2) - 1$$

In order to complete the proof for this theorem, some term rearrangement needs to be done which can be performed easily by MAPLE. Walsh argues that the simplicity of the planning mechanism in Clam-Lite prevents a proof-plan being yielded for this theorem.

The results of the work confirm the main thesis of the research which is that it is a better choice to combine existing deduction systems via protocols instead of re-implementing them on top of a Computer Algebra System.

2.3.c Implementation of the GS System into the $\lambda Clam$ Proof Planning System

Predrag Janičić and Alan Bundy have implemented the GS framework in (Teyjus) LambdaProlog and within the $\lambda Clam$ proof planning system [242][10]. All GS macro inference rules are implemented as methods and all combination/augmentation schemes are implemented as compound methods. These methods add new power to the $\lambda Clam$ system. The GS framework has been reformulated as a backward reasoning system (in the original version, it is a proof by refutation system [83]).

A number of conjectures belonging to combinations of decidable theories can be proved by the GS methods in only seconds. It is believed that further development of Teyjus LambdaProlog will increase the efficiency and robustness of their implementation of GS framework.

A detailed account of this work is given in the paper (this paper will be submitted to a major automated reasoning conference):

Predrag Janičić, Alan Bundy: *Implementation of the* GS *Framework for Using Decision Procedures within the $\lambda Clam$ Proof–Planning System*

There are plans to work on further refinements and improvements of the presented implementation. This includes adding support for new underlying theories, working on improving their efficiency, and also working on automatic and semi-automatic synthesis of decision procedures.

As a part of the work on using decision procedures in theorem proving, and as a powerful extension of the GS framework [83] the problem of automatic and semi-automatic synthesis of decision procedures has been tackled. This work is based on Alan Bundy's programme published in [82].

Decision procedures are often vital in theorem proving [83]. In order to have decision procedures usable in a theorem prover it is often necessary to have them implemented not only efficiently, but also flexibly. The implementation should also be such that can be verified in some formal way. In addition, it is important to have decision procedures for new theories, arising from some specific examples. For all these reasons, it can be fruitful if the process of synthesising and implementing decision procedures can be automated. It would also utilise avoiding human mistakes in implementing decision procedures. All routine steps in

[10] $\lambda Clam$ is a tool for automated theorem proving in higher order theories. In particular, $\lambda Clam$ specialises in proof using induction based on the rippling heuristic. $\lambda Clam$ is a higher-order version of $Clam$. As $Clam$, $\lambda Clam$ also uses proof planning to guide the search for a proof.

these tasks should be automated. Since most of the steps in different decision procedures can be described via rewriting, object level proofs can be relatively easily derived from the sequence of methods applied.

As discussed in [82], most steps of many decision procedures can be described via sets of rewrite rules. Specific subsets of rewrite rules are organised into methods, while methods are organized into compound methods (or decision procedures themselves). Due to its importance in software and hardware verification, the work focuses on linear arithmetic. As it is interesting and non-trivial, the Fourier/Motzkin procedure is used as an illustrating example.

The following method generators have been implemented: generators for remove methods, stratify methods, thin methods, left-assoc (for more details see [82]) and absorb methods. These generators can take a given BACKUS-NAUR form (BNF), transform it into another one, and build a method which uses some available rewrite rules such that each input formula (which belongs to the first BNF) will be transformed into a formula which belongs to the second BNF. On the set of all these generators, a (heuristically guided) search for a sequence of methods can be performed, which goes from the starting BNF to a trivial BNF (consisting of only $\top$ and $\perp$), hence, giving a decision procedure.

Their first target theory was ground arithmetic. All the necessary rewrite rules were available. Their system synthesised the decision procedures for this theory completely automatically in around 10 seconds of CPU time. While a decision procedure for ground arithmetic can be also obtained by exhaustive application of all rewrite rules, their system gives a procedure which uses them only in steps and giving a structured proofs (easily understandable to a human).

Their second target theory was (quantified) linear arithmetic. For this theory three more method generators needed to be implemented: one for adjusting the innermost quantifier, one for isolating a variable, and one for removing a variable (cross-multiply and add step). Their system automatically synthesised a decision procedure for linear arithmetic in around 20 seconds of CPU time.

For most of used conditional rewrite rules, the above procedure can be used in order to prove that their conditions cover all possible cases.

This approach gives decision procedures in some cases, but also a high-level way of understanding syntactical transformations and formula rewriting.

2.3.d Discovering Security Protocol Attacks by Finding Counterexamples to Inductive Conjectures

Graham Steel has implemented a system which can be used to find counterexamples to universally quantified conjectures in first order logic, and has applied the work to automatically detecting attacks to security protocols.

Cryptographic protocols are used in distributed systems to allow agents to communicate securely. Assumed to be in the system is a spy, who can see all the traffic in the network and may send malicious messages in order to try and impersonate users and gain access to secrets.

The method chosen in this work to tackle this problem is *proof by consistency* which is a technique for automating inductive proof. Recent versions have been proven to detect non-theorems in a finite amount of time- in other words they are *refutation complete*. Comon and Nieuwenhuis have proved that all previous techniques for proof by consistency can be generalised to a new form which they call an *I-Axiomatisation*, as defined in definition 1, which provides an easy separation between the inductive completion and inconsistency detection. The crucial result of the theory is given by theorem 1.

Definition 1 *A set of first-order formulae $\mathcal{A}$ is an* I-Axiomatisation *of I if*

1. *$\mathcal{A}$ is a set of purely universally quantified formulae*

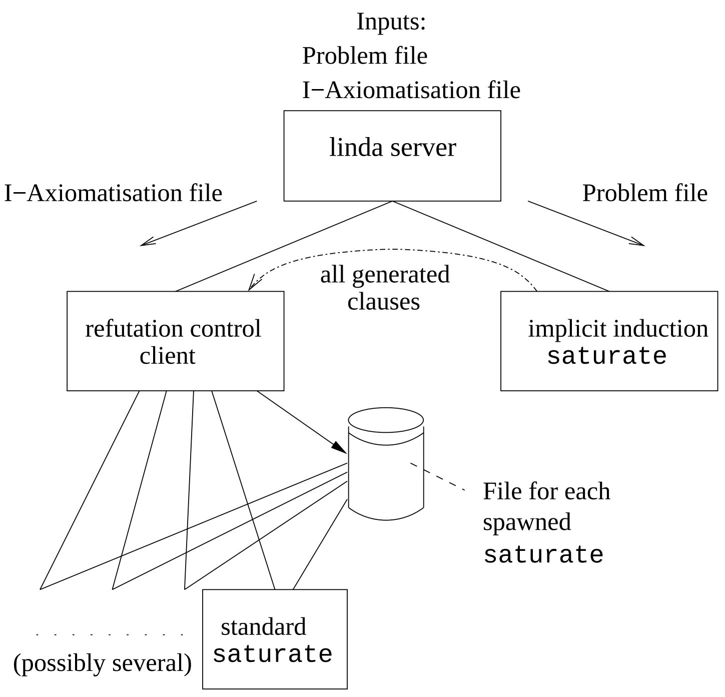


Figure 14: System operation

2. *I is the only Herbrand model of $E \cup \mathcal{A}$ up to isomorphism.*

An I-Axiomatisation is normal *if $\mathcal{A} \models s \neq t$ for all pairs of distinct normal terms s and t*

Theorem 1 *Let $\mathcal{A}$ be a normal I-Axiomatisation, and $C_0, C_1, \ldots$ be a fair induction derivation. Then $I \models C_0$ iff $\mathcal{A} \cup \{c\}$ is consistent for all clauses c in $\bigcup_i C_i$.*

Figure 14 illustrates the operation of the system. The input is an inductive problem in `Saturate` format and a normal I-Axiomatisation (see Definition 1, above). The version of `Saturate` customised by Nieuwenhuis for implicit induction (the right hand box in the diagram) gets the problem file only, and proceeds to pursue inductive completion, i.e. to derive a fair induction derivation. Every non-redundant clause generated is passed via the server to the refutation control program (the leftmost box). For every new clause received, this program generates a problem file containing the I-Axiomatisation and the new clause, and spawns a standard version of `Saturate` to check the consistency of the file. Crucially, these spawned `Saturate`s are not given the original axioms – only the I-Axioms are required, by Theorem 1. This means that almost all of the search for an inconsistency is done by the prover designed for inductive problems and the spawned `Saturate`s are just used to check for inconsistencies between the new clauses and the I-Axiomatisation. This should lead to a false conjecture being refuted after fewer inference steps have been attempted than if the conjecture had been given to a standard first-order prover together with all the axioms and I-Axioms

The system has been applied to cryptographic protocol problems using a first-order version of the higher order formalism used by Paulson in Isabelle/HOL. Preliminary results include rediscovering several known attacks. The problem that Paulson encountered when proving properties of the protocol interactively was that it was very hard to tell, even for an expert, whether the reason for a proof not succeeding was a false conjecture. By applying the counterexample finder developed in the work presented here, attacks can be automatically detected and presented [244, 245].

2.3.e Proof Planning Non-Standard Analysis

Ewen Maclean has been using $\lambda Clam$ to automatically construct proof-plans for real analysis proofs using non-standard analysis [131, 179]. The latest work incorporates induction in constructing proof-plans for such theorems as the Intermediate Value Theorem and Rolle's Theorem [180]. This technique is illustrated here by means of presenting an outling of a proof-plan for an example theorem; namely the Intermediate Value Theorem.

Intermediate Value Theorem in non-standard analysis is expressed as follows:

$$\begin{gathered}
f : \mathbb{R} \to \mathbb{R} \\
a, b, c : \mathbb{R} \\
\forall x, y \in {}^*\mathbb{R}.\ \widehat{a} \le x \le \widehat{b} \ \wedge\ \widehat{a} \le y \le \widehat{b} \ \wedge\ x \approx y \ \to\ {}^*f(x) \approx {}^*f(y) \\
a < b \\
f(a) \le c \le f(b) \\
\vdash \exists x \in \mathbb{R}.\ a \le x \le b \to f(x) = c
\end{gathered} \tag{1}$$

In order to establish a witness for the existential in the conclusion, the notion of a partition is introduced. A sequence $\{[a_i, b_i]\}$ of partitions is generated, starting from the initial interval $[a, b]$, which are guaranteed to contain the point x. A recursive function is defined to construct this sequence, characterised by the illustration in figure 15. Now conjectures can be stated representing properties about this recursive function

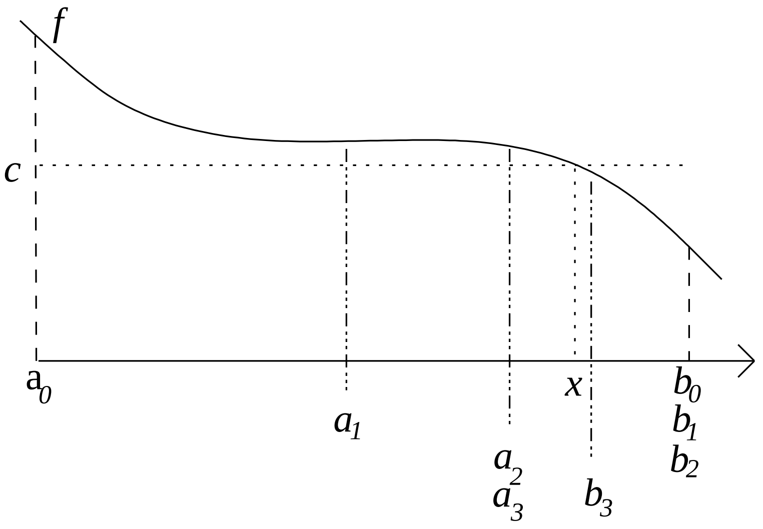


Figure 15: The sequence of partitions

which ascertain that the point x lies inside any interval in the sequence. These lemmas are proved using induction, and the proof-plans for these proofs take advantage of the existing expertise in proof planning for induction; in particular, exploiting the rippling mechanism in $\lambda Clam$. Once the conjectures have been established as theorems they are generalised, and added as hypotheses to the initial theorem. The variables θ are introduced and ϕ, and add the following hypotheses:

$$\begin{gathered}
\theta\ , \phi :\ {}^*\mathbb{R} \\
\theta \approx \phi \\
\widehat{a} \le \phi \le \widehat{b} \\
\widehat{a} \le \theta \le \widehat{b} \\
{}^*f(\phi) \ge \widehat{c} \ge {}^*f(\phi)
\end{gathered}$$

From this it can be determined using non-standard analysis that there exists just one real in the infinitesimal neighbourhood of θ, say x, and that there is just one real in the infinitesimal neighbourhood of ${}^*f(\theta)$, namely c. Then the result ${}^*f(\widehat{x}) \approx \widehat{c}$ is deduced, which reduces easily to $f(x) = c$ establishing the validity of the theorem by finding a witness for the existentially quantified real x.

This technique has been extended to other examples such as Rolle's Theorem and the Mean Value Theorem. Also proof-plans can be automatically generated to account for higher order meta-theorems about the technique.

2.3.e Integration of Computer Algebray Systems into ΩMEGA

Integration of computer algebra systems into the proof planner of the ΩMEGA system is work that has already begun before the start of the Calculemus project and is mainly undertaken by Volker Sorge [151, 152, 243]. The main achievements are an integration of certain complex computations from large-scale computer algebra systems that guarantees correctness of the computations by justifying them with machine-checkable logical calculus proofs. These justifications are computed with a small. self-tailored system that gives protocol information on its calculations. More details on this are given in section 1.1.c of this report.

Technically the integration is achieved with the Sapper interface that can generically connect arbitrary computer algebra systems to ΩMEGA. The interface contains bridging functionality to various systems but also takes care of collecting and processing protocol information to construct correctness proofs. During the Calculemus project the interface has been connected to the MathWeb software bus and it has been extended and now integrates several CAS, such as MAPLE, GAP, and MAGMA.

Besides the described integratio technique we have devised additional techniques to employ symbolic computations in proof planning, in particular to enable the use CAS in control rules and for multi-strategy proof planning. An account of these techniques can also be found under task 1.1.

The enhanced proof planner of ΩMEGA has been successfully applied in several case studies; for details see the report on task 3.5 of the project.

2.3.f LearnΩmatic – Enhancing Proof Planning by a Machine Learning Algorithm

Introduction We devised a framework within which a proof planning system [81] can learn frequently occurring patterns of reasoning automatically from a number of typical examples, and then use them in proving new theorems. The availability of such patterns, captured as proof methods in a proof planning system, reduces search and proof length. We implemented this learning framework for the proof planner ΩMEGA [238], and call our system LEARNΩMATIC. The entire process of learning and using new proof methods in LEARNΩMATIC consists of the following steps:

1. The user chooses informative examples and gives them to ΩMEGA to be automatically proved. Traces of these proofs are stored.
2. Proof traces of typical examples are given to the learning mechanism which automatically learns so-called *method outlines*.
3. Method outlines are automatically enriched by adding to them additional information and performing search for information that cannot be reconstructed in order to get fully fleshed proof methods that ΩMEGA can use in proofs of new theorems.

Learning and Using Learnt Methods The methods we aim to learn are complex and are beyond the complexity that can typically be tackled in the field of machine learning. Therefore, we simplify the problem and aim to learn *method outlines*, which are expressed in the following language L, where P is a set of known identifiers of primitive methods used in a method that is being learnt:

- for any $p \in P$, let $p \in L$,
- for any $l_1, l_2 \in L$, let $[l_1, l_2] \in L$,
- for any $l_1, l_2 \in L$, let $[l_1|l_2] \in L$,
- for any $l \in L$, let $l^* \in L$,
- for any $l \in L$ and $n \in \mathbb{N}$, let $l^n \in L$,
- for any $list$ such that all $l_i \in list$ are also $l_i \in L$, let $T(list) \in L$.

"[" and "]" are auxiliary symbols used to separate subexpressions, "," denotes a *sequence*, "|" denotes a *disjunction*, "$*$" denotes a *repetition* of a subexpression any number of times (including 0), n a fixed number of times, and T is a constructor for a branching point ($list$ is a list of branches), i.e., for proofs which are not sequences but branch into a tree. For more information on the choice of this language, the reader is referred to [139].

Here is an example from group theory of a *simplify* method outline which applies the associativity left method several times, and then reduces the theorem by applying appropriate inverse methods and afterwards an identity method: [*assoc-l* *, [*inv-r* | *inv-l*], *id-l*].

Learning Technique Our learning technique considers some typically small number of positive examples which are represented in terms of sequences of identifiers for primitive methods (e.g., *assoc-l, inv-r*), and generalises them so that the learnt pattern is in language L (e.g., *simplify* given above). The pattern is of *smallest size* with respect to a defined heuristic measure of *size*, which essentially counts the number of primitives in an expression. The pattern is also *most specific* (or equivalently, least general) with respect to the definition of specificity *spec*. *spec* is measured in terms of the number of nestings for each part of the generalisation. Again, this is a heuristic measure. We take both, the size (first) and the specificity (second), in account when selecting the appropriate generalisation. If the generalisations considered have the same rating according to the two measures, then we return all of them.

The algorithm is based on the generalisation of the simultaneous compression of well-chosen examples. Here is an abstract description of the learning algorithm, but the detailed steps with examples of how they are applied can be found in [139]:

1. Split every example trace into sublists of all possible lengths.
2. If there is any branching in the examples, then recursively repeat this algorithm on every element of the list of branches.
3. For each sublist in each example find consecutive repetitions, i.e. patterns, and compress them using exponent representation.
4. Find compressed patterns that match in all examples.
5. If there are no matches in the previous step, then generalise the examples by joining them disjunctively.
6. For every match, generalise different exponents to a Kleene star, and the same exponents to a constant.
7. For every matching pattern in all examples, repeat the algorithm on both sides of the pattern.
8. Choose the generalisations with the smallest size and largest specificity.

For instance, the three sequences of method outlines

$$[\mathit{assoc\text{-}l,assoc\text{-}l,inv\text{-}r,id\text{-}l}], [\mathit{assoc\text{-}l,inv\text{-}l,id\text{-}l}], [\mathit{assoc\text{-}l,assoc\text{-}l,assoc\text{-}l,inv\text{-}r,id\text{-}l}]$$

will be generalised to the *simplify* method

$$[\mathit{assoc\text{-}l}^*, [\mathit{inv\text{-}r} \mid \mathit{inv\text{-}l}], \mathit{id\text{-}l}].$$

The learning algorithm is implemented in SML of NJ v.110. Its inputs are the sequences of methods extracted from proofs that were constructed in ΩMEGA. Its output are method outlines which are passed back to ΩMEGA. The algorithm was tested on several examples of proofs and it successfully produced the required method outlines. Properties of our learning algorithm are discussed in [139].

There are some disadvantages to our technique, mostly related to the run time of the algorithm relative to the length of the examples considered for learning. The algorithm can deal with relatively small examples, which we encounter in our application domain, in an optimal way. The complexity of the algorithm is exponential in the worst case. Hence, we use some heuristics for large and badly behaved examples.

Using learnt methods From a learnt outline a learnt method can automatically be generated. The learnt method is applicable if some instantiation of the method outline, i.e., a sequence of methods, is applicable. Since methods are planning operators with pre- and postconditions, these conditions must be checked for the methods of the method outline. The complex structure of methods does not allow the precondition of a subsequent method of the learnt outline to be tested, without the instantiated postconditions of the previous methods. That is, the methods of an outline have to be applied to the current proof situation.

The applicability test performs a depth first search on the learnt outline. Besides the choice points from the operators of the outline language, i.e., disjunctions and number of repetitions for the Kleene operator, there can be more than one goal where a method of the learnt outline can be applied. Additionally, for methods containing parameters, an instantiation has to be chosen. The parameters of a method are instantiated by control rules that guide the proof search. Every control rule that gives an instantiation of parameters for the current method is evaluated and the resulting possibilities for parameters are added to the search space.

The application test is performed as the precondition of the learnt method. The application of a learnt method for which the test was successful will introduce the open nodes and hypotheses generated during the applicability test as postcondition of the learnt method to the current proof.

Dissemination/Availability In order to evaluate our approach, we carried out an empirical study in different problem domains on a number of theorems. A demonstration of LEARNΩMATIC implementation can be found on the following web page: `http://www.cs.bham.ac.uk/~mmk/demos/LearnOmatic/`. Further information, also with links to papers with more comprehensive references can be found on `http://www.cs.bham.ac.uk/~mmk/projects/CALCULEMUS/`.

This work has been carried out in collaboration of the nodes in Birmingham and Saarbrücken. It particularly involved the YVR Martin Pollet. The results where published in [139, 140, 141].

2.3.g The MathSat Procedure

The MathSat solver [20, 23], developed by ITC-IRST, is a state-of-the-art solver based on the MathSat framework, able to reason on boolean combinations of linear arithmetic formulas. The efficiency of MathSat is due both to the tight integration of the boolean and mathematical solving subroutines, and to the layered structure of the mathematical decider, which is organized into levels dealing with subclasses of formulas of increasing complexity.

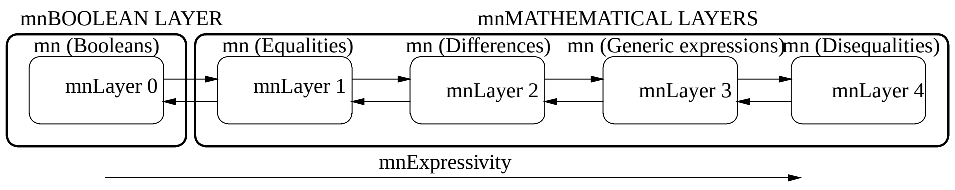


Figure 16: The layered structure of MathSat

The MathSat solver is based on a highly layered structure (see figure 16). The goal of the solver is to decide about the satisfiability of an input formula including both boolean and mathematical atoms. The first layer deals with the boolean component and is implemented as a DPLL procedure, as described in the previous section. The basic idea is that of performing a boolean abstraction of the mathematical atoms (i.e., assigning a new propositional symbol to every different mathematical atom in the original formula), and use the DPLL subroutine for generating (one by one) a complete collection of boolean assignments *possibly* satisfying the input formula. Despite the abstraction, the boolean phase is already capable of cutting a large part of the search space consisting of unsatisfiable assignments.

The role of the mathematical solving component is that of checking whether the assignments produced by the DPLL procedure are consistent or not, by taking into account the actual mathematical atoms which correspond to the artificially introduced propositional symbols. The mathematical component is in turn organized into different layers of increasing complexity. The layered structure allows for early detection of inconsistencies, thus greatly improving the overall performance of the algorithm. The idea is that simpler subclasses of mathematical formulas can be dealt with by specialized, faster satisfiability procedures, whereas more complex satisfiability procedures should be called only if needed.

The current MathSat implementation is able to deal with (a subset of) linear mathematical formulas with equality, disequality and comparison operators. The layers corresponding to the current MathSat implementation are the following:

- the first layer consider equality constraints, performing equality propagation, building equality-driven clusters of variables and detecting equality-driven unsatisfiabilities;
- the second layer handles also inequalities of the kind $v_1 - v_2 \ op \ c$, with *op* being a comparison operator, by using a variant of the Bellman-Ford minimal path algorithm;
- the third layer also consider general inequalities, except for negated equalities, using a standard simplex algorithm.
- the last layer considers also negated equalities.

A number of optimization techniques are added to improve the performance of the MathSat solver. In particular, *early pruning* strategies allows for the early detection of inconsistencies, and *learning* strategies are used for theory-dependent, run-time tuning of the MathSat procedure.

2.3.h RDL, Rewrite and Decision Procedure Laboratory

RDL [6] simplifies clauses in a quantifier-free first-order logic with equality using a tight integration between rewriting and decision procedures. If, on the one hand, the integration of rewriting with decision procedures is considered to be the key ingredient for the success of state-of-the-art verification systems, such as ACL2 [149], STEP [118], Tecton [148], and *Simplify* [116], on the other hand, obtaining a principled and effective integration is notoriously difficult. This is due to the following reasons:

- there are no formal accounts of the incorporation of decision procedures in rewriting. This makes it difficult to reason about basic properties such as soundness and termination of the implementation of the proposed schema.
- Secondly, most integration schemas are targeted to a given decision procedure and they do not allow to easily plug new decision procedures in the rewriting activity.
- Thirdly, only a tiny portion of the proof obligations arising in many practical verification efforts falls exactly into the theory decided by the available decision procedure.

RDL solves the problems above as follows:

1. RDL is based on CCR (Constraint Contextual Rewriting) [11, 12], a formally specified integration schema between (ordered) conditional rewriting and a satisfiability decision procedure [210]. RDL inherits the properties of soundness [11] and termination [12] of CCR. It is also fully automatic.
2. RDL is an open system which can be modularly extended with new decision procedures provided these offer certain interface functionalities (see [12] for details).

   In its current version, RDL offers *'plug-and-play' decision procedures* for the theories of Universal Presburger Arithmetics over the Integers (UPAI), Universal Theory of Equality (UTE), and UPAI extended with uninterpreted function symbols [237].
3. RDL implements instances of a *generic extension schema* for decision procedures [13]. The key ingredient of such a schema is a *lemma speculation mechanism* which 'reduces' the validity problem of a given theory to the validity problem of one of its sub-theories for which a decision procedure is available. The proposed mechanism is capable of generating lemmas which are entailed by the union of the theory decided by the available decision procedure and the facts stored in the current context. Three instances of the extension schema lifting a decision procedure for UPAI are available. First, *augmentation* copes with user-defined functions whose properties can be expressed by conditional lemmas. Second, *affinization* is a mechanism for the 'on-the-fly' generation of lemmas to handle a significant class of formulae in the theory of Universal Arithmetic over Integers (UAI). Third, a *combination* of augmentation and affinization puts together the flexibility of the former with the automation of the latter. Finally, RDL can be extended with new lemma speculation mechanisms provided these meet certain requirements (see [13] for details).

Since extensions of quantifier-free first-order logic with equality are useful in practically all verification efforts, RDL can be seen as an open reasoning module which can be integrated in larger verification systems. In fact, most state-of-the-art verification systems feature similar components, e.g. ACL2's simplifier, STEP validity checker, Tecton's integration of contextual rewriting and a decision procedure for UPAI, and *Simplify* developed within the Extended Static Checking project.

RDL is available via the *Constraint Contextual Rewriting Project Home Page* at `http://www.mrg.dist.unige.it/ccr`.

Task 3.1: Automated Support to Writing Mathematical Publications

TASK LEADER: RISC
SCIENTISTS IN CHARGE: BRUNO BUCHBERGER, WOLFGANG WINDSTEIGER, TUDOR JEBELAN
RESEARCH TEAM: USAAR, RISC, UWB

3.1.a Overview

The general goal in the frame of Task 3.1 is to use the developed prototype systems during the process of writing mathematical publications. Typically, a mathematical publication contains the following ingredients:

- natural language text,
- mathematical formulae,
- formal text, i.e. definitions and theorems,
- proofs,
- examples, typically with computations,
- graphics, tables, drawings, sketches, etc.

In the optimal case, a software system for supporting mathematical publications would support all the facets of mathematical publications listed above. Several systems and languages have been used for case studies in this area.

3.1.b MIZAR

MIZAR approach to this task is based on two kinds of software which automate the process of writing formal mathematical papers:

- software used to prepare an article as a formal text whose correctness is computer verified and
- the software for automatic (or semi-automatic) translation into the natural language (particularly English); this includes also the software for translation into XML-based formats.

The cooperation with other Calculemus sites includes development of the Mizar Mathematical Library (MML) and also the above mentioned translation into XML formats. The main partners are: USAAR, UED and Charles University (ChU) in Prague. The Young Visiting Researches involved in the work are: Josef Urban (ChU), Markus Moschner (USAAR) and Hazel Duncan (UED). Quite intensive cooperation with TUE has been realized using other funds. It includes the visits of Freek Wiedijk, Femke van Raamsdonk and Dan Synek.

Mizar Mathematical Library

In order to write a new article one uses the knowledge stored in the MIZAR database. Currently, it contains 33186 theorems, 6448 definitions, and 680 schemes. The database is based on previously submitted articles which are stored in the Mizar Mathematical Library (MML). At the moment, in the library there are 755 articles. The only acceptable way to increase the database is to provide a new article. Moreover, all changes of the database are processed by modifying the articles already stored. Because of the size of the database specific software for searching through its contents is provided (MML Query, `http://megrez.mizar.org/mmlquery/three.html`). Additionally, MIZAR mode for Emacs created by Josef Urban includes tools for searching in the MML.

The contents of the MML is revised quite often. There are two main reasons for that: finding a better way of formalization and steady development of the MIZAR software. The revisions enable writing articles in more concise and transparent way. Taking into account their scope the revisions may be classified as:

- revisions that improve the quality of the MML without changing the database or
- deeper revisions during which not only the revised article is changed, but also other articles depending on it.

On the other hand, we can classify the revisions according to the size of changes done. There are two extreme cases:

- huge systematic revisions (e.g. changing the type `Element of REAL` to the type `real number`, where practically the whole MML is involved (this kind of revisions is usually carried out automatically by specific software) or
- small ad hoc revisions changing and, as a rule, generalizing only one theorem, always done by hand.

Rarely, so-called restructuring of the MML is performed. It concerns moving some of the information from one article to another.

The basic concepts of set theory are quite well developed in the MML. This includes:

- the theory of boolean operations on sets (XBOOLE_0[11], XBOOLE_1)
- infinite operations on sets (ZFMISC_1),
- binary relations (RELAT_1, RELAT_2),
- set families (SETFAM_1),
- enumerable sets (ENUMSET1),
- functions (FUNCT_1, FUNCT_2),
- ordinal and cardinal numbers (ORDINAL1, ORDINAL2, CARD_1, CARD_2).

[11]This is an example of Mizar article identifier. A full list of MIZAR articles can be found at `http://mizar.uwb.edu.pl/JFM/mmlident.html`

Also the beginnings of classical mathematics are already done. This includes:

- real numbers,
- complex numbers,
- trigonometry,
- real functions,
- differential and integral calculus,
- sequences and series.

An example of a recent achievement is the proof of Fundamental Theorem of Algebra by Robert Milewski [199].

In the modern mathematics the most developed fields are:

- general topology,
- lattice theory (that is partially caused by one of the attacks to formalize contemporary mathematics, [126]),
- category theory; two approaches to this theory were investigated: one in which a category is a quintuple
  $$< O, M, dom, cod, comp >$$
  the second approach closer to homological algebra in which a category is presented as a triple
  $$< O, \{\hom(o_1, o_2)\}_{o_1,o_2 \in O}, \{\otimes_{o_1,o_2,o_3}\}_{o_1,o_2,o_3 \in O} >$$
  where $\otimes_{o_1,o_2,o_3} : \hom(o_1, o_2) \times \hom(o_2, o_3) \to \hom(o_1, o_3)$.
- the theory of vector spaces (real spaces as well as vector spaces over arbitrary fields – actually a specific part of the theory of modules over rings); it includes Hahn-Banach theorems,
- group theory, e.g. Frattini subgroup and solvable groups are already defined.

A serious effort has been made in the theory of Random Access Turing Machines (RATM). It includes both RATMs working on specific data types (integers, finite sequences of integers) and generic ones (RATMs over an arbitrary ring).

The MML consists mostly of 'primary' information i.e. articles which apart from main theorems include also lemmas related to various fields of mathematics. It is the reason why the knowledge related to the same topic is dispersed. Recently, some effort has been made to develop 'secondary' information approach and organize the most useful theorems into theories. As yet, the idea has been applied in articles XBOOLE_0, XBOOLE_1 devoted to the basic properties of boolean operations on sets. In the nearest future the same will be done with elementary theory of real numbers. It is the basis of what we call *The Encyclopedia of Mathematics in Mizar* (EMM).

Writing an article

A new article is prepared as a plain text file using any ASCII editor. However, GNU Emacs is preferable since all MIZAR distributions provide a special mode which facilitates the process of writing MIZAR articles. A MIZAR article is written by step-wise refinement approach. It usually starts with a proof plan and then, after getting report on errors, the gaps in the reasoning are filled. Every article consists of two parts: the environment declaration and the proper text. They are processed by separate programs. The first part is processed by ACCOMMODATOR that communicates with the database and prepares local environment for the article. The author declares which resources from the database are needed. The directives that control this process may be divided into three categories: lexical directives which are used to prepare lexical environment of the article - used symbols, library directives related to previously stored articles and requirements directives which enable using built-in information related to particular mathematical objects (sets, numbers). There are also some utilities which assist the preparation of the environment declaration. Some of them are:

- FINDVOC - informs in which vocabulary a particular symbol has been introduced
- CONSTR - shows which constructors are necessary to use a specific resource from the database (e.g. a theorem).

The proper text of an article is processed by a program called VERIFIER. Its most important logical modules are REASONER, CHECKER and SCHEMATIZER.

- REASONER The main role of this module is to check whether the structure of the proof is correct with respect to the formula being proved. We say that a proof 'fits' a formula if it is created according to tactics determined by the structure of the formula. Apart from checking the fitness of formulae REASONER computes also the result of diffused reasoning and generates formulae corresponding to generic words used in Mizar (e.g., `uniqueness`, `existence` (correctness conditions of functor definitions), `symmetry` (a default property of some predicates), `thesis` (the current goal in the proof)). It also generates definitional theorems.
- CHECKER It is a shortcut for 'inference checker'. It checks whether a reference provided in a straightforward justification makes an inference obvious for the system.
- SCHEMATIZER This module enables the use of so-called 'schemes' that are theorems with second-order free variables.

All parts of the system are under steady development. Let us list the work in progress:

- new `requirements` that introduce to the system more elements of computer algebra
- automation of the use of definitional expansions in CHECKER (now it is only done in REASONER)
- new implementation of `attributes` (static reconstruction of arguments of adjectives); it will enable removing some peculiarities of the system (e.g. non-clusterable attributes) and will make the CHECKER stronger.
- new semantics for SCHEMATIZER with the goal to make it more transparent
- introducing the concept of 'structure loci' that will make obvious facts like 'if a topological group is compact then the topological space of it is compact'.

Enhancing an Article

A collection of so-called 'enhancers' plays a special role in the MIZAR system. These are utilities that are used to make articles better in quality and usually more concise. They are used in various ways:

- by the author to enhance his articles,
- after a revision to make use of newly introduced changes to the MML or MIZAR itself, and

- in the referee process.

The most important of them are:

- IRRVOC - finds unnecessary vocabulary names in a `vocabulary` directive
- IRRTHS - reports irrelevant article names in `theorem` and `schemes` directives
- RELPREM - finds unnecessary references in a justification
- RELINFER - finds consecutive proof steps that may be done in one step
- RELITERS - indicates two consecutive steps in iterative equalities that can be done in one step
- TRIVDEMO - reports when a proof can be substituted by a straightforward justification
- CHKLAB - finds labels that are not referred to
- INACC - shows superfluous fragments of the text.

The output of the above utilities is marked as errors in the text of an article and the author corrects these errors by hand. For the sake of revisions an automatic version of the utilities are also maintained.

Publishing an Article

Formalized Mathematics (a computer assisted approach) (FM), ISSN 1426-2630, publishes the contents of MIZAR articles submitted to the MML. All atricles published in FM are in English. The translation of MIZAR texts into English is made automatically. The material published concerns the surface part of Mizar articles. Proofs forming the inner parts are not yet translated, but other elements - theorems, definitions, schemes, and reservations and global sets if necessary - are.

An electronic extension of FM, *Journal of Formalized Mathematics*, is available from URL address `http://mizar.uwb.edu.pl/JFM/`. In contrast to the paper edition, the electronic extension is dynamic and can follow changes in the MIZAR system and in the MML as well as reflect the improvements to the translation process. As a result, there are translations of MIZAR articles updated according to their current state in the MML.

The process of translation is mechanized, but the final result may be improved by author-editor interaction. The goal of the interaction is to optimize the translation patterns for new MIZAR formats from the article being translated.

The process of translation is performed in several steps. First, a MIZAR article is parsed and stored in some abstract form. Next, the analyzing rules are applied to enrich the abstract form by some statistic information. After that, translation patterns for formulae and formats are used. Finally, the filling text, the summary, and the section titles are added.

These steps and the management of translation patterns are realized by the following programs listed in Table 1.

In Table 1, \$1 stands for the name of MIZAR article, \$2 stands for the name of a database file "\$2.frd" with translation patterns, and \$3 stands for the name of a list of MIZAR articles (usually a code for the journal issue) in the file "\$3.lar". The question marks in "\$1.?" and "$*$.?" stand for the section number of the MIZAR article and $*$ indicates the names of MIZAR articles from the list in file \$3.lar. ACCOM(\$1), PREL(\$1), and PREL stand for MIZAR internal format files created by the ACCOMMODATOR or available from MIZAR directories.

The contents of the files are as follows:

Program	param.	input files	output files
accom	$1	$1.miz, PREL	ACCOM($1)
fmparse	$1	$1.miz, ACCOM($1)	$1.fma, $1.nfr
newfmfrm	$1 -l$2	$2.frd, ACCOM($1), PREL($1)	$1.fmn, $1.fmd
addfmfrm	$1 -l$2	$2.frd, $1.fmn	$2.$-$
fmfrm	$1 -l$2	$2.frd, $1.nfr, $1.fmn, ACCOM($1)	$1.fmf, $1.fmz
resvar	$1	$1.miz, ACCOM($1)	$1.ire
fmnotats	$1	PREL($1)	$1.fms
fmanalyz	$1	$1.fma, $1.fmf, $1.ire, ACCOM($1)	$1.?
jformath	$3 [/d|/f]	[formath.set,] $3.lar, ∗.bnt, ∗.fms, ∗.?	$3.tex
latex	$3	$3.tex, ∗.?, formath.cls, fom10.clo, mizarfrm.tex [, ∗.bbl]	$3.dvi, $3.aux

Table 1: Management of translation patterns

$1.fma - an abstract description of the surface part of article $1.miz (reservations, definitions, theorems, schemes, and global sets).

$1.nfr - new formats introduced in article $1.miz.

$1.fmd - generated translation patterns (with identification) of old formats existing in $2.frd which are used in definitions in article $1.miz.

$1.fmn - generated (or re-edited by the editor or the author of an article) translation patterns (with identification) of new formats from article $1.miz (not yet introduced into $2.frd).

$2.frd - database file of translation patterns (with identification).

$1.fmf - translation patterns (without identification) of formats ordered according to $1.frm from ACCOM($1) and $1.nfr.

$1.fmz - format identifications ordered according to $1.frm from ACCOM($1) and $1.nfr.

$1.ire - information for reserved variables if they are used in elements translated.

$1.fms - (signature) list of articles introducing notation (constructors) used in translated elements from article $1.miz.

$1.?, ∗.? - final LaTeX input including a translation of the section nr ? from article $1.miz (or ∗.miz).

$3.lar - list of article names.

$1.bnt - bibliographic notes of the article, the summary, and the section titles.

formath.set - basic information of the publication issue.

formath.cls, fom10.clo, mizarfrm.tex - LaTeX style files.

We plan to change the internal format used by the translator (∗.fma files) to the XML-based format. As a result of cooperation within the Calculemus project variants of MIZAR utilities have been designed to work with the XML version of the database. Also, the translation of the MML into OMDOC style is in progress.

3.1.c THEOREMA

THEOREMA is a prototype software system designed to give computer-support to the working mathematician during all phases of mathematical activity. The aspect of automated generation of mathematical proofs in THEOREMA has already been described in Task 2.2. In this section, we want to describe facilities offered in the THEOREMA system that make the system feasible for writing entire mathematical publications inside the system.

THEOREMA is implemented on top of the well-known computer algebra system *Mathematica*, see [188], in the high-level programming language that is provided by *Mathematica* and it is designed currently as an add-on package to *Mathematica*. This has the implication that *Mathematica* is needed in order to run THEOREMA, but, on the other hand, THEOREMA can share the highly sophisticated user front-end from *Mathematica*. Moreover, it runs on all hardware environments on that *Mathematica* is supported, which ranges from Windows 95/NT over virtually all Unix environments to also Macintosh computers. The *Mathematica* notebook front-end is an almost perfect environment for composing mathematical texts allowing to mix input, output, graphics, and structured text in one type of document. Special mathematical notation is supported in all categories, even in the input to the system. The user front-end is highly configurable and through the *Mathematica* programming language we even have access to manipulate the input parser. This particularly nice feature made it possible to implement the entire THEOREMA mathematical language on top of *Mathematica* by appropriate modifications to the *Mathematica* input parser.

The style of using THEOREMA is, thus, very similar to the way of communicating with *Mathematica*. The *Mathematica* front-end is an interactive environment that accepts user input typically in form of commands written into input cells that can be sent to the *Mathematica* kernel for evaluation. After loading the THEOREMA package during a standard *Mathematica* session, the basic user mode is a “command-process-answer” loop, in which the user enters a command in the *Mathematica* notebook front-end, THEOREMA processes the command and provides the answer for the user within the *Mathematica* notebook front-end again. Depending on the type of command, the THEOREMA system answer can be the result of a computation in a *Mathematica* output cell or a natural language representation of a proof generated by the system in a separate window, i.e. a separate *Mathematica* notebook. Some THEOREMA commands, when they are of an administrative nature, do not produce visible output but they only manipulate the internal system state, on which the following interactions may depend.

The THEOREMA system, basically, consists of a language for mathematics, in which all the commands can be formulated. Furthermore, it contains methods for processing different types of commands given by the user. The entire system is implemented as extensions of the *Mathematica* kernel written in the *Mathematica* programming language, thus being fully portable across all imaginable hardware platforms. Except for above mentioned administrative commands, the basic commands supported by the THEOREMA system are `Prove`, `Compute`, and `Solve`. When formulating statements having some mathematical content, the highly sophisticated language of mathematics developed over the last centuries – the language of predicate logic – can be used both in input and in system output including all two-dimensional features such as superscripts, indices, writing something on top of another or below, using special mathematical characters like the set braces, logical quantifiers, the integral sign, or the summation sign. In addition to this, we developed a small language, the THEOREMA formal text language, for organizing mathematical knowledge into hierarchically structured definitions, propositions, theorems, knowledge bases, etc. that can be passed as arguments in the calls to Prove, Compute, or Solve. A description of the formal text language has already been given in the report on Task 2.2 of this document, all details can be found in [268].

From the *Mathematica* notebook front-end we inherit additional text-processing facilities, which are necessary, because a mathematical publication does not solely consist of mathematical formulae. For this purpose, a notebook can contain text cells, which allow formatting of natural text like standard text-processors, e.g. different fonts, font sizes, and font faces, inlined and displayed mathematical formulae, structuring into sections and subsections, automatic numbering and cross-referencing, hyperlinks, importing pictures or drawings, etc. The *Mathematica* front-end also serves as a WYSIWYG-editor for notebooks that contains several tools supporting document creation (keyboard shortcuts for editing commands and input palettes). The appearance of notebooks can be customized through style sheets, which determine the behavior and the formatting of each cell type.

These features qualify THEOREMA as a powerful system for creating mathematical publications entirely inside the system. “Classical” mathematical documents can be written that are intended mainly for printout, as for instance the thesis [268] or the conference papers [266], [267], and [269]. In the remainder of this section, we will report on two case studies using THEOREMA to develop *interactive lecture notes*. The Calculemus node RISC is very active in the maths curriculum at the University of Linz and offers two mandatory courses for first-year students. Both, "Algorithmic Methods" and "Predicate Logic as a Working Language", provide computer-support for students through interctive course material based on THEOREMA.

Theorema in Algorithmic Methods

The lecture "Algorithmic Methods" focuses on the interplay between *mathematical knowledge* and *mathematical algorithms*, concretely on the transformation of mathematical knowledge into algorithms. The approach chosen is to show that many times mathematical knowledge *is* already an algorithm without any need of further translation. The course material available for students is written entirely in THEOREMA and it is available in electronic form for the students. *Mathematica* and THEOREMA are installed on laptops that can be used by the students during their first year of study.

The emphasis of the course is being put on the *mathematical representation of mathematical objects* suitable for performing computations. Mathematical data-structures, such as polynomials or matrices, are presented in an algorithmic form, such that their definitions can immediately serve as algorithms for domain operations. Using this approach, we do not need to translate mathematical definitions into their representation in some programming language, and mathematical statements can be used as executable algorithms.

In the THEOREMA language, *Functors* are available for this type of algorithmic construction of mathematical domains, see [266]. Roughly speaking, a functor defines, how new operations on objects of a new domain are defined in terms of known operations in known domains. The THEOREMA language semantics provides algorithmic definitions for basic operations on numbers, tuples, and sets. For arithmetic on numbers we access the available operations from *Mathematica*, thus we have all kinds of numbers available that *Mathematica* can deal with. Tuples are represented as *Mathematica* lists, basic tuple operations are implemented based on list operations available in *Mathematica*. For basic operations on finite sets we implemented our own semantics, based again on list operations available in *Mathematica*. Using functors we can then build up new mathematical domains from the basic domains numbers, tuples, and sets.

An Example: Algorithms using Polynomials

As an example, we demonstrate the definition of univariate polynomials over some field K and an algorithm for polynomial interpolation written in THEOREMA. The functor definition of the domain of univariate polynomials over K in THEOREMA is shown in Figure 3.1.c.

The polynomial functor given in Figure 3.1.c defines, for any K, Poly[K] to be *such a* domain P, such that, for any p, q, n, a

- p is an element of P iff p is a tuple and each component of p is in K,
- the 0 in P is the tuple consisting of the 0 in K,
- the 1 in P is the tuple consisting of the 1 in K,
- the x in P is the tuple consisting of the 0 in K and the 1 in K,
- the *degree* in P of p is defined via some case distinction (note the special language construct $\underset{i}{\text{æ}}\, \mathcal{P}_i$ denoting *the* i such that $\mathcal{P}_i$),
- the *canonic* form in P of p is p without trailing zeros,
- the *sum* in P of p and q is the canonic form of the component-wise sum in K of p and q,

Definition[“Polynomial Domain”, any$[K]$,

$\text{Poly}[K] := \text{Functor}\big[P, \text{any}[p, q, n, a],$

$$\underset{P}{\in}[p] \Leftrightarrow \text{is-tuple}[p] \wedge \underset{i=1,\ldots,|p|}{\forall} \underset{K}{\in}[p_i]$$

$$\underset{P}{0} := \langle \underset{K}{0} \rangle$$

$$\underset{P}{1} := \langle \underset{K}{1} \rangle$$

$$\underset{P}{x} := \langle \underset{K}{0}, \underset{K}{1} \rangle$$

$$\underset{P}{\deg}[p] := \begin{cases} 0 & \Leftarrow \underset{j=1,\ldots,|p|}{\forall} p_j = \underset{K}{0} \\ (\underset{i=1,\ldots,|p|}{\text{æ}} p_i \neq \underset{K}{0} \wedge \underset{j=i+1,\ldots,|p|}{\forall} p_j = \underset{K}{0}) - 1 & \Leftarrow \text{otherwise} \end{cases}$$

$$\underset{P}{\text{coef}}[p, n] := \begin{cases} p_{n+1} & \Leftarrow n \geq 0 \wedge n \leq \underset{P}{\deg}[p] \\ \underset{K}{0} & \Leftarrow \text{otherwise} \end{cases}$$

$$]\ ]$$

$$\underset{P}{\text{canonic}}[p] := \langle p_i \,\big|_{i=1,\ldots,\underset{P}{\deg}[p]+1} \rangle$$

$$p \underset{P}{+} q := \underset{P}{\text{canonic}}\big[\langle \underset{P}{\text{coef}}[p, i] \underset{K}{+} \underset{P}{\text{coef}}[q, i] \,\big|_{i=0,\ldots,\text{Maximum}[\underset{P}{\deg}[p], \underset{P}{\deg}[q]]} \rangle\big]$$

$$p \underset{P}{-} q := \underset{P}{\text{canonic}}\big[\langle \underset{P}{\text{coef}}[p, i] \underset{K}{-} \underset{P}{\text{coef}}[q, i] \,\big|_{i=0,\ldots,\text{Maximum}[\underset{P}{\deg}[p], \underset{P}{\deg}[q]]} \rangle\big]$$

$$p \underset{P}{*} q := \langle \sum_{j=0,\ldots,i}^{K} \underset{P}{\text{coef}}[p, j] \underset{K}{*} \underset{P}{\text{coef}}[q, i-j] \,\big|_{i=0,\ldots,\underset{P}{\deg}[p]+\underset{P}{\deg}[q]} \rangle$$

$$a \underset{P}{\cdot} p := \langle a \underset{K}{*} \underset{P}{\text{coef}}[p, i] \,\big|_{i=0,\ldots,\underset{P}{\deg}[p]} \rangle$$

$$p \underset{P}{/} a := \langle \underset{P}{\text{coef}}[p, i] \underset{K}{/} a \,\big|_{i=0,\ldots,\underset{P}{\deg}[p]} \rangle$$

Figure 17: Functor definition for the domain of univariate polynomials

- the *difference* in P of p and q is …,
- the *product* $*$ in P of p and q is …,
- the *product* $\cdot$ in P of a constant a and p is the tuple containing the product in K of a with each component of p, and
- the *division* $/$ in P of p by a constant a is the tuple containing the quotient in K of each component of p with a.

Computations with polynomials can now immediately be performed with the `Compute` command of THEOREMA[12]. Knowledge, which should be available throughout the session, can be put to a global knowledge base using the command `Use`.

In[1]:= $\mathrm{Use}[\langle \mathbf{Built\text{-}in}[\text{"Tuples"}], \mathbf{Built\text{-}in}[\text{"Quantifiers"}], \mathbf{Built\text{-}in}[\text{"Connectives"}], \mathbf{Built\text{-}in}[\text{"Numbers"}], \mathbf{Built\text{-}in}[\text{"NumberDomains"}], \mathbf{Definition}[\text{"PolynomialDomain"}]\rangle]$

In[2]:= $\mathrm{Compute}\left[\langle\frac{4}{3},\frac{2}{5},0,\frac{1}{3}\rangle \underset{\mathrm{Poly[Q]}}{+} \langle 0,\frac{1}{2}\rangle\right]$

Out[2]= $\langle\frac{4}{3},\frac{9}{10},0,\frac{1}{3}\rangle$

In[3]:= $\mathrm{Compute}\left[\langle\frac{4}{3},\frac{2}{5},0,\frac{1}{3}\rangle \underset{\mathrm{Poly[Q]}}{*} \langle 0,\frac{1}{30},\frac{2}{13},\frac{8}{7}\rangle\right]$

Out[3]= $\langle 0,\frac{2}{45},\frac{71}{325},\frac{2164}{1365},\frac{59}{126},\frac{2}{39},\frac{8}{21}\rangle$

In[4]:= $\mathrm{Compute}\left[7 \underset{\mathrm{Poly[Q]}}{\cdot} \langle 0,\frac{1}{30},\frac{2}{13},\frac{8}{7}\rangle\right]$

Out[4]= $\langle 0,\frac{7}{30},\frac{14}{13},8\rangle$

In[5]:= $\mathrm{Compute}\left[\langle 0,\frac{1}{30},\frac{2}{13},\frac{8}{7}\rangle \underset{\mathrm{Poly[Q]}}{/} 7\right]$

Out[5]= $\langle 0,\frac{1}{210},\frac{2}{91},\frac{8}{49}\rangle$

Having given the algorithmic definition of univariate polynomials as one of the fundamental data structure for algorithmic mathematics, we can then use polynomial arithmetic when discussing algorithms for *polynomial interpolation*. The algorithm for Lagrange interpolation, as an example, computes the interpolating polynomial for (tuples) x, a of degree n over some field K as a certain linear combination of Lagrange basis polynomials. What we need for being able to compute the interpolation polynomial is the algorithm

Algorithm["Lagrange Interpolation", any$[x, a, n, K]$,

$$\mathrm{LagrangeInterpolation}[x,a,n,K] := \mathrm{pf}\left[\sum_{j=1,\dots,n}^{\mathrm{Poly}[K]} a_j \underset{Poly[K]}{\cdot} L_j[x,n,K]\right] \quad]$$

and the definition of the Lagrange basis polynomials L_j.

Definition["Lagrange Basis Polynomial", any$[x, n, j, k]$,

$$L_j[x,n,K] := \left(\prod_{\substack{i=1,\dots,n\\ i\neq j}}^{\mathrm{Poly}[K]} \left(\underset{\mathrm{Poly}[K]}{x} \underset{\mathrm{Poly}[K]}{-} \langle x_i\rangle\right)\right) \underset{\mathrm{Poly}[K]}{/} \prod_{\substack{i=1,\dots,n\\ i\neq j}}^{K} \left(x_j \underset{K}{-} x_i\right) \quad]$$

After adding Algorithm["Lagrange Interpolation"] and Definition["Lagrange Basis Polynomial"] to the global knowledge base we can immediately *compute* the interpolation polynomial in concrete examples.

In[6]:= $\mathrm{UseAlso}[\mathrm{Algorithm}[\text{"LagrangeInterpolation''}], \mathrm{Definition}[\text{"LagrangeBasisPolynomial''}]]$

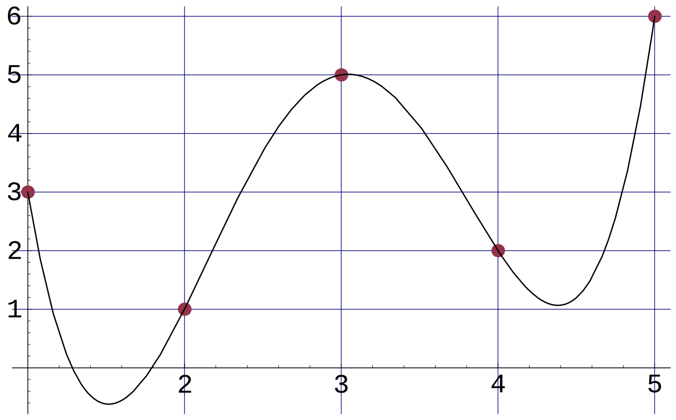
In[7]:= $\mathrm{Compute}[\mathrm{LagrangeInterpolation}[\langle 1,2,3,4,5\rangle, \langle 3,1,5,2,6\rangle, 5, \mathrm{Q}]]$

Out[7]= $\mathrm{pf}\left[\langle 51,\frac{-1093}{12},\frac{443}{8},\frac{-161}{12},\frac{9}{8}\rangle\right]$

[12]In a *Mathematica*/THEOREMA session lines preceded by '`In[i]:=`' denote input to the system, lines preceded by '`Out[i]=`' show the corresponding output.

This means: the polynomial function of degree 5 over $\mathbb{Q}$, which evaluates at 1 to 3, at 2 to 1, at 3 to 5, at 4 to 2, and at 5 to 6, is the polynomial function (pf) associated to the polynomial $\langle 51, \frac{-1093}{12}, \frac{443}{8}, \frac{-161}{12}, \frac{9}{8}\rangle$, i.e. $51 - \frac{1093}{12}x + \frac{443}{8}x^2 - \frac{161}{12}x^3 + \frac{9}{8}x^4$. We now use the *Mathematica* command `Plot` for visualization:

In[8]:= $\text{Plot}\left[51 - \frac{1093}{12}\text{x} + \frac{443}{8}\text{x}^2 - \frac{161}{12}\text{x}^3 + \frac{9}{8}\text{x}^4, \{\text{x}, 1, 5\}, \text{GridLines} \to \text{Automatic}\right];$



The correctness of this interpolation method could now be formulated in the same language as

Theorem[“Correctness of Lagrange Interpolation”, any$[x, a, n, K]$,

$$\underset{i=1,\ldots,n}{\forall} \text{LagrangeInterpolation}[x, a, n, K][x_i] = a_i \quad]$$

and it could be proven automatically by an appropriate THEOREMA prover using the same knowledge in the knowledge base as it was needed for computing[13]! Further lessons in this lecture cover Gaussian elimination for systems of linear equations, Groebner bases for solving systems of polynomial equations, and Newton’s method for arbitrary systems of equations.

Theorema in Predicate Logic as a Working Language

“Predicate Logic as Working Language” is a course which was taught by B. Buchberger for the first time in the summer semester 2002 for undergraduate students of mathematics. The course evolved from [77], the course “Thinking, Speaking, Writing”, and the course “Mathematics for Computer Science: Algorithmic and Nonalgorithmic Aspects”, which B. Buchberger taught various times in the period 1980 - 2002 for computer science and math students at both the undergraduate and graduate level. The course “Predicate Logic as a Working Language” differs from the earlier courses by being completely written in the THEOREMA language. For each of the elementary and more advanced language constructs of predicate logic, first, the syntax in many notational variants, including natural language syntax, is explained. The external two-dimensional THEOREMA syntax is used as a reference standard and the internal THEOREMA syntax (a linear syntax of nested terms) is used as a means for clarifying standard parsing. Then, the inference rules for the language constructs are explained in detail accompanied by lots of examples. In addition, various practical strategies are supplied for inventing proofs by combining inference rules. Also, various ways of structuring and presenting proofs are given. The various rules and strategies are trained by examples and exercises. Also, it is shown how most of these techniques are used in the THEOREMA system for generating proofs in a completely automated way. It is shown how the typical THEOREMA provers imitate the heuristics for finding proofs given to the students during the course.

In addition to introducing inference rules and strategies for proving, inference rules and strategies for solving and simplifying formulae and terms are given. Thus, predicate logic is presented as a uniform frame for proving, solving, and simplifying. During the course, “reasoning” is developed more and more as a network and an interaction of proving, solving, and simplifying steps.

[13]Proving mathematical theorems is, however, not in the scope of the lecture.

Computing is viewed as a special case of simplifying (namely, simplifying ground terms to canonical forms). In this way, it becomes clear that predicate logic contains a universal programming language as a natural sublanguage. Basically, this sublanguage consists of conditional terms and quantifier expressions with bounded ranges. In this part of the language, "sequence variables" play an important practical role. Sequence variables are variables, for which an arbitrary finite number of terms can be substituted. This is in contrast to the ordinary variables of predicate logic for which we can only substitute exactly one term. By having a programming language within predicate logic, algorithmic mathematics can be presented in the same language frame as nonalgorithmic mathematics. Thus, in particular, algorithm verification is just a special case of proving within predicate logic. Also, the correctness proofs of algorithms and the execution of algorithms on concrete data can be done in the same language frame, namely the THEOREMA version of predicate logic.

Particular emphasis is also given to the reasoning techniques for introducing new concepts by definitions of various kinds and for extending the language by special quantifiers which are meaningful only in special theories as, for example, the sum and product quantifier, the set quantifiers, or the limit, derivation, and integration quantifiers. Using definitions, a layered build-up of mathematics is explained and the technique of "theory exploration" (the systematic exploration of the properties of new concepts introduced by definitions) is introduced. It is shown how typical proofs in the exploration of theories that result from a given theory by introducing new concepts by definitions proceed in the three steps of unfolding definitions, working in the given theory, and folding formulae by using definitions backwards.

Finally, it is shown how the general reasoning rules for predicate logic, i.e. the rules valid in the empty theory, can be gradually augmented by special reasoning rules and strategies, i.e. rules and strategies that are valid only in special theories. As examples, reasoning rules for set theory and for inductive theories are provided.

A first version of lecture notes for this course was distributed to the students and the students were also given the opportunity to experiment with the THEOREMA system. However, significant additional work will be necessary for a complete and polished version of the lectures notes. In the math curriculum at the University of Linz, this new course is scheduled for the second semester. In the course evaluation, some students said that they would appreciate to get these fundamental techniques for exact reasoning right at the beginning of their study in a concentrated form. However, it is clear that this would entail radical changes in the structure of the math curriculum. Also, one may argue that it is necessary for the students to get some first acquaintance with the contents and style of university mathematics before they can really appreciate the techniques presented in this practical predicate logic course.

3.1.d The OMDOC Markup Language for Open Mathematical Documents on the Web

Mathematical texts are usually very carefully designed to give them a structure that supports understanding of the complex nature of the objects discussed and the argumentations about them. The OMDOC content markup scheme [157], which has been developed by Michael Kohlhase at USAAR, supports authors with writing formal mathematical documents including articles, textbooks, interactive books, and courses. OMDOC allows to capture the semantics and structure of these documents. Various tools, such as XSL transformation, are available to transform OMDOC documents into other formats for presentation purposes (using, e.g., MathML) or to support inter-system communication (e.g., by transformation into the logic of a theorem prover).

Task 3.2: Support to the Development of an Industrial-Strength Application of Formal Methods to Program Verification

TASK LEADER: USAAR
SCIENTISTS IN CHARGE: JÖRG SIEKMANN, CHRISTOPH BENZMÜLLER
RESEARCH TEAM: USAAR, UED, ITC-IRST

3.2.a Overview

The primary goal in this work task is to investigate and emphasize the relevance of the methods and mathematical service tools developed by the Calculemus Network for the application of formal methods to program verification. While formal methods is undoubtedly a very prospectous application area for our research, we have, in addition, identified the education sector as another interesting area where our contributions may have an impact. To stay abreast of changes we propose a respective amplification of the definition of research task 3.2 in the yearly report for 2002.

We briefly sketch some applications in the formal methods and education area.

Formal Methods Applications

- In a cooperation between Saarland University, the semi-industrial German Research Centre for Artificial Intelligence (DFKI), and the University of Edinburgh an approach to support the verification of hybrid systems with the help of mathematical services in MathWeb is currently being developed and analyzed [42, 41]. An internship of a young visiting researcher of the network in a major international communication systems company is currently being accomplished with the aim to investigate the relevance of this verification approach for practical applications.
- In a cooperation with the DFKI in Saarbrücken, University of Genua, and Saarland University we investigate whether specialized reasoning tools within the MathWeb-sb can fruitfully support the formal verification of information flow properties. Information flow properties can be used to express confidentiality and integrity requirements of systems.
- A similar topic has been investigated in a cooperation between the University of Edinburgh and the University of Genoa. It concerns error dedection in security protocols.

- At ITC-IRST in Trento a symbolic verification technique that extends Bounded Model Checking is investigated and MathSat is employed in this context. Another topic is error detection in security protocols with SAT-based model checking.
- University of Edinburgh tries to apply proof planning in first order linear temporal logic (FOLTL) to feature interactions as they arise in large telephone networks.

Education systems related activities

- The Theorema system, which has been presented in Task 3.1, is employed in education practice already at University of Linz.
- ACTIVEMATH [197] is a learning environment for mathematics being developed mainly at the DFKI in Saarbrücken and partly at the University of Saarbrücken. The goal of the project's research and development is a web-based interactive learning system (for mathematics) that uses instruction as well as constructivist elements. The project provides an architecture, basic knowledge representations, and techniques for new-generation on-line interactive mathematics documents (textbooks, courses, tutorials) and e-learning. The ACTIVEMATH system, which makes or wants to make use of mathematcial services and techniques developed in the network, is already prototypically applied in lectures at Saarland University.

  ACTIVEMATH cooperates with external systems, such as proof planning systems or computer algebra systems. They can be called in order to support a user in exercise problem solving or to check the correctness of a solution. External systems can also be used to present examples. In particular, the knowledge acquired for automated proof planning can be used to explicitly teach mathematical methods and control knowledge. At present, ACTIVEMATH rather concentrates on employing external systems individually and independent from each other. However, it will clearly benefit from the research of the network on the integration of systems for deduction and computation, since many exercises do require such an interplay.

A subset of the above applications will now be discussed in more detail.

3.2.b Hybrid System Verification

Hybrid systems are heterogenous dynamical systems characterized by interacting continuous and discrete dynamics. The enormous presence of hybrid systems in safety critical applications, such as automated highway systems [138], air traffic management systems [178], embedded automotive controllers [28], and chemical processes [120], increasingly calls for safety guarantees. Since traditional program verification methods allow at best to approximate continuously changing environments by discrete sampling, special verification methods for hybrid systems, such as [132, 133, 134], have been developed. A frequently employed method is to model hybrid systems by hybrid automata. A hybrid automaton is a closed system with a *built-in* control structure determining when and how the system switches between its various discrete states. Thereby the continuous behaviour in each discrete state is governed by a differential equation. The verification method we will employ in our work is the deduction-based model checking approach for hybrid systems described in [212]. Given a specification of a hybrid system H (a hybrid automaton) and a safety property Φ the approach generates a second order formula $\lceil\Phi\rceil_H$ such that the validity of the latter guarantees that property Φ is valid for H. To support the validation of $\lceil\Phi\rceil_H$ this method eliminates second order location predicates in $\lceil\Phi\rceil_H$ one by one in order to transform $\lceil\Phi\rceil_H$ into an equivalent first order formula Ψ, if possible. With the validation of Ψ the verification approach terminates.

For the above deduction-based model checking approach we have identified the following mathematical subtasks: (1) The solution of sets of differential equations, (2) checking subsumption between sets of constraints, and (3) checking consistency of sets of constraints.

In general, solving these tasks is feasible in case of linear constraints and linear differential equations. Our aim, however, is to widen the spectrum of the approach, for instance, by allowing also non-linear constraints and differential equations. Mathematical tasks like (1) – (3) may also be relevant for other hybrid system verification approaches. For instance, [135] employed the computer algebra system MATHEMATICA to solve linear constraints. MATHEMATICA was later replaced by a more efficient implementation of a specialized constraint solving algorithm [136]. However, multiple implementations of the same kind of mathematical services in different verification systems could and should best be avoided, especially if their realization is complex and challenging, such as in our context.

We propose to model existing reasoning systems, such as computer algebra systems and constraint solvers, as mathematical services and to provide them in a network of mathematical tools in a way that they can reasonably support subtasks as they may occur in formal methods applications. The motivation is to make it simpler to implement and test verification approaches by out-sourcing complex but precisely identifiable mathematical subtasks for which specialized reasoners do already exist. Allowedly, in case a verification approach later turns out to be successful (see for instance [136]) it may be reasonable from efficiency aspects and also from concession aspects to replace the connections to mathematical services again by fast re-implementations of the particularly needed algorithms. However, starting with the latter may dramatically slow down a quick development and implementation of new verification environments. This is particularly true in case the automation of the mathematical subtasks is already on the edge of current research, such as given in our case. This motivates our proposal to build up a network of mathematical reasoning services for formal methods. The more services will be appropriately added to such a network the more likely it will be that also other verification approaches can directly employ them (in early development stages) for the same purpose.

We summarize the deductive model checking approach described in [212] and identify the subtasks we want to model as mathematical service requests.

Modelling of Hybrid Systems Hybrid systems are modelled as hybrid automata, which are presented as finite graphs whose nodes correspond to global states (locations). An example of an hybrid automaton is given in Fig. 18.

The local reachability theory is a logical representation of all immediate future states that can be reached from a given state. For instance, the local reachability theory for state Two of the automaton in Fig. 18 is given as the first order formula

$$\forall x, y. Two(x,y) \rightarrow \begin{cases} y \geq 5 \wedge \\ \forall \delta\ \delta \geq 0 \wedge y - 2\delta \geq 5 \rightarrow Two(x+\delta, y-2\delta) \wedge \\ y = 5 \rightarrow Three(0,y) \end{cases}$$

Here the possible immediate state transitions are characterized by the conjunct $y = 5 \rightarrow Three(0,y)$. In our example, $Three(0,y)$ is the only potential immediate future location of $Two(x,y)$ and $y = 5$ is a guard for the transition from Two to $Three$. In case of a transition data variable x is re-initialized with 0. The first conjunct tells us that any valid state $Two(x,y)$ has to satisfy the location invariant $y \geq 5$. Finally, the second conjunct characterizes the potential timed successors, i.e., the change of the data variables x and y in case no transition to a successor state occurs. Similarly, an inevitability theory is introduced which characterizes all states that are inevitable.

For the construction of local reachability theory for L it is generally required to solve the differential equations characterizing the timed successors of L. In our example we only consider very simple linear differential equations like $\dot{x} = 1$ and $\dot{y} = -2$ in state Two. They express that the value of x continuously increases over time with slope 1 and that the value of y continuously decreases over time with slope 2. The corresponding solutions for the differential equations are $x(\delta) = \delta + c$ and $y(\delta) = -2\delta + d$. Note that respective information is required for the formulation of the formula $\forall \delta\ \delta \geq 0 \wedge y - 2\delta \geq 5 \rightarrow Two(x+\delta, y-2\delta)$ characterizing the timed successors. If we extend the approach to handle also sets of non-linear differential equations, such as $\{\dot{x} = x + y, \dot{y} = 3\}$ or $\{\dot{x} = \sqrt{x+3}, \dot{y} = 1, \ddot{x} = y * x\}$, the

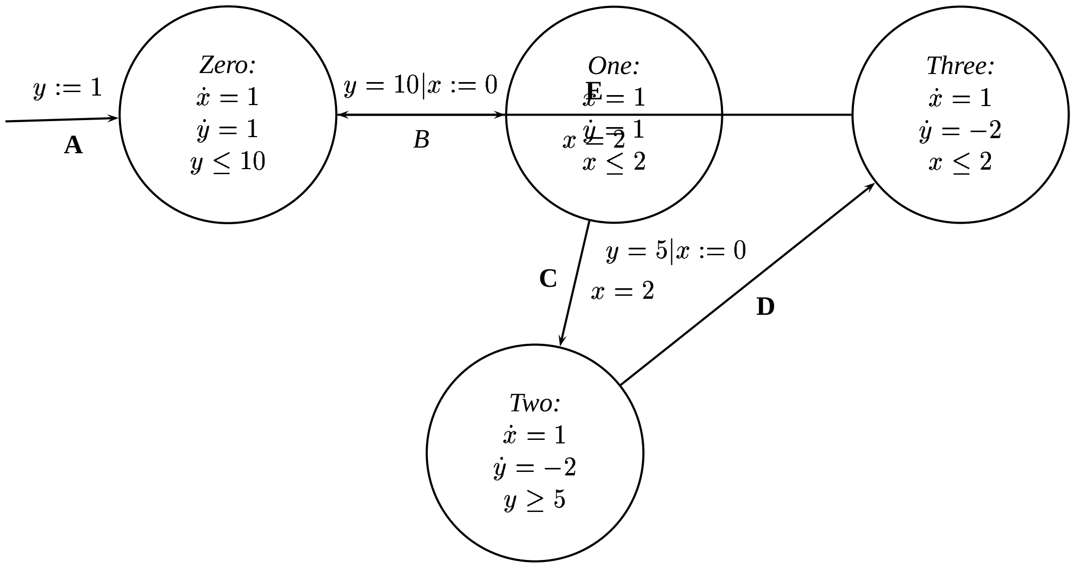


Figure 18: *Water level monitor:* the water level in a tank is controlled through a monitor, which continuously senses the water level and turns a pump on and off. We wish to keep the water level (denoted by the data variable y) always between 1 and 12. The figure describes the monitor that signals whenever the water level passes 5 and 10 inches. x is the clock of the system that describes the delay of the pump start-up/shutdown. We have to prove the *ICTL* property $AG(1 \leq y \wedge y \leq 12)$.

construction of the timed successor formulas can become a complicated task. Thus, the follwoing relevant mathematical tasks can be identified:

Mathematical service 1: Solving of differential equations To support the construction of the timed successor formulas we propose to identify suitable mathematical service systems for solving sets of differential equations. Our interest is to widen the current spectrum of the approach, for instance, by extending it to non-linear differential equations or second derivatives in the specification of hybrid systems.

Proving Properties of Hybrid Systems Properties of hybrid systems are described by formulas of the *Integrator Computation Tree Logic ICTL* [2]. Here we avoid a formal introduction to *ICTL*.

An example property for our hybrid in Fig. 18 is that for all reachable states holds that $1 \leq y \wedge y \leq 12$. With the help of the *ICTL* always operator AG this can be formalized as $\Phi = AG(1 \leq y \wedge y \leq 12)$.

In the context of our work we are interested in extending the definition of *ICTL* formulas such that they may also contain mathematical concepts such as $\sqrt{T}$, $sin(T)$, $cos(T)$, $tan(T)$, $cotan(T)$, and ${T_1}^{T_2}$. Given a hybrid system *H*, e.g the system described in Fig 18, and a property $\Phi \in$ *ICTL*, e.g. $\Phi = AG(1 \leq y \wedge y \leq 12)$. We are interested to analyse the validity of Φ for *H*. $\hat{\phi} = \{\phi \mid H, (L, \phi) \models \Phi\}$ describes the set of all valuations ϕ where property Φ is true. Suppose $\hat{\phi}$ can be described by a (finite) characteristic constraint formula $\lceil\Phi\rceil_H^L \in$ *CF*. Intuitively a characteristic constraint formula describes the necessary and sufficient conditions on the data variables such that Φ holds for L in H. Checking whether $H, (L, \phi) \models \Phi$ holds can be reformulated as to checking whether $\phi(\lceil\Phi\rceil_H^L)$ is valid. Generally, it is not possible to find a characteristic constraint formula. Instead of attempting to construct $\lceil\Phi\rceil_H^L$ directly, it is described first as a formula of the second order predicate calculus. Then the elimination approach tries to simplify this description to a first order constraint formula step by step, if possible.

In our example we are interested to verify property $\Phi = AG(1 \leq y \wedge y \leq 12)$ for automaton H sketched in Fig. 18. This means we want to verify that $H, (Zero, \phi) \models AG(1 \leq y \wedge y \leq 12)$ for all initial states

$(Zero, \phi)$ and this reduces to checking whether $\phi(\lceil AG(1 \le y \wedge y \le 12)\rceil_H^{Zero})$ for all ϕ. According to our approach $\lceil AG(1 \le y \wedge y \le 12)\rceil_H^{Zero}$ is given as the following second order formula:

$$\exists\ \begin{array}{l} Zero \\ One \\ Two \\ Three \end{array} \left(\begin{array}{lcl} Zero(x,1) & & \\ \forall x,y\ Zero(x,y) & \rightarrow & \left\{ \begin{array}{l} y \le 10 \\ \forall\delta\ \delta \ge 0 \wedge y+\delta \le 10 \rightarrow Zero(x+\delta, y+\delta) \\ y = 10 \rightarrow One(0,y) \end{array}\right. \\ \forall x,y\ One(x,y) & \rightarrow & \left\{ \begin{array}{l} x \le 2 \\ \forall\delta\ \delta \ge 0 \wedge x+\delta \le 2 \rightarrow One(x+\delta, y+\delta) \\ x = 2 \rightarrow Two(x,y) \end{array}\right. \\ \forall x,y\ Two(x,y) & \rightarrow & \left\{ \begin{array}{l} y \ge 5 \\ \forall\delta\ \delta \ge 0 \wedge y-2\delta \ge 5 \rightarrow Two(x+\delta, y-2\delta) \\ y = 5 \rightarrow Three(0,y) \end{array}\right. \\ \forall x,y\ Three(x,y) & \rightarrow & \left\{ \begin{array}{l} x \le 2 \\ \forall\delta\ \delta \ge 0 \wedge x+\delta \le 2 \rightarrow Three(x+\delta, y-2\delta) \\ x = 2 \rightarrow Zero(x,y) \end{array}\right. \\ \forall x,y\ Zero(x,y) & \rightarrow & 1 \le y \wedge y \le 12 \\ \forall x,y\ One(x,y) & \rightarrow & 1 \le y \wedge y \le 12 \\ \forall x,y\ Two(x,y) & \rightarrow & 1 \le y \wedge y \le 12 \\ \forall x,y\ Three(x,y) & \rightarrow & 1 \le y \wedge y \le 12 \end{array} \right.$$

Here $Zero(x, 1)$ characterizes the initial state. The next four conjuncts specify the reachability theory for H composed from the local reachability theories for all states as described before. The last four conjuncts finally guarantee the validity of property $(1 \le y \wedge y \le 12)$ within all (reachable) states. This formula is equivalent to

$$\exists\ \begin{array}{l} Zero \\ One \\ Two \\ Three \end{array} \left(\begin{array}{lcl} Zero(x,1) & & \\ \forall x,y\ Zero(x,y) & \rightarrow & \left\{ \begin{array}{l} y \le 10 \wedge 1 \le y \wedge y \le 12 \\ \forall\delta\ \delta \ge 0 \wedge y+\delta \le 10 \rightarrow Zero(x+\delta, y+\delta) \\ y = 10 \rightarrow One(0,y) \end{array}\right. \\ \forall x,y\ One(x,y) & \rightarrow & \left\{ \begin{array}{l} x \le 2 \wedge 1 \le y \wedge y \le 12 \\ \forall\delta\ \delta \ge 0 \wedge x+\delta \le 2 \rightarrow One(x+\delta, y+\delta) \\ x = 2 \rightarrow Two(x,y) \end{array}\right. \\ \forall x,y\ Two(x,y) & \rightarrow & \left\{ \begin{array}{l} y \ge 5 \wedge 1 \le y \wedge y \le 12 \\ \forall\delta\ \delta \ge 0 \wedge y-2\delta \ge 5 \rightarrow Two(x+\delta, y-2\delta) \\ y = 5 \rightarrow Three(0,y) \end{array}\right. \\ \forall x,y\ Three(x,y) & \rightarrow & \left\{ \begin{array}{l} x \le 2 \wedge 1 \le y \wedge y \le 12 \\ \forall\delta\ \delta \ge 0 \wedge x+\delta \le 2 \rightarrow Three(x+\delta, y-2\delta) \\ x = 2 \rightarrow Zero(x,y) \end{array}\right. \end{array} \right.$$

where we have exactly one location predicate for each state.

The Elimination Approach The remaining problem consists in proving the validity of the second order formula $\lceil AG(1 \le y \wedge y \le 12)\rceil_H^{Zero}$ above. The elimination approach proposes a stepwise transformation of these kinds of formulas to equivalent first order statements. This is done by a one by one elimination of location predicates in $\lceil AG(1 \le y \wedge y \le 12)\rceil_H^{Zero}$ based on the elimination theorem [213]. General applications of the elimination theorem require the evaluation of a greatest fixpoint for the location variable to be evaluated. For special applications, however, the elimination theorem can be refined by a much simpler simplification lemma which does not require such complicated computations. The special cases are those where a state L does not have direct edge transitions to itself, such as given for our automaton H. If we can validate $\lceil AG(1 \le y \wedge y \le 12)\rceil_{H''}^{Zero}$ we are done. Thus it remains to prove

$$\exists\ Zero \left(\begin{array}{lcl} Zero(x,1) & & \\ \forall x,y\ Zero(x,y) & \rightarrow & \left\{ \begin{array}{l} y \le 10 \wedge 1 \le y \wedge y \le 12 \\ \forall\delta\ \delta \ge 0 \wedge y+\delta \le 10 \rightarrow Zero(x+\delta, y+\delta) \\ x = 10 \rightarrow Zero(2,1) \end{array}\right. \end{array} \right.$$

Since we now face a self-loop the simplification lemma is no longer applicable and we have to proceed with the elimination theorem where a greatest fixpoint evaluation is generally required. In our special case the self-loop is directly subsumed by the initial state. In order to generally detect and evaluate fixpoints, however, we are interested to check whether a current iterative in the fixpoint computation is already subsumed by a previous iterative. The iterations thereby are typically growing compounds of first order constraint formulas like

$$y \leq 10 \wedge 1 \leq y \wedge y \leq 12$$

which is also the final greatest fixpoint to be evaluated in our example. As this constraint formula is consistent the deductive model checking approach tells us that property ϕ is indeed valid for H, hence, we are done.

Thus, we can identify the following two mathematical tasks:

Mathematical service 2: Subsumption of sets of constraints To support the fixpoint computations we are interested in subsumption checks for sets (or conjunctions) of constraints.

Mathematical service 3: Solving sets of constraints We are interested in the consistency of sets of constraints generated by the approach. Note that both tasks are relatively trivial in case of linear constraints. However, just as for the differential equations our interests is to widen the spectrum of the approach by, for instance, including non-linear constraints in the specification of hybrid systems. The challenge, however, will extend the elimination approach to handle also non-linear cases. This in turn poses a challenge for the requested mathematical services. Their strengths and weaknesses will determine how far we can go.

3.2.c Verification of Information Flow Properties

Being able to construct systems that are reliable even if they operate in hostile environments is of critical importance. When engineering secure systems one has to take into account that there are malicious programs like computer viruses or Trojan horses. Moreover, bugs in (otherwise non-malicious) programs may have similar devastating consequences. In order to ensure that critical systems indeed are secure the application of formal methods during their development appears most appropriate. This means that security requirements have to be specified formally in a way that they can be verified with mathematical rigour.

A very elegant approach to specify security requirements is to use information flow properties. Following this approach, e.g., the requirement that a particular input given to the system must not be leaked to some user is specified by the requirement that the actions of the system at the interface to that user do not depend on the confidential input. Various different ways to specify information flow properties have been proposed over the last 20 years. The approach that we follow has been proposed in [182]. The core of this approach is a framework (called *MAKS*) for the representation of information flow properties. In *MAKS* an information flow property is specified by a pair consisting of a view (specifying where information flow is restricted) and a security predicate (defining what restricted information flow means). Security predicates are assembled from so-called basic security properties, which are very primitive information flow properties. This modular representation of information flow properties has motivated the name of the framework, i.e. *Modular Assembly Kit for Security Properties*, which we abbreviate by *MAKS*.

Techniques that simplify the verification of information flow properties have been suggested in [183]. These so called unwinding results reduce the task to verify complex information flow properties to the task of verifying simpler local verification conditions, the unwinding conditions. Finding ways to automate the verification of these local conditions has been the object of our investigations. In the following, we will illustrate the techniques that we have developed using a simple example and will compare the advantages and disadvantages of different approaches.

We proceed as follows: Firstly, we specify the example system as an event system. Secondly, we specify the information flow property that shall be verified. Thirdly, we specify an unwinding relation, i.e. a binary

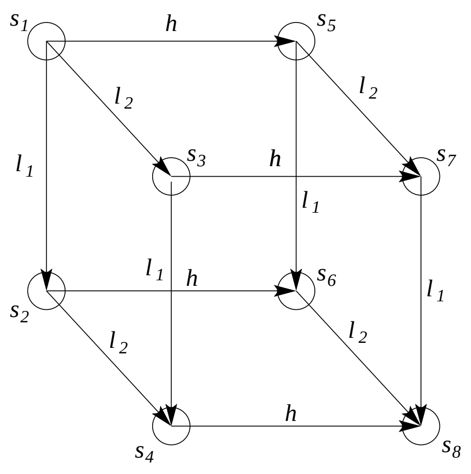


Figure 19: A simple security example

relation between states. This is a necessary prerequisite for verifying unwinding conditions. Fourthly, we state the instantiated unwinding conditions, i.e. the theorems that we have proved. After this we are ready to elaborate on the actual verification effort and our experiences made with different approaches and tools.

System Specification We consider a very simple system with only 8 different *states*, i.e. $s_1, \ldots, s_8$. The set S of states is specified by a predicate *is-in-S* (see below). For our example system, there are only three *events* (events model atomic actions), i.e. l_1, l_2 and h. The set E of events is specified by a predicate *is-in-E*.

In the following, we assume that the three names for events and the eight names for states all refer to different objects, i.e. $l_1 \neq l_2, l_1 \neq h, l_2 \neq h$ and $s_i \neq s_j$ if $i \neq j$ hold.

$$\begin{array}{lcl} \forall e.\textit{is-in-E}(e) & \Leftrightarrow & (e = l_1 \vee e = l_2 \vee e = h) \\ \forall s.\textit{is-in-S}(s) & \Leftrightarrow & (s = s_1 \vee s = s_2 \vee s = s_3 \vee s = s_4 \\ & & \vee s = s_5 \vee s = s_6 \vee s = s_7 \vee s = s_8) \end{array} \tag{2}$$

The possible transitions for the example system are viewed in Figure 19. Note that each transition involves a starting state s, an event e that causes the transition, and a resulting state s'. The transition relation is specified by a predicate *is-in-T*.

$$\begin{array}{ll} \forall s, s', e.\textit{is-in-T}(s, e, s') \Leftrightarrow & [(e = h \wedge ((s = s_1 \wedge s' = s_5) \vee (s = s_2 \wedge s' = s_6) \\ & \vee (s = s_3 \wedge s' = s_7) \vee (s = s_4 \wedge s' = s_8))) \vee \\ & (e = l_1 \wedge ((s = s_1 \wedge s' = s_2) \vee (s = s_3 \wedge s' = s_4) \\ & \vee (s = s_5 \wedge s' = s_6) \vee (s = s_7 \wedge s' = s_8))) \vee \\ & (e = l_2 \wedge ((s = s_1 \wedge s' = s_3) \vee (s = s_2 \wedge s' = s_4) \\ & \vee (s = s_5 \wedge s' = s_7) \vee (s = s_6 \wedge s' = s_8)))] \end{array} \tag{3}$$

Specification of Security Property The specification of a security property involves that all events are associated with security domains. For our simple example system, we assume only two security domains, i.e. H (for high) and L (for low). The security requirement is that no information shall flow from high to low. This can be read as the confidentiality requirement: information in the high domain must not be obtainable for the low domain. The association of events with security domains is specified by a function *dom*.

$$\begin{array}{lcl} \forall e.\textit{dom}(e) = H & \Leftrightarrow & e = h \\ \forall e.\textit{dom}(e) = L & \Leftrightarrow & (e = l_1 \vee e = l_2) \end{array} \tag{4}$$

Besides the association of events with security domains, the specification of security properties involves two further tasks. These are

- specification of a *flow policy*, saying where flow of information is allowed/ forbidden
- specification of a *security predicate*, defining when a restriction to the flow of information is satisfied

For our simple example, the flow policy simply says that there should not be any information flow from H to L.

As security predicate we choose *BSD*. This basic security predicate is formally defined, e.g, in [184]. However, this definition is not required here because, according to a result in [183], *BSD* is implied by the two conditions *lrf* and *osc* that will be defined further in the text. Hence, what we have to do during proof search is: we have to verify these two conditions.

Specification of Unwinding Relation The unwinding relation $\ltimes$ is a binary relation between states. This relation is specified as follows.

$$\begin{array}{lll} \forall s, s'. \quad (\ltimes(s, s') & \Leftrightarrow & ((s = s_5 \wedge s' = s_1) \vee (s = s_6 \wedge s' = s_2) \\ & & \vee(s = s_7 \wedge s' = s_3) \vee (s = s_8 \wedge s' = s_4))) \end{array} \tag{5}$$

The unwinding relation is an auxiliary object that is used as a parameter of the conditions *lrf* and *osc* below.

Unwinding Conditions The two unwinding conditions *lrf* and *osc* imply that *BSD* holds. These conditions can be specified as follows:

$$\begin{array}{lll} \mathit{lrf}_{E,S,T,\ltimes} & \Leftrightarrow & \forall s, e, s'.((\mathit{dom}(e) = H \wedge \mathit{is\text{-}in\text{-}T}(s, e, s')) \Rightarrow \ltimes(s', s)) \\ \mathit{osc}_{E,S,T,\ltimes} & \Leftrightarrow & \forall s, e, s', u, u'.((\mathit{dom}(e) = L \wedge \mathit{is\text{-}in\text{-}T}(s, e, s')) \wedge \ltimes(u, s)) \\ & & \Rightarrow \exists u'.(\mathit{is\text{-}in\text{-}T}(u, e, u') \wedge \ltimes(u', s')) \end{array} \tag{6}$$

Overall Proof Obligation Using the previous definitions, we want to prove the following theorem:

$$(2) \wedge (3) \wedge (4) \wedge (5) \wedge (6) \vdash \mathit{lrf} \wedge \mathit{osc} \tag{7}$$

The General Case Some remarks in order to avoid that misunderstandings arise from the simplicity of this example.

- We have fixed the unwinding relation. In the general case, the unwinding relation is not given. Thus, to find an unwinding relation for which the unwinding conditions hold is a subtask in proof construction.
- The example system is finite. The theory of security that we use can also be used for infinite systems. We make this simplification here, in order to set up a simple starting point.
- There are other basic security predicates than *BSD* in the theory. The corresponding unwinding conditions differ from the ones for *BSD*. However, they are still quite similar. So if we succeed for *BSD*, there is a good chance that this will extend to other basic security predicates.

Employing SAT or QBF solver to tackle the problem As part of the Calculemus initiative we want to investigate whether specialized and powerful tools for testing the satisfiability of quantified boolean formulae (QBF) can be employed as mathematical services to support the verification of information flow properties as the ones sketched above.

Mathematical Service 1 & 2 Among the systems we want to employ are the SAT solver Chaff, which is suited for solving large real world SAT instances, and the QBF-solver QuBE (version 1.3).

Mathematical Service 3 We have to to provide appropriate transformation services in order to map information flow properties specified in first order predicate logic as illustrated above to the respective input format requested by the SAT and QBF solvers we want to employ.

As part of the project we also want to explore the differences between the SAT the QBF approach. Ideally this analysis leads to a strategy that chooses the best approach and an optimal encoding for the problem at hand.

The SAT and QBF encoding of the above example For a proper encoding into a SAT or QBF problem we have to transform the original problem formulation into a boolean formulation. As a preliminary step we must provide binary encodings for *events* and *states*.

For the events we choose the following encoding: $\lceil h \rceil = 00$, $\lceil l_1 \rceil = 01$, and $\lceil l_2 \rceil = 10$. For the states the following encoding is particularly convenient: $\lceil s_i \rceil = (i-1)_b$ for $i = 1, \ldots, 8$, where n_b is the binary representation of the number n. For instance, $\lceil s_1 \rceil = 000$ and $\lceil s_3 \rceil = 010$.

We now provide new encodings for the formulae (3), (5), and (6) defined before. This can be done by replacing each variable of “sort” event, say e, with two boolean variables, say $\mathbf{e}_0$ and $\mathbf{e}_1$, and each variable of “sort” state, say s, with three boolean variables, say $\mathbf{s}_0$, $\mathbf{s}_1$, and $\mathbf{s}_2$.

The transition relation then becomes:

$$\textit{is-in-T}(\mathbf{s}_0,\mathbf{s}_1,\mathbf{s}_2,\mathbf{e}_0,\mathbf{e}_1,\mathbf{s}'_0,\mathbf{s}'_1,\mathbf{s}'_2) \equiv \left[\begin{array}{c} \neg\mathbf{e}_0 \wedge \neg\mathbf{e}_1 \wedge \neg\mathbf{s}_0 \wedge \mathbf{s}'_0 \wedge \mathbf{s}_1 \leftrightarrow \mathbf{s}'_1 \wedge \mathbf{s}_2 \leftrightarrow \mathbf{s}'_2 \\ \vee \\ \neg\mathbf{e}_0 \wedge \mathbf{e}_1 \wedge \neg\mathbf{s}_2 \wedge \mathbf{s}'_2 \wedge \mathbf{s}_0 \leftrightarrow \mathbf{s}'_0 \wedge \mathbf{s}_1 \leftrightarrow \mathbf{s}'_1 \\ \vee \\ \mathbf{e}_0 \wedge \neg\mathbf{e}_1 \wedge \neg\mathbf{s}_1 \wedge \mathbf{s}'_1 \wedge \mathbf{s}_0 \leftrightarrow \mathbf{s}'_0 \wedge \mathbf{s}_2 \leftrightarrow \mathbf{s}'_2 \end{array}\right]$$

The unwiding relation becomes:

$$\ltimes(\mathbf{s}_0,\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}'_0,\mathbf{s}'_1,\mathbf{s}'_2) \equiv (\mathbf{s}_0 \wedge \neg\mathbf{s}'_0 \wedge \mathbf{s}_1 \leftrightarrow \mathbf{s}'_1 \wedge \mathbf{s}_2 \leftrightarrow \mathbf{s}'_2)$$

To prove the unwinding conditions, say (6.1) and (6.2), by means of a SAT-solver we turn them into satisfiability problems by reasoning by refutation. For instance, the validity of (6.1) is equivalent to the unsatisfiability of its negation which, after some obvious simplifications, is:

$$\exists s, s'.(\textit{is-in-T}(s,h,s') \wedge \neg \ltimes (s',s)) \tag{8}$$

The boolean encoding of (8) is:

$$\begin{array}{ll} \exists \mathbf{s}_0,\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}'_0,\mathbf{s}'_1,\mathbf{s}'_2. & (\neg\mathbf{s}_0 \wedge \mathbf{s}'_0 \wedge \mathbf{s}_1 \leftrightarrow \mathbf{s}'_1 \wedge \mathbf{s}_2 \leftrightarrow \mathbf{s}'_2 \wedge \\ & \neg(\mathbf{s}'_0 \wedge \neg\mathbf{s}_0 \wedge \mathbf{s}'_1 \leftrightarrow \mathbf{s}_1 \wedge \mathbf{s}'_2 \leftrightarrow \mathbf{s}_2)) \end{array} \tag{9}$$

From the unsatisfiability of (9) we can readily conclude the validity of (6.1).

As above, the validity of (6.2) is equivalent to the unsatisfiability of its negation which is:

$$\begin{array}{ll} \exists s, s', u. & ((\textit{dom}(e) = L \ \wedge \textit{is-in-T}(s,e,s')) \ \wedge \ \ltimes(s,u) \wedge \\ & \forall u'.(\neg(\textit{dom}(e) = L \ \wedge \textit{is-in-T}(u,e,u')) \ \vee \ \neg \ltimes (s',u'))) \end{array} \tag{10}$$

Where $L = l_1, l_2$.

This can be encoded into a boolean formula as in the previous case with the only additional difficulty due to the replacement of the subformula

$$\forall u'.(\neg(\textit{dom}(e) = L \ \wedge \textit{is-in-T}(u,e,u')) \vee \neg \ltimes (s',u')) \tag{11}$$

with the corresponding expansion. The resulting boolean formula can be fed to a SAT-solver to decide its unsatisfiability.

To underline differences between SAT and QBF encoding, we have to investigate the quantifiers rule into the encoding. SAT-encodings require that all variables involved must be existential quantified. When we encounter a universal one, we have to unfold it, using a ground instantiation. This easily explains the exponential growing of the space involved into the computation. Using QBF-encoding, we can maintain universal quantifiers, declaring them in the head of the Q-dimacs format(a variant of dimacs). The main advantages are the limitation in the growing and an easier encoding.

Moreover, our strategy works with the quantifier list, trying to reduce the number involved into the PNF. Every time we encounter as last quantifier close to the formula, a universal one, we purge all its instances into the PNF, reducing the number of variables. In the best case, we can reduce the PNF to a SAT-problem, where all involved variables are existential quantified. Looking at the previous example, the strategy results with a SAT-encoding, due to the elimination of the universal quantifier over u'.

Solving the problem with Chaff and QuBE As back-engines we choose state of the art solver in their fields. For SAT we employ Chaff [14] and for QBF our choice was QuBE [15]. We start encoding the problem into a Q-dimacs format, if the quantifier elimination results with only existential quantifiers, we convert it into a dimacs format. After the computation with the convenient solver, we parse the output, showing the path that satisfies the negation of the formula. Note that the solver returns only variables instantiated to true, all the others are intended to false.

3.2.d Error Detection in Security Protocols

A related topic has been investigated in a cooperation between the University of Edinburgh and the University of Genoa. It concerns the use of deduction/computation techniques to tackle the verification of industrial-strengths applications.

In particular, a procedure for error detection in security protocols has been studied and developed. The idea of this approach is to encode security protocol problems into propositional logic which can be effectively used to find attacks to protocols by exploiting the computational power of state-of-the-art SAT-Solvers (e.g. Chaff, SIM, etc.). While the approach is quite successful in finding attacks in security protocols, if the analyzed security protocol is not affected by an attack, the procedure may not terminate. Hence, it has been started a study on how to extend the approach from falsification to verification. By security protocol falsification we mean the problem to prove that the analyzed security protocol does not satisfy the specified security property (e.g. secrecy, authentication, etc.) by returning the appropriate counter-example. Viceversa, with security protocol verification we mean the problem of proving that the analyzed security protocol satisfies the specified security property.

Notice that the security protocol verification problem is undecidable in the general case. Therefore, assumptions about the security protocols have been identified in order to ensure termination. However, the related work done in this domain is often characterized by strong assumptions that make the results applicable only to restricted class of protocols. Our aim is to extend and integrate the ideas/techniques proposed in the related work and to adapt it to our approach based on the reduction of the security problems to propositional satisfiability.

[14]`http://www.ee.princeton.edu/~chaff/`
[15]`http://www.mrg.dist.unige.it/star/qube/`

3.2.e Bounded Model Checking for Timed System

ITC-IRST has successfully used the MathSat solver to solve verification problems for timed systems [23]. The verification of timed systems is a very important and challenging problem, in that it combines the challenge of handling finite-state variables, which is typically encoded as a boolean deduction problem, with the problems related to time elapsing, which are encoded into mathematical constraints on real variables. In fact, a state can be seen as an assignment to propositional variables and to real variables, representing absolute time and clocks.

The approach extends the Bounded Model Checking (BMC) [52] technique for the verification of timed systems, and is based on the following ingredients. First, a BMC problem for timed systems is reduced to the satisfiability of a mathematical formula, i.e., a boolean combination of propositional variables and linear mathematical relations over real variables (used to represent clocks). Then, the MathSat procedure described in the previous sections is used to check the satisfiability of the resulting formula.

The approach is rather general, since it allows to express specifications in full Linear Temporal Logic (LTL), such as fairness properties. Furthermore, the approach is fully symbolic: it allows one to tackle the digital component of timed systems with symbolic technologies as in the untimed case, while the timed component is tackled by means of specialized mathematical constraint solvers. Finally, the mathematical formulas generated are polynomial with respect to the size of the representation of the input system and the maximum path length k, and are solved by a solver requiring a polynomial amount of memory. The experimental analysis performed in [23] confirms the potential of the approach and shows that, with a proper tuning, MathSat can overperform traditional approaches based on Difference Bound Matrices (DBMs), Difference Decision Diagrams (DDDs), or Clock Difference Diagrams (CDDs).

3.2.f Proof Planning in First-Order Linear Temporal Logic

University of Edinburgh has investigated the combination of proof planning and reasoning in first-order temporal logics, in particular the linear, discrete time variant (FOLTL). The original claims of the project were that (a) there are examples of problems which naturally fit into FOLTL, (b) Proof Planning can help.

During these years, a theoretical framework has been devised, consisting of a family of labelled sequent calculi, sound and complete for a wide range of first-order modal logics, which can be extended to FOLTL, obviously losing completeness but keeping some of the benefits. See [101] for more details.

On the practical side, the problem of Feature Interactions in telecommunication systems (FI) is currently being addressed; an example has been mechanized (see [100]) and a full set of FI test cases is under examination.

As far the above items are concerned, (a) is confirmed by the FI test case and as well by some other publications (e.g., [1] and [273]); (b) is the object of the current study.

Task 3.3: Support to the Solution of Undergraduate Exam in Calculus and Economics

TASK LEADER: USAAR
SCIENTISTS IN CHARGE: JÖRG SIEKMANN, CHRISTOPH BENZMÜLLER
RESEARCH TEAM: USAAR, UED

3.3.a Overview

The aim of this task is to apply systems and approaches developed in the network to mathematical problems as they arise in maths education. Our initial proposal was to choose problems as they occur in exams (e.g. Harvard) in calculus and economics.

We propose to slightly modify the definition of the problem domain to be considered in this task. Instead of strictly sticking to the proposal of considering Harvard calculus and economics exams we propose to leave the choice of problems more flexible. A constraint, however, should be that the problems considered in this task are rather at math exam level than on math research level. Calculus remains, of course, a problem domain of interest.

The research question in this task is not whether the approaches and systems developed in the calculemus network are capable of solving challenging and probably open mathematical problems. This is the aim of the Task 3.5 and there we actually illustrate how challenging theorems such as the fundamental theorem of algebra and the fundamental theorem of analysis are attacked.

In this task we rather more focus on much simpler and maths education oriented problems with a strong emphasis on the particular way the problems are solved, how interaction with the user is supported and how the solution is presented. We want to analyze whether our systems can be employed in a user friendly and adequate way and whether the interaction and maths presentation capabilities of the systems are appropriate.

Different task relevant case studies have been completed, are currently carried out, or are planned for the near future. Amongst them are:

- Irrationality of $\sqrt{2}$
- Exercises from the German *Bundeswettbewerb Mathematik*
- Calculus examples from [36] currently investigated in the ACTIVEMATH project [197]. ACTIVEMATH is a web-based maths learning environment that currently encodes various parts of mathematical textbooks in order to make them available for online education purposes. The ACTIVEMATH

project wants to gradually improve its integaration and usage of computer algebra systems and deduction systems to support interactive exercises where the students can measure their learning progress. It thus appears useful to coordinate the exercise investigated in this project with the exercises to be investigated within ACTIVEMATH, since both parties may benefit from each others experience and preparatory work.

3.3.b Irrationality of $\sqrt{2}$

Henk Barendregt and Freek Wiedeijk at Nijmegen University proposed this case study, in which the integration of computation and deduction plays an important role. The idea is to compare the most prominent state-of-the-art systems with respect to a variety of criteria such as whether they support the de Bruijn principle (provide proof objects), the Poincare principle (capable of proving correctness of calculations automatically), facilitate a user-oriented interaction style, etc. The results of this case are reported in [265, 263]. In this case study also the systems Omega, Theorema, Coq, and Mizar developed by the partners in the Calculemus network particated and demonstrated their capabilities. Further information on the Omega solution on this case study, for instance, is available in [240, 48, 239].

3.3.c Exercises from the German *Bundeswettbewerb Mathematik*

Saarland University has begun to investigate examples from the German *Bundeswettbewerb Mathematik*[16] for high school maths students. These examples are very attractive since they typically have an elementary and elegant solution, which is often rather tricky to find. The motivation of these examples is to get students engaged in small, but interesting and challenging problems in order to stimulate their general interest for mathematics. The examples, which are usually formulated in natural language, can typically be formalized in various ways. Which formalisation may lead to the most elegant solution is often not easy to determine at the beginning. An important issue for tackling these problems is to provide an adequate system framework that is capable of supporting all relevant aspects including, for instance, playing with representations, acces to strong mathematical knowledge base, investigations of vague ideas at a rather abstract formalisation level, access to possibly integrated deduction and computation systems, and facilities to check a proof attempt at calculus level.

The following is an example exercise [17].

Problem: Given 51 points in a sqare with a side length of 7. Prove that there exists a unit circle containing at least 3 of them.

Solution: We cover the square with 25 smaller sqares of side length $7/5$. By application of the *pigeonhole principle*, one of them contains at least 3 points. Since the diagonal of the small quares are $7/5 * sqrt(2) < 2$, we can cover it with a unit circle. QED.

This example shows that we need to apply several lemmas from a knowledge base, including (a) pigeonhole principle, (b) it is possible to cover a square of side length a with n^2 smaller sqares of side length a/n (alternatively, we can use a more general version for rectangles), (c) for a square of side length a, there is a smallest circle covering it. It has a radius of $a/2 * sqrt(2)$. Also, we need to verify the inequality $1.4 < sqrt(2)$.

Although the examples of the Bundeswettbewerb are still rather simple they nevertheless require quite heterogeneous system support to be solved in a adequate way (from the perspective of a maths education system). Similar to the "Irrationality of $\sqrt{2}$" example they are also well suited to illustrate the functioning and working principles of our systems to high school scholars and maths students since they lie in the scope of their mathematical capabilities (at least when working with paper).

[16]See `http://www.bundeswettbewerb-mathematik.de/`
[17]See also `http://www.oliver-faulhaber.de/mathematik/bwm70.htm\#BWM722`

3.3.d Discussion

The intention of this task is to examine whether our systems and approaches can attack education oriented examples and present the respective results in a human oriented way. We are particularly interested in examples where a variety of requirements come into play, among them also the integration computation and deduction. While the work in the task has just begun, we can already conclude from the case studies on the *Irrationality of* $\sqrt{2}$ that in particular the issue of providing adequate maths formalization and representation facilities are not sufficiently solved yet. While the systems participating in this case study showed that they are indeed capable of tackling the problem sufficiently in case they are orchestrated by experts of these systems, we claim that no novice student user would easily be able to come up with similar solutions if he/she is not first given a detailed introduction to the pecularities of the system they use and its mathematical knowledge representation facilities. Hence, there is a challenge to attack the gap between the elegance and beauty informal maths often shows and the low level formalization tricks typically required in current systems.

Task 3.4: Modelling of Existing Systems as Mathematical Services

TASK LEADER: ITC-IRST
SCIENTISTS IN CHARGE: FAUSTO GIUNCHIGLIA, ROBERTO SEBASTIANI, MARCO BOZZANO, ALESSANDRO CIMATTI
RESEARCH TEAM: ITC-IRST, UWB, UGE

3.4.a Overview

The primary goal of this task is to investigate the possibility of modeling existing computer algebra systems and deductive systems as mathematical services. The work done so far has concentrated both on developing the required infrastructure (languages, protocols, semantic specifications, architectural schemata) for making existing systems interoperate, and on studying extensions and enhancements of the reasoning capabilities of some existing tools. The relevant contributions are:

- the **MathSat** framework developed at ITC-IRST [22, 21]. As previously said in task 1.2, the MathSat framework introduces a formal framework, a generalized algorithm and architecture for integrating boolean reasoners and mathematical solvers so that they can efficiently solve boolean combinations of boolean and mathematical propositions. Many techniques are described to optimize this integration. Moreover, the MathSat framework evidences the main requirements boolean reasoners and mathematical solvers must fulfill in order to achieve the maximum benefits from their integration. The MathSat procedure [20, 23] is ITC-IRST implementation of an integrated procedure based on the MathSat framework.
- **RDL** (Rewrite and Decision procedure Laboratory), developed by UNIGE, is a system for formula simplification developed within the Constraint Contextual Rewriting Project. The system allows for experimenting with the integration of decision procedures and conditional rewriting.
- **LBA** (Logic Broker Architecture), developed by UNIGE, is an architecture which provides the required infrastructure for making mechanized reasoning systems interoperate. In the LBA each mechanized reasoning system is seen as an entity providing and/or requiring a set of mathematical services. The LBA provides location transparency, a way to forward requests for logical services to appropriate reasoning systems via a simple registration/subscription mechanism, and a translation mechanism ensuring the transparent and provably sound exchange of logical services.
- Within the **MathWeb-SB** architecture, developed by USAAR, the generalisation algorithm for the learning of methods learning (described in task 2.3) has been added to the pool of mathematical services. Furthermore the proof planning system $\lambda Clam$, developed at UED, has been integrated into the MathWeb-SB framework. As a result, $\lambda Clam$ can now use reasoning services provided by existing systems in MathWeb-SB, and provide new reasoning services to them.

In the remainder of this report we discuss these contributions in more detail.

3.4.b The MathSat Framework

As pointed out in [22, 21], a significant number of existing procedures used in various application domains can be modeled within the MathSat framework. These procedures either are purely symbolic or combine symbolic and numeric techniques. We briefly recall some of them.

Omega [224] is a symbolic+numeric procedure used for dependence analysis of software. It is an integer programming algorithm based on Fourier-Motzkin variable elimination method. It handles boolean combinations of linear constraints by pre-computing the DNF of the input formula.

PtautEq [7] is a purely symbolic procedure which handles boolean combinations of boolean variables and equalities between first-order variables, which was embedded in the GETFOL [127] system. It combines a variant of DPLL boolean solver [113, 112] with an ad-hoc solver for sets of equalities.

SMV+QUAD-CLP [102] is an incomplete symbolic+numeric procedure integrating Ordered Binary Decision Diagrams, OBDDs [57], with a quadratic constraint solver to verify transition systems with integer data values. It performs a form of intermediate assignment checking.

TSAT [4] is an optimized symbolic+numeric procedure for temporal reasoning able to handle sets of disjunctive temporal constraints. It integrates DPLL with a simplex LP tool, adding some form of forward checking and (static) learning.

LPSAT [272] is an optimized symbolic+numeric procedure for math-formulae over linear real constraints, used to solve problems in the domain of resource planning. It accepts only formulae with positive mathematical constraints. LPSAT integrates DPLL with an incremental simplex LP tool, and performs backjumping and learning.

DDD's [200] are OBDD-like data structures handling boolean combinations of temporal constraints in the form $(x - z \leq 3)$, which are used to verify timed systems. They combine OBDDs with an incremental version of Belman-Ford algorithm.

ICS [121] is a mostly symbolic decision procedure for combined theories, including theory of arrays, bitvectors, lists and inductive datatypes, linear arithmetic over the integers. Very recently (2002) it has been integrated with the DPLL solver CHAFF [203].

CVC [246] is a symbolic+numeric decision procedure for combined theories, including theory of arrays, inductive datatypes, linear arithmetic over the reals. It combines, among others, the DPLL solver CHAFF with a Fourier-Motzkin procedure.

3.4.c The CCR Framework

The generality of Constraint Contextual Rewriting (CCR) is witnessed by the number of state-of-the-art systems whose simplification mechanisms can be seen as instances of CCR. These systems range from automated theorem provers (such as NQTHM, Tecton, and SPIKE) to computer algebra systems (such as MAPLE). In particular:

NQTHM. The simplifier of NQTHM can be seen as an engineered version of CCR(LA), where LA is a decision procedure for Universal Presburger Arithmetics over the Integers (UPAI). Details can be found in [11].

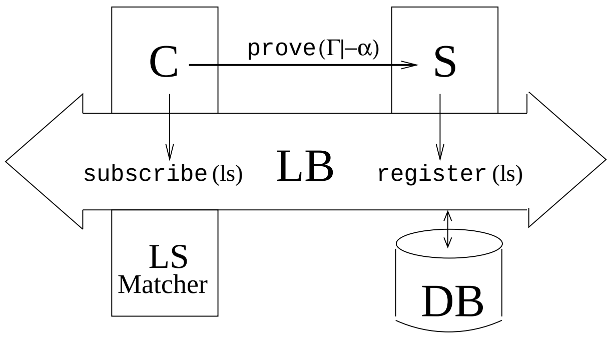


Figure 20: The Logic Broker Architecture. A client C wants to prove a formula; it subscribes its query to the Logic Broker (LB), waiting for a result. LB tries to find in the database (DB) a server matching the requested service and provide to C the service pointer.

Tecton. The simplifier of Tecton can be seen as CCR(LA+EQ), where LA+EQ is a decision procedure for the combination of UPAI and the Universal Theory of Equality (UTE). Details can be found in [11].

SPIKE. The simplifier of SPIKE can be seen as an extension of CCR(LA+EQ). The extension is here needed to allow the decision procedure to use induction hypotheses during proof by implicit induction. Details can be found in [16].

MAPLE. MAPLE's evaluation process can be seen as CCR(SR) where SR is a specialised reasoning module reproducing the functionalties of MAPLE's property reasoner and solver. Details can be found in [3].

3.4.d The LBA Architecture

UNIGE has both designed the conceptual model of the LBA [17, 18] and developed two prototype implementations of the LBA: one based on CORBA and – recently – one based on XML. Moreover, a bridge between LBA and MathWeb has been defined [276].

The Logic Broker Architecture (LBA) addresses the problems arising from the integration of different reasoning systems. In particular, interconnection of two different reasoners can lead to unsound results, because of differences in the underlying semantics. The LBA architecture addresses this problem by means of a diversification between the logic layer and the communication layer. A *reasoning theory* can be thought of as composed of a *sequent system* and some *inference rules*, which respectively model assertions and inference steps. Before instantiating a communication, a client sends to the Logic Broker a pair containing its set of inference rules and the service requested. The broker then tries to make a match between client query and services registered in its own database. If there is a positive result, then the Logic Broker provides the connection between the objects. The architecture can be seen from the point of view of the client like a library of functions, which can be easily integrated into the local environment, without any overhead coming from network connections. Note that the client does not ask about a specific server, but calls a service like `simplify` an equation, `solve` a set of constraints, etc. As a result, the same client can receive many solutions coming from different servers and it can apply some policies to decide which is the best for its computation. This level of decision can be shifted to the broker, asking for the first solution, or for the complete list of them.

One of the main goals of LBA is to use only consolidate standards, which can be easily implemented in the most common development environments. Due to this, the new version of LBA supports two main technologies, namely CORBA and XML. CORBA comes out from the tradition of LBA, guarantees the possibility to share and distribute not only results, but also parts of the proof or parts of the strategy, when this is possible. XML ensures the possibility to communicate with a large variety of web services and to

`Name`	ripple
`Context`	Rewrite Theory
`Types`	
`Input`	omdoc: `OMDoc`;
`Output`	result: `OMDoc`
`InConstraints`	elements(omdoc, Elements), lemmas(Elements, RewRules) member(sequent(_), Elements), not(RewRules = nil).
`OutConstraints`	elements(result, [Sequent]), not(member(Rule, RewRules), applicable(Rule, goal(Sequent)).
`TextDescriptions`	Tries to reduce the difference between the goal of the given OMDOC sequent and one of its hypotheses using LClams step_case method and the rewrites given as lemmas in the OMDOC.

Figure 21: The rippling service offered by $\lambda Clam$

interpolate data very fast. LBA also uses a standard for sharing mathematical documents, namely OPENMATH. Thanks to the extremely open design, OPENMATH can be used to describe a huge variety of mathematical knowledge. Everything is regulated by the agreement of the Content Dictionaries, which contain the classifications of mathematical symbols. Each client/server has its own phrase-book that translates the local language into the common layer OPENMATH.

3.4.e The MathWeb Software Bus

The architecture of MathWeb-SBhas been introduced in task 1.2. In this section we describe the integration of existing systems within this framework.

The $\lambda Clam$ Proof Planning System During his stay as a young visiting researcher (YVR) in Edinburgh, J. Zimmer integrated the proof planning system $\lambda Clam$ into the MathWeb-SB [114]. Due to this integration, $\lambda Clam$ can not only use the services of any reasoning specialist already integrated in the MathWeb-SB, such as the CAS MAPLE, but can also offer its theorem proving expertise to other systems in the MathWeb-SB. First, $\lambda Clam$ offers an inductive theorem proving service to the MathWeb-SB which takes a problem description formulated in OMDOC as an input and runs $\lambda Clam$ on the given problem. Second, the rippling heuristics of $\lambda Clam$ [241] is offered as a service that takes a set of rewrite rules and a proof planning sequent as an input and applies the rippling method of $\lambda Clam$ with the given rewrites. The two services offered by $\lambda Clam$ are new examples for mathematical services offered by the MathWeb-SB that have not been formally specified until now.

However, we tried to use the description language LARKS described in section 1.2.d to give a first characterization of the rippling service offered by $\lambda Clam$ (see figure 21).

Generalisation Algorithm of LearnΩmatic The application of learning techniques to proof planning has been described in task 2.3. LEARNΩMATIC is a hybrid system consisting of the implementation of a generalisation algorithm as computational component and the proof planner ΩMEGA. The generalisation algorithm has been integrated as an extremely specialized and new kind of mathematical service into MathWeb-SB. The generalisation algorithm accepts arbitrary sequences of objects in OMDOC syntax and returns a generalised description.

3.4.f Discussion

In this document we have reported the current status of the research about modeling existing systems as mathematical services. In particular, the work done so far has concentrated both on developing the required infrastructure (languages, protocols, semantic specifications, architectural schemata) for making existing systems interoperate, with an emphasis on the infrastructure for mathematical web services (see Section 3.4.d), and on studying extensions and enhancements of existing systems (see Sections 3.4.b and 2.3.h).

We expect a positive impact on these research direction to come from the work being done in the context of task 1.2, which concerns the definition of mathematical service.

Task 3.5: Challenge Mathematical Problems

TASK LEADER: UKA
SCIENTISTS IN CHARGE: JACQUES CALMET, VOLKER SORGE
RESEARCH TEAM: USAAR, UED, UKA, TUE, UWB, UBIR

3.5.a Overview

One of the obvious challenging mathematical problems relevant to this network is probably to answer the question “What are the limits of (artificial or natural) intelligence?”. This simple, apparently philosophical question leads in fact to very difficult mathematical problems such as the decidability of the Mandelbrot set. A brief description can be found in the January 2000 issue of the ”Gazette des Mathematiciens” published by the SMF, the French Mathematical Society. The paper of Smale proposes some challenging mathematical problems for the 21st century. The quest for mathematical challenges is to be found in any field. For instance, Jean E. Taylor introduces those for material science in the January 2003 issue of the Bulletin of the AMS (Vol. 40, No. 1).

To provide a definition of a challenge mathematical problem that can be accepted by all partners is almost impossible. However, if we introduce the word ”mechanized” in front of mathematical, we can then open a few tracks along the following directions (see [174]):

1. Mechanize new areas of Mathematics such as algebraic topology or even Grothendieck’s theory when also including geometry. Symbolic integration is a well-known example where a problem in Analysis was turned into an algebraic problem,

2. Devise new proofs techniques for domains where the amount of computation, not the theoretical difficulties is the challenge. An example is to prove some theorems on p-groups that would take a lifetime by hand calculation,

3. Identify and master new representation of mathematical objects. This is well understood when designing usual algebraic algorithms for CASs. A certainly challenging task is to investigate how algebraic fields could be introduced in mechanizing algebraic geometry problems,

4. Space and time complexity issues when designing proofs and algorithms. A prototypical example is the factorization of integer numbers. Other examples arise when trying to improve doubly exponential algorithms such as Groebner bases or when dealing with parameters which leads to constraint programming (see [27]).

An Example in Group Theory

A test problem is as follows, where the word *suitable* is used to avoid a too long presentation of the problem:

> *Given a "suitable" infinite collection of p-groups, give a formula for the least n such that the i-th group in the collection can be embedded in S_n, not in s_{n-1}.*

This is, according to the experts, a very long term project even when using DSs and CASs. However, when analyzing the problem, it is possible to identify sub-problems. Many of them are purely computational ones. For instance, one must compute determinants of matrices. Depending on the size of these matrices, a very thorough management of the computation is required. There are deduction problems as well. One of them is supposed to be simple and can be seen as a test of feasability.

> *Can we prove, by machine, that every subgroup of Q_{n^2}, the quaternion group of order 2^n, is normal?*

3.5.b Preliminary Work

During the work on other tasks some challenging mathematical problems had to be tackled already, in order to have non-trivial working examples. Here we briefly summarize that work that has been done so far in this respect. Some of the examples were done either by single partner nodes or in collaboration between some of the nodes.

Fundamental Theorem of Algebra:

At the Calculemus meeting in Eindhoven, Henk Barendregt has put as a "challenge problem" to formalize the Fundamental Theorem of Algebra in a theorem prover. (Every non-constant polynomial $f(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0$ over the complex numbers has a root, i.e. a $z \in \mathbb{C}$ such that $f(z) = 0$.) Consequently, this task has been taken up by the Mizar group and the research group of Barendregt at Nijmeen Nijmegen (subsite of EUT). In Nijmegen a *constructive* formalization of the Fundamental Theorem of Algebra in Coq has been made. We report on that development and its present continuations.

Formalizing a Constructive Proof of the Fundamental Theorem of Algebra in Coq

This work has been done by Herman Geuvers, Freek Wiedijk, Jan Zwanenburg, Randy Pollack, Milad Niqui, and Henk Barendregt from the University of Nijmegen, NL. It was called the *FTA project*. The motivations for starting this project were the following

- Formalize a large piece of real mathematics. See whether it can be done and which problems arise.
- Create a library for basic constructive algebra and analysis, to be used by others. Often, a formalization is only used by the person that created it (or is not used further at all!), whereas one of the goals of formalizing mathematics is to create a joint repository of mathematics.
- Investigate the current limitations of theorem provers, notably Coq, and the type theoretic approach towards theorem proving.
- Manage this project. Work with a group of people on one theory/proof-development. Initially, we distinguished the following three sequential/parallel phases:

Mathematical proof	LaTeX document (the mathematical proof with lots of details filled in)
Theory development	Coq file (just definitions and statements of lemmas)
Proof development	Coq file (formal proofs filled in)

The goal is to keep these phases consistent, so the theory/proof development process proceeds in a "literate programming" style: by working (in parallel) on three documents, one creates a complete formal development of FTA, together with a documentation, which consists of the LaTeX document (the high level specification) and the theory development (the low level specification, containing all the precise definitions and names of lemmas etc.) We note here that it is not trivial to keep these phases consistent (and in fact we did not maintain them till the end): a lemma in the LaTeX version may be just wrong, a definition may be incomplete or the 'basic properties' that one thinks one needs (say about fields) are just not the ones that one really needs.

- Constructive proof. We view a real number as a (potentially) infinite object. So the equality on them is undecidable and one can not define functions by cases. A positive aspect is that we are actually proving the correctness of a root-finding algorithm. Details of the proof can be found in [125].

We did not proceed by constructing the reals in Coq, but by axiomatic reasoning. So we have defined the axioms of the real numbers in Coq. As a matter of fact, we have proceeded even more generally by first defining an algebraic hierarchy (semi-groups, monoids, groups, rings, fields, ordered fields); see [124]. The advantages of this approach are: reuse of proven results and reuse of notation. (The reals and complex numbers are fields and the polynomials form a ring.) Then we have defined $\mathbb{R}$ to be an (arbitrary) Cauchy-complete Archimedean ordered field and given such an $\mathbb{R}$, we define $\mathbb{C} := \mathbb{R} \times \mathbb{R}$. To make sure that our axioms for $\mathbb{R}$ make sense, a concrete instantiation for $\mathbb{R}$ has been constructed by Niqui.

Completely formalized in the theorem prover Coq, the proof and theory development amounts to the following. This is the size of our input files (definitions, lemmas, tactic scripts)

Sets and Basics	41 kb
Algebra (upto Ordered Fields)	165 kb
Reals	52 kb
Polynomials	113 kb
Real-valued functions / Basic Analysis	30 kb
Complex numbers	98 kb
FTA proof	70 kb
Construction of $\mathbb{R}$ (Niqui)	309 kb
Rational Tactic	49 kb

To modularize the proof and in order to create a real "library", we have first defined an algebraic hierarchy in the FTA project. In proving FTA, we have to deal with real numbers, complex numbers and polynomials and many of the properties we use are generic and algebraic. To be able to reuse results (also for future developments) we have defined a hierarchy of algebraic structures. The basic level consists of constructive setoids, $\langle A, \#_A, =_A \rangle$, with $A : \mathsf{Set}$, $\#_A$ an apartness and $=_A$ an equivalence relation. (Classically, apartness is just the negation of equality, but constructively, apartness is more 'primitive' than equality and equality is usually defined as the negation of apartness. To understand this, think of two reals x and y as (infinite) Cauchy sequences: we may determine in a finite amount of time whether $x \# y$, but we can in general never know in a finite amount of time that $x = y$.) On the next level we have semi-groups, $\langle S, + \rangle$, with S a setoid and $+$ an associative binary operation on S.

Inside the algebraic hierarchy we have 'inheritance via coercions'. We have the following coercions.

```
   OrdField >-> Field >-> Ring >-> Group
                Group >-> Monoid >-> Semi_grp >-> Setoid
```

This means that all properties of groups are inherited by rings, fields, etc. Also notation is inherited: `x[+]y` denotes the addition of `x` and `y` for `x,y:G` from any semi-group (or monoid, group, ring,...) `G`. The coercions must form a tree, so there is no real multiple inheritance, e.g. it is not possible to define rings in such a way that it inherits both from its additive group and its multiplicative monoid.

In the proof of FTA we needed proofs of equalities between rational expressions such as

$$\frac{1}{x+y}+\frac{1}{x-y}=\frac{2x}{x^2-y^2}$$

These are obtained by 'partial reflection'. Following the reflection method: we define $[\![-]\!] : E \to \mathbb{R}$ with E the type of rational expressions. So E contains a constructor `erecip :  E -> E`
But in the case of rational expressions, the $[\![-]\!]$ can not be total, so we have a so-called 'partial reflection'.

The axioms for real numbers are (apart from the fact that the reals form a constructive ordered field)

- All Cauchy sequences have a limit:

$$\begin{aligned}\text{SeqLim} \quad &: \quad (\Sigma g{:}\text{nat}{\to}F.\text{Cauchy}\, g) \to F \\[1em] \text{CauchyProp} \quad &: \quad \forall g{:}\text{nat}{\to}F.(\text{Cauchy}\, g) \to \\ & \quad\quad \forall \epsilon{:}F_{>0}.\exists N{:}\mathbb{N}.\forall m \geq N.(|g_m - (\text{SeqLim}\, g)| < \epsilon)\end{aligned}$$

- Axiom of Archimedes: (there are no non-standard elements)

$$\forall x{:}F.\exists n{:}\mathbb{N}(n > x)$$

The axiom of Archimedes proves that 'ϵ-Cauchy sequences' and '$\frac{1}{k}$-Cauchy sequences' coincide (similar for limits):
Viz: g : nat $\to F$ is a $\frac{1}{k}$-Cauchy sequence if

$$\forall k{:}\mathbb{N}.\exists N{:}\mathbb{N}.\forall m \geq N(|g_m - g_N| < \frac{1}{k+1})$$

To be sure that our axioms can be satisfied, we have also constructed a real Number structure via the standard tehnique of taking the Cauchy sequences of rational numbers and defining an appropriate apartness on them. It turns out (as was to be expected) that real number structures are categorical: Any two real numbers tructures are isomorphic. This fact has be proved within Coq.

In conclusion we have found that real mathematics, involving both a bit of algebra and a bit of analysis can be formalized completely within a theorem prover (Coq). Setting up a basic library and some good proof automation procedures is a substantial part of the work. An important issue remains how to present the development (and the proof). In the formalization process, the connection with the LaTeX file has been abandoned. We believe that it is essential to provide a system in which one can write the formalization and the documentation. We have attempted to extract an algorithm from the proof, but that turned out to be very difficult, because all the coercions get cluttered up in the extracted program. This will be further investigated. It should be noted that computationally, the behaviour of the root-finding algorithm depends mainly on the representation of the reals.

Involutive Bases

Involutive bases are a special kind of Gröbner bases with additional combinatorial properties that make them very useful for many applications (see [89]). They exist in many polynomial algebras (also non-commutative ones) including ordinary polynomials and linear differential or difference operators. They are thus a possible approach to investigate symbolic solutions to system of (partial) differential equations. An INTAS project coordinated by UKA was devoted to this topic. It terminated in April 2002 (see [85]) and some of the results obtained by UKA are also relevant for Calculemus. On the theoretical side, numerous results on the relations between different kinds of involutive bases, Grbner bases and characteristic sets have been obtained both for ordinary and for differential ideals. Several characterisation theorems for involutive bases have been proved and the computation of (differential) dimension polynomials has been studied. We have thoroughly investigated the homological approach to involution via Spencer cohomology. An algebraic algorithm for the geometric completion to involution was developed (including a constructive solution of the problem of delta-regularity.

Exploration in Finite Algebra

This work was a case study on combining proof planning with computer algebra systems. Its goal was to show that various human-oriented proving techniques can be realized with a multi-strategy proof planner and that the search space of the proof planner can be drastically reduced by employing computations of computer algebra systems during the planning process.

The case study essentially consisted of three parts: (1) To implement a set of proof planning strategies that realize different proof techniques for the residue class domain. Thereby we were interested in examining basic algebraic properties of given residue class structures to classify them into terms of the algebraic structure they form. Furthermore, structures of the same type and cardinality are then classified into sets of isomorphic structures. The implemented proof planning strategies employ to a varying degree the computations of CASs to ease the planning process. (2) To test the effectiveness of the implemented machinery we conducted a large number of experiments by automatically and systematically classifying residue class structures in terms of their basic algebraic properties and into different isomorphism classes. And (3) to verify the usefulness of the combination of proof planning and computer algebra we also compared our approach with alternative techniques. In particular, we experimented with substituting computer algebra by model generation and by proving theorems with a first order equational theorem prover instead of a proof planner. The former turned out to be quite effective and can fruitfully complement the use of computer algebra. The latter proved to be applicable for constructing most of the required proofs but is less robust in a large case study than our combined proof planning and computer algebra approach.

The case study was conducted in the ΩMEGA system, using its proof planner Multi and the CAS MAPLE [227] and GAP [123]. Major parts of the work has been carried out in collaboration of the Saarbrücken and Birmingham nodes and have involved the YVR Martin Pollet. Part (1) and (2) of the case study were reported in [195] and [193], where the former was concerned with proofs of simple algebraic properties and the latter with the isomorphism proofs. An extensive report on both, including a detailed presentation of the constructed proofs has also been published in [192]. Part (3) of the case study has been presented in [194].

The Residue Class Domain

We define a residue class set over the integers as the set of all congruence classes modulo an integer n, i.e., $\mathbb{Z}_n$, or as an arbitrary subset of $\mathbb{Z}_n$. More concretely, we are dealing with sets of the form $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_3 \backslash \{\bar{1}_3\}, \mathbb{Z}_5 \backslash \{\bar{0}_5\}$, $\{\bar{1}_6, \bar{3}_6, \bar{5}_6\}$, etc. where $\bar{1}_6$ denotes the congruence class of 1 modulo 6. If c is an integer we also write $cl_n(c)$ for the congruence class of c modulo n. A binary operation $\circ$ on a residue class set is given in λ-function notation, and $\circ$ can be of the form $\lambda xy_{\bullet}\, x$, $\lambda xy_{\bullet}\, y$, $\lambda xy_{\bullet}\, c$ where c is a constant congruence class (e.g., $\bar{1}_3$), $\lambda xy_{\bullet}\, x \bar{+} y$, $\lambda xy_{\bullet}\, x \bar{*} y$, $\lambda xy_{\bullet}\, x \bar{-} y$, where $\bar{+}$, $\bar{*}$, $\bar{-}$ denote addition, multiplication, and subtraction of congruence classes over the integers, respectively. Furthermore, $\circ$ can be any combination of the basic operations with respect to a common modulo factor, e.g., $\lambda xy_{\bullet}\, (x \bar{+} \bar{1}_3) \bar{-} (y \bar{+} \bar{2}_3)$. We often abbreviate the operations $\lambda xy_{\bullet}\, x \bar{+} y$, $\lambda xy_{\bullet}\, x \bar{*} y$ and $\lambda xy_{\bullet}\, x \bar{-} y$ by $\bar{+}$, $\bar{*}$ and $\bar{-}$, respectively.

For a single structure $(RS_n, \circ)$ we are interested in what kind of algebraic structure it forms, i.e. whether it is a group, a monoid, a semigroup, etc., by showing simple algebraic properties such as associativity, existence of inverses, and so on. For two given structures $(RS^1_n, \circ^1)$ and $(RS^2_m, \circ^2)$ we examine whether or not they are isomorphic; that is, we determine whether or not there is a function $h{:}(RS^1_n, \circ^1) \to (RS^2_m, \circ^2)$ such that h is injective, surjective, and homomorphic.[18] Both for showing simple properties and isomorphism/non-isomorphism proofs we employ proof planning guided by computer algebra computation. In particular for isomorphism and non-isomorphism proofs, the appropriate guidance is crucial for success.

Checking Simple Properties

First, we are interested in classifying residue class sets over the integers together with given binary operations in terms of what algebraic structure they form. We automatically classify structures of the form $(RS_n, \circ)$ in terms of magma (also called groupoid), semi-group, monoid, quasi-group, loop, group, and whether or not they are Abelian. The classification is done by first checking successively if the properties: closure, associativity, existence of the unit element, existence of inverse elements, and the quasi-group axiom (i.e., that for each two elements $a, b \in RS_n$ there exist elements $x, y \in RS_n$ such that $a \circ x = b$ and $y \circ a = b$) hold and then constructing and discharging an appropriate proof obligation. The properties are checked mainly with GAP and proofs for the constructed obligations are planned with Multi. For instance, GAP is used to check whether a given structure contains a unit element; depending on the result, a proof obligation is constructed stating there exists or there does not exist a unit element in the structure. Multi then tries to produce a proof plan for this statement. If it succeeds the next property is checked; if it fails Multi tries to prove the negation. For discharging proof obligations we have implemented three different proving techniques with strategies in Multi, which use symbolic computations to a varying degree.

Exhaustive case analysis. This technique is possible since we are in a finite domain and can always enumerate all occurring cases. The planning process usually starts with the expansion of defined concepts such as *unit*, *associative*, etc. For resulting universally quantified goals ranging over a residue class a case split on all elements of the structure is performed. For existentially quantified goals all possible instantiations for the quantified variable are successively checked. The latter is done by introducing a meta-variable that is bound successively to the different elements of the residue class set. Here the search space can be reduced by computing the (probably) correct instantiation immediately as a hint with a computer algebra system. For instance, when showing that for each element in the structure there indeed exists an inverse, GAP can compute the respective inverses. When the subsequent subgoals cannot be proved, Multi backtracks to the instantiation of the meta-variable and chooses the next element. After the quantifiers are eliminated, the statements about residue classes are transformed to statements about integers which can be solved by numerical simplifications.

Equational reasoning. This approach tries to construct the proofs by using as much as possible general equation solving. Problems are decomposed to the level of equations on integers; universally quantified variables are replaced by constants and existentially quantified variables by meta-variables. The property then holds, when all equations can be solved by the CAS MAPLE to check the universal validity of the equation or, in case the equation contains meta-variables, if there is an instantiation of these meta-variables, such that the equation is universally valid. For instance, the equation for the inverse element $cl_n(c)\bar{+}cl_n(mv) = \bar{0}_n$ containing congruence classes (where c is a constant and mv is a meta-variable) is reduced to the corresponding equation on integers $(c + mv)\, mod\, n = 0\, mod\, n$ before MAPLE returns a general solution for mv.

Applying known theorems. For this technique the planner uses theorems from ΩMEGA's knowledge-base to reduce a given problem. This strategy does not depend on the support of a computer algebra system. Moreover, the theorems are applied to statements about residue class structures directly; a reduction to statements about integers is not necessary.

[18]Observe that we avoid confusion between indices and modulo factors by writing indices as superscripts, except in indexed variables such as x_i, y_j as they are clearly distinct from congruence classes of the form $cl_i(x)$.

Techniques for Isomorphism Proofs

In order to prove that two given structures are isomorphic the proof planner can reuse the three strategies developed for checking simple algebraic properties. But unlike those proofs that could be solved in most cases within one strategy, for isomorphism proofs different subproofs can be solved by different strategies.

Exhaustive case analysis. When constructing an isomorphism proof we have to search for a bijective homomorphism h among all existing mappings between the two residue class structures involved. The mapping h is represented as a pointwise defined function, where the image of each element of the domain is explicitely specified as an element of the codomain. The search can be abbreviated by computing a pointwise isomorphism with MAPLE.

As an example consider the proof that $(\mathbb{Z}_2, \bar{+})$ and $(\mathbb{Z}_2, \lambda xy_{\bullet} x \bar{+} y \bar{+} \bar{1}_2)$ are isomorphic. There exist 4 possible pointwise functions $h : \mathbb{Z}_2 \longrightarrow \mathbb{Z}_2$. MAPLE computes as function $h(\bar{0}_2) = \bar{1}_2, h(\bar{1}_2) = \bar{0}_2$, which is used to subsequently show the properties injectivity, surjectivity, and homomorphy with an exhaustive case analysis. Each of the subproofs has the complexity n^2 where n is the cardinality of the structures involved.[19] However, if no suitable hint can be computed there are n^n pointwise defined functions to check. This becomes infeasible already for relatively small n.

Equational reasoning. Isomorphism proofs can often be simplified by computing a polynomial that interpolates the pointwise defined isomorphic mapping. If an interpolation polynomial can be computed it is introduced into the proof instead of the pointwise mapping. For the construction of the interpolation polynomial from a given pointwise function we employ again MAPLE.

For the example problem that $(\mathbb{Z}_2, \bar{+})$ is isomorphic to $(\mathbb{Z}_2, \lambda xy_{\bullet} x \bar{+} y \bar{+} \bar{1}_2)$ the corresponding pointwise isomorphism mapping is $h(\bar{0}_2) = \bar{1}_2, h(\bar{1}_2) = \bar{0}_2$. MAPLE computes the interpolation polynomial $x \rightarrow (x + 1 \ mod \ 2)$ which is introduced into the proof. Multi now has a chance to find the subproofs for surjectivity and the homomorphism property by equational reasoning, i.e. by reducing these two subproblems to equations, which might be solvable by MAPLE. However, in the subproof for injectivity we have to show for each two distinct elements that their images differ, which cannot be concluded by equational reasoning.

Applying known theorems. Like for the proofs of simple algebraic properties this strategy can be applied to the overall problem directly. Moreover, it can also be applied during the proof of one of the injectivity, surjectivity, or homomorphy subgoals. In particular, it is used to exploit the simple mathematical fact that in finite domains surjectivity implies injectivity and vice versa. Usually Multi proves first the surjectivity subgoal; then the injectivity subgoal is shown by applying the following theorem: *A surjective mapping between two finite sets with the same cardinality is injective.*

Techniques for Non-Isomorphism Proofs

In our previous work [193, 194], we have implemented several proof techniques for the proof planner Multi to show that two structures are not isomorphic. These require varying degrees of guidance from computer algebra or model generation:

Testing all possible functions h. Essentially this corresponds to a case split on all possible instantiations for the mapping h and showing in each case that h is not an isomorphism. While this technique does not require any guidance for Multi, for two structures whose sets have cardinality n, Multi has to consider n^n possible functions, which becomes infeasible even for relatively small n.

Proof by contradiction. The idea of this technique is to find a pair of distinct elements in one structure that is always mapped to the same image under each homomorphism h. This shows that there exists no injective h and therefore no isomorphism. For this technique, a prospective pair of elements can be computed

[19]The proof of each of these properties results in formulas with two nested quantifications ranging over sets of cardinality n. This results into n^2 possible cases.

either with the computer algebra system MAPLE or, more reliably, with the SEM model generator [274]. However, even with this guidance, the subsequent proof process is essentially equational theorem proving, and success is not guaranteed.

Using predefined invariants. An intuitive way to show non-isomorphism is to find an invariant property of one structure that the other structure does not exhibit. We have already implemented a proof planning approach for the following predefined invariants: (1) the structures involved are of different cardinality; (2) the structures form different algebraic entities; e.g., one structure is a group while the other is a semigroup; (3) one of the structures contains an element of some order k and no element in the other structure has order k. For structures without a unit element, we can similarly use the order of traces of elements. Multi checks these invariants in this order. To compute both orders and traces of elements, Multi uses the computer algebra system GAP. In the automatic exploration of the residue class domain (see [194]) we usually start with sets of similar algebraic structures of the same cardinality (e.g., quasigroups of order 5). Hence invariant (3) is the only one of relevance, and the predefined criteria are often not sufficient to successfully construct a non-isomorphism proof.

Evaluation

For assessing the effectiveness of the combination of proof planning and computer algebra we compared our approach with alternative techniques. In particular, we experimented with (1) substituting computer algebra by model generation and (2) by proving theorems with a first order equational theorem prover instead of a proof planner.

For (1) we employed the model generator SEM [274] and substituted its computations for all computer algebra computation in the proof planning process. It turned out that in general both approaches are equally robust and do not outperform each other. In fact, the approaches complement each other in some respects: For instance for guiding the non-isomorphism proofs, MAPLE was less effective since it not always returned all possible solutions to a homomorphism equation system. SEM on the other hand provided the necessary answers.

The experiments (2) were performed by using the theorem prover Waldmeister [137]. It proved to be applicable for constructing most of the required proofs (except for isomorphism problems) but is less robust in a large case study than the combined proof planning and computer algebra approach. While the theorem prover has a clear advantage with respect to runtime, producing solutions much faster than the proof planner, it could find proofs for all stated problems. In particular, if structures with larger cardinality were involved the likelihood of failure would grow. Moreover, the problem formulation was rather intricate and unintuitive.

Proving with Invariants

In the case study presented in section 3.5.b it turned out that the most difficult problem was to show that two given structures are non-isomorphic. The proof attempts often failed because the proof planner did not get the appropriate guidance from a computer algebra system or a model generator, which was necessary for the more advanced proving techniques. To overcome this dilemma we generalize the technique of using invariant properties to show non-isomorphism: Given two structures, we construct an appropriate, bespoke discriminant (i.e., an invariant property that only one of the structures exhibits) to show that the structures are not isomorphic. More formally, for two structures S^1 and S^2 we want to find a property P such that $P(S^1) \wedge \neg P(S^2)$ holds.

For example, consider the pairwise non-isomorphic quasigroups S^1, S^2, S^3 given with their respective multiplication tables in Fig. 22. When comparing the tables of S^1 and S^2, one discriminant is fairly obvious: while S^1 has only $\bar{0}_5$ on the main diagonal, all elements on the main diagonal of S^2 are distinct. Thus, the invariant property we can use is $\exists x. \forall y. x = y \circ y$. Things become less obvious when we

$S^1 = (\mathbb{Z}_5, \bar{-})$

S^1	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$
$\bar{0}_5$	$\bar{0}_5$	$\bar{4}_5$	$\bar{3}_5$	$\bar{2}_5$	$\bar{1}_5$
$\bar{1}_5$	$\bar{1}_5$	$\bar{0}_5$	$\bar{4}_5$	$\bar{3}_5$	$\bar{2}_5$
$\bar{2}_5$	$\bar{2}_5$	$\bar{1}_5$	$\bar{0}_5$	$\bar{4}_5$	$\bar{3}_5$
$\bar{3}_5$	$\bar{3}_5$	$\bar{2}_5$	$\bar{1}_5$	$\bar{0}_5$	$\bar{4}_5$
$\bar{4}_5$	$\bar{4}_5$	$\bar{3}_5$	$\bar{2}_5$	$\bar{1}_5$	$\bar{0}_5$

$S^2 = (\mathbb{Z}_5, \lambda xy.(\bar{2}_5 \bar{*} x) \bar{+} y)$

S^2	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$
$\bar{0}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$
$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$
$\bar{2}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$
$\bar{3}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$
$\bar{4}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$

$S^3 = (\mathbb{Z}_5, \lambda xy.(\bar{3}_5 \bar{*} x) \bar{+} y)$

S^3	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$
$\bar{0}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$
$\bar{1}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$
$\bar{2}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$
$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$	$\bar{2}_5$	$\bar{3}_5$
$\bar{4}_5$	$\bar{2}_5$	$\bar{3}_5$	$\bar{4}_5$	$\bar{0}_5$	$\bar{1}_5$

Figure 22: Some quasigroup multiplication tables

compare the multiplication tables of S^2 and S^3. Here, one invariant of S^3, which does not hold for S^2, is $\forall x. \forall y. (x \circ x = y) \Rightarrow (y \circ y = x)$.

The generalized proof procedure is as follows: given two structures S^1 and S^2 we have to:

1. find an appropriate discriminant P,
2. show that $P(S^1)$ holds,
3. show that $\neg P(S^2)$ holds, and finally
4. show that $\forall X. \forall Y. P(X) \wedge \neg P(Y) \Rightarrow X \not\sim Y$ holds [20].

The single proof parts combine to give the following, sketched formal proof:

$$\dfrac{\dfrac{\overset{\overset{?}{\vdots}\,(2)}{P(S_1)} \qquad \overset{\overset{?}{\vdots}\,(3)}{\neg P(S_2)}}{P(S_1) \wedge \neg P(S_2)}\wedge Intro \qquad \dfrac{\overset{\overset{?}{\vdots}\,(4)}{\forall X. \forall Y. P(X) \wedge \neg P(Y) \Rightarrow X \not\sim Y}}{P(S_1) \wedge \neg P(S_2) \Rightarrow S_1 \not\sim S_2}\forall Elim(S_1, S_2)}{S_1 \not\sim S_2} ModusPonens$$

The proof strategy is realized in Multi with the help of various external systems: To automatically detect the discriminant P we employ the HR system [107]. HR performs automated theory formation by inventing concepts, making conjectures, proving theorems, and finding counterexamples. The main functionality used for the application for finding discriminants discussed here is concept formation, which is achieved by using production rules which take one (or two) old concepts as input and return a new concept.

Discriminants computed by HR are translated into appropriate concepts and provided to the proof planner. The remainder of the proof is subsequently constructed as follows: Subproofs (2) and (3) are planned with Multi possibly using the support of a computer algebra system. The latter depends on the actual strategy Multi choses.

Subproof (4) on the other hand is contributed by first-order automated theorem provers (ATPs), which are called via the TRAMP-system [191], an interface and transformation system. It transforms a given problem into the input formats of connected ATPs and runs these systems concurrently. Output of the ATPs is then translated back into natural deduction (ND) proofs and inserted as proof for the original subproblem. For our work we employed first order resolution provers OTTER, BLIKSEM, and SPASS.

This work has been carried out in collaboration of the nodes in Birmingham, Edinburgh, and Saarbrücken. It particularly involved the YVR Simon Colton, the author of the HR system. The results where published in [196].

[20]While step 4 is fairly obvious for a human mathematician, it is crucial for a formal proof.

The Jordan Curve Theorem for Special Polygons

The proof of *Jordan curve theorem* for special polygons is the first part of the formalization of general *Jordan curve theorem* for simple closed curves:

> **Jordan curve theorem**
> *The theorem that states that every simple closed curve divides a plane into two parts and is the common boundary between them.* (see [217]).

This theorem seems quite obvious, however it is common knowledge that it is very difficult to prove it rigorously. MIZAR formalization follows the script [248] by Y. Nakamura and Y. Takeuchi. The work actually started in 1992 with the article [208] by Y. Nakamura and J. Kotowicz in which the Jordan property was introduced. That article was preceded by several other articles in which Euclidean spaces and *special* polygonal arcs were defined. A polygonal curve is called *special* if its line segments are parallel to the axes. Another useful concept of so-called *Go-boards* was also defined before the submission of [208]. By a *Go-board* the authors ment a matrix of points of the plane as below

$$\begin{bmatrix} (x_1, y_1) & \dots & (x_1, y_n) \\ \vdots & \ddots & \vdots \\ (x_n, y_1) & \dots & (x_n, y_n) \end{bmatrix}$$

with the property that ordinates of points in the same column are equal as well as abscissae of points in rows and, moreover, $x_i < x_j$ and $y_i < y_j$ for $i < j$. Using the techniques of *Go-boards* the following theorem was proved in [163]:

> *Every two special arcs lying in a rectangle R such that the first arc joins the upper and lower sides of R and the second arc joins the left and right sides of R have a non empty intersection.*

Together with several subsequent articles devoted to further development of the theory of *Go-boards*, the above lemma made it possible to complete the first part of the Jordan theorem (saying that the complement of the curve is the union of two components) in [209]. Later, the second part (stating that these components are different) was proved in [255], and finally, the complete theorem was proved in [160].
The preliminary work on the proof of general *Jordan curve theorem* started with defining the external (so-called *Cages*) and internal (so-called *Spans*) approximation of the curve by special polygons in [84] and [254] respectively. Recently, we work on proofs of lemmas left to complete the whole proof. 74 articles devoted to this theorem have been collected so far, which makes about 10% of the Mizar Mathematical Library. However, some of the articles contain suplementary theory, only indirectly relevant to the proof.

Continuous Lattices

One of the largest concentrated efforts in developing MML has been the formalization of *A Compendium of Continuous Lattices* by G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove and D.S. Scott [126] in its entirety. This project started in 1996. The effort was originally motivated by the question of whether or not the MIZAR system was sufficiently developed for the task of expressing advanced mathematics[21]. The current state of the formalization—57 MIZAR articles written by 16 authors—indicates that in principle the MIZAR system has successfully met the challenge.

More detailed reports on the project may be found in [29], [30], and [31].

The formalization was divided into two series of MIZAR articles:

`YELLOW`: articles bridging the gap between the contents of MML and the knowledge assumed in [126],

[21]The above question was raised at the 2nd QED Workshop held in Warsaw in 1995 `http://www-unix.mcs.anl.gov/qed`

WAYBEL[22]: articles formalizing the main course of [126].

No formalization of examples and exercises were done unless some item in the main text depends on it. This was meant to reduce the workload as none of the participants specialized in continuous lattices.

The formalization is as close as possible to [126], but provisions were made for some MIZAR peculiarities such as built-in concepts and mechanisms, reuse of MML, and the like.

Whenever possible, the formalization was more general than [126].

	[126]	Formalized	The de Bruijn factor
Characters (bytes)	327,929	3,098,460	9.44
Compressed	100,839	566,720	5.62
Tokens (words)	55,142	808,020	14.65

Table 2: The de Bruijn factors

The book [126] contains 334 pages divided into 8 chapters covering a total of 715 items. Of these, 254 items are examples and exercises which we did not plan to cover. By the end of April 2002, the formalization covered 231 items which is slightly more than 50% of the work originally planned. On average, an article in the WAYBEL series covers 7 items, varying from 1 to 18 with a median of 5.

	1996	1997	1998	1999	2000	2001	Total
YELLOW	8	1	5	3	1	4	22
WAYBEL	10	6	8	4	5	2	35
All	18	7	13	7	6	6	57

Table 3: Articles over the years

Table 3 summarizes the number of articles submitted to MML from this project. The YELLOW series constitutes 38.6% of the articles, much less than originally anticipated. However, this may change in the near future as we are approaching the part of [126] which is poorly covered in MML.

Order Sorted Algebras

The theory of *order sorted algebras* (a concept originally introduced by Joseph Goguen) was developed in MIZAR by Josef Urban from Charles University (Prague) during his visit to University of Bialystok as YVR of CALCULEMUS. This work consists of 5 MIZAR articles covering basic concepts of the theory. The notion of a *signature* of an *order sorted algebra* introduced in [257] is a 5-tuple $<\mathcal{C}, \mathcal{O}, \mathcal{E}, \alpha, \rho>$, where $\mathcal{C}$ is the set of sort symbols, $\mathcal{O}$ is the set of operation symbols, $\mathcal{E}$ is an equivalence relation of $\mathcal{O}$, $\rho : \mathcal{O} \to \mathcal{C}$ and $\alpha : \mathcal{O} \to \mathcal{C}^*$ ($\mathcal{C}^*$ denotes the free monoid generated by $\mathcal{C}$). *Order sorted algebra* was defined using the structure of a *many-sorted algebra* previously introduced in [253]. A *many sorted algebra* (over a given *signature*) is a 2-tuple of the form $<\mathcal{S}, \chi>$, where $\mathcal{S}$ is a function which assigns some sort to any sort symbol and χ is the characteristic of the algebra which assigns some operation given by the *signature* to every operation symbol. The development of the theory of *many sorted algebras* covers the concepts of subalgebras, quotient algebras and homomorphisms between algebras. Eventually, *free order sorted algebras* were defined in [256].

3.5.c Ongoing and Future Work

In the remainder of the project we intend to tackle the following challenging problems and anticipate the following collaborations.

[22]The *way below* relation is the key concept in continuous lattices, see [126, p. 38].

Involutive Bases

A work in progress is to investigate the impact of involutive techniques in field theory, a domain of Physics. Most if not all physical systems are represented by systems of partial differential equations. Among such systems are the well-known Yang-Mills or Einstein equations for instance. Without aiming at doing better than what the very many expert physicists of string theory are doing, it is possible to study whether some systems are integrable or not. What is challenging is to solve symbolically over- or under-determined systems of polynomial or differential equations or in simpler terms to extend the concept of Groebner bases to such systems. This is an old well-known problem that was much earlier investigated by Cartan and his co-workers before being put aside. The need to design constructive methods in mechanized mathematics was at the origin of a revival. But, we still need to find out the proper representations in which to better formulate involutive bases. Again in very simple words, we are in a situation where we can get some information on local solutions of non-linear systems and we aim at extending them to some kind of non-local neighborhood.

Proofs in Homological Algebra

In the course of the Calculemus project the Basic Perturbation Lemma was identified as an interesting challenge problem, since its field is the Algebraic Topology and the role of the infinity in this field. In the particular case in which the application domain of a symbolic computation system is Homological Algebra or Algebraic Topology the analysis of the correctness of its algorithms, using as a tool an ATP, is particularly complex, due to the need of using infinite data structures and, then, higher-order functional programming [235], [233]. This specific situation implies that there is a deep semantic gap between the proofs which appear in the standard literature on Algebraic Topology and the semantics of the implementation language used to build the symbolic computation system. Our aim is to bridge this gap by using ATP technology.

Kenzo (Sergeraert et al. [117]) is a Common Lisp symbolic computation system for Algebraic Topology. Kenzo has computed homology groups unknown until its construction [233]. Hence, the following objective defines an interesting research task.

Goal 1. Give a proof of the Kenzo correctness.

This goal being very complex (due both to the size and conceptual complexities of Kenzo), we derive the following sequence of subgoals.

Subgoal 1.1. Verify and establish formal models for Kenzo *fragments*.

Subgoal 1.1.1. Give automated *certified* versions of some central parts of the program.

Since the Basic Perturbation Lemma (BPL) is the most important tool in algorithmic homological algebra, this is sensible to state as a first task:

Subgoal 1.1.1.1. Give a certified version of the BPL implementation used in Kenzo.

Here, it is necessary to choose a theorem prover to make the automated proof of correctness. Due to the previous work done in Isabelle [211] about algebraic structures and its expresiveness, this has been our choice. But linking Common Lisp (the Kenzo implementation programming language) and Isabelle seems difficult from a technical point of view. So, we prefer starting with a BPL implementation in ML (the Isabelle implementation programming language).

Subgoal 1.1.1.1.1. Implement in ML a certified version of the BPL algorithm.

Termination problems cause difficulties to the automated proving process, so in a first step, we intended to give an Isabelle automated proof of the BPL *theorem*. This is the problem tackled in this extended abstract.

Subgoal 1.1.1.1.1.1. Give an Isabelle automated proof of the BPL theorem.

In the following definitions, some notions of homological algebra are briefly introduced (for details, see

[173] for instance).

Definition 2. A *graded group* C_* is a family of abelian groups indexed by the integer numbers: $C_* = \{C_n\}_{n\in\mathbb{Z}}$, with each C_n an abelian group. A *graded group morphism* $f : A_* \to B_*$ *of degree k* ($\in \mathbb{Z}$) between two graded groups A_* and B_* is a family of group homomorphisms: $f = \{f_n\}_{n\in\mathbb{Z}}$, with $f_n : A_n \to B_{n+k}$ group homomorphism $\forall n \in \mathbb{Z}$. A *chain complex* is a pair (C_*, d_{C_*}), where C_* is a graded group, and d_{C_*} (*the differential map*) is a graded group homomorphism $d_{C_*} : C_* \to C_*$ of degree -1 such that $d_{C_*}d_{C_*} = 0$. A *chain complex homomorphism* $f : (A_*, d_{A_*}) \to (B_*, d_{B_*})$ between two chain complexes (A_*, d_{A_*}) and (B_*, d_{B_*}) is a graded group homomorphism $f : A_* \to B_*$ (degree 0) such that $fd_{A_*} = d_{B_*}f$.

Definition 3. A *reduction* $D_* \Rightarrow C_*$ between two chain complexes is a triple (f, g, h) where: (a) The components f and g are chain complex morphisms $f : D_* \to C_*$ and $g : C_* \to D_*$; (b) The component h is a homotopy operator on D_*, that is to say: a graded group homomorphism $h : D_* \to D_*$ of degree +1; (c) The following relations are satisfied: (1) $fg = \mathrm{id}_{C_*}$; (2) $gf + d_{D_*}h + hd_{D_*} = \mathrm{id}_{D_*}$; (3) $fh = 0$; (4) $hg = 0$; (5) $hh = 0$.

Definition 4. Let D_* be a chain complex. A *perturbation* of the differential d_{D_*} is a morphism of graded groups $\delta_{D_*} : D_* \to D_*$ (degree -1) such that $d_{D_*} + \delta_{D_*}$ is a differential for the underlying graded group of D_*. A perturbation δ_{D_*} of d_{D_*} satisfies the *nilpotency condition*, with respect to a reduction $(f, g, h) : D_* \Rightarrow C_*$, if the composition $\delta_{D_*} \circ h$ is pointwise nilpotent, that is, $(\delta_{D_*} \circ h)^n(x) = 0$ for an $n \in \mathbb{N}$ depending on each x in D_*.

Theorem 5. ***Basic Perturbation Lemma*** — Let $(f, g, h) : D_* \Rightarrow C_*$ be a chain complex reduction and $\delta_{D_*} : D_* \to D_*$ a *perturbation* of the differential d_{D_*} satisfying the nilpotency condition with respect to the reduction (f, g, h). Then a new reduction $(f', g', h') : D'_* \Rightarrow C'_*$ can be obtained where the underlying graded groups of D_* and D'_* (resp. C_* and C'_*) are the same, but the differentials are perturbed: $d_{D'_*} = d_{D_*} + \delta_{D_*}, d_{C'_*} = d_{C_*} + \delta_{C_*}$, and $\delta_{C_*} = f\phi\delta_{D_*}g$; $f' = f\phi$; $g' = (1 - h\phi\delta_{D_*})g$; $h' = h\phi$, where $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*} \circ h)^i$.

The BPL is a central result in algorithmic homological algebra (in particular, it has been intensively used in the symbolic computation systems EAT [234] and Kenzo [117]). It first appeared in [236] and it was rewritten in modern terms in [56]. Since then, plenty of proofs have been described in literature (see, for instance, [130], [35], [232]).

Work on this project is currently under way at the Universidad de La Rioja (Dr. Julio Rubio and Jesús Aransay) in collaboration with Dr. Clemens Ballarin at TU München, where the formalization of proofs of some auxiliary lemmas in the theorem prover Isabelle has been completed.

Proofs in Graph Theory

The goal of this case study is to generate non-isomorphism proofs in graph theory, i.e. to show that two given graphs are not isomorphic. To answer this question there exist currently elaborate computational techniques as, for instance, implemented in the Nauty system [190]. Unfortunately, these systems usually only give an answer but no detailed justifications why the two graphs are not isomorphic. However, a working mathematician might be interested in more insights into the problem, not only to check the correctness, but also to put the additional information to further use. To further this end we want to employ the technique we have developed for task 1.1 during the integration of computer algebra into proof planning as well as the mechanisms implemented for the case study described in section 3.5.b.

The overall idea is to prove that two given graphs Γ_1 and Γ_2 are not isomorphic, by showing that there respective automorphism groups are not isomorphic. So far we have defined a set of eight problems in permutation groups that will part of the more general solution:

In computational permutation group theory, a group G is specified by a list of generating permutations $A = [a_1, a_2, \ldots, a_k]$, where A consists of permutations of the points $\Omega := \{1, 2, \ldots, n\}$, i.e. the elements of A belong to the symmetric group Sym_n. We often write $G = \langle A\rangle$ to denote that G is generated by A.

We let our permutation act on points from the right.

1. **Membership** The first question is how to prove that a permutation g belongs to the group G. We define a *word* in A to be an expression of the form

   $$a_{i_1}^{e_1} a_{i_2}^{e_2} \cdots a_{i_m}^{e_m}$$

   where the indices i_j are in the range $1, \ldots, k$ and the exponents e_j are integers. It is now easily shown that a permutation $g \in \mathrm{Sym}_n$ is an element of G if, and only if, it can be expressed as a word in A.

2. **Subgroup** Suppose H is another permutation group with generating set B and that we wish to prove that H is a subgroup of G. From the definition of a generating set it follows that H is a subgroup of G if, and only if, every element of B is contained in G.

3. **Orbit** The *orbit* of x under the action of G is $xG = \{xg : g \in G\}$. We wish to determine an orbit containing a given point of Ω.

4. **Schreier tree** Stabiliser subgroups are of fundamental importance to both theoretical and computational permutation group theory. The *stabiliser subgroup* in G of x is

   $$G_x = \{g \in G : xg = x\}.$$

   It is not immediately clear how to compute with this subgroup, for, although the definition gives us a test for whether g is an element of G_x, it does not give us a generating set.

   The following *Orbit Lemma* establishes a one-to-one correspondence between the orbit of a point and the set of cosets of its stabiliser.

   If $y \in xG$, then $\{g \in G : xg = y\}$ is a coset of G_x. In particular, $|xG| = |G|/|G_x|$.

   Suppose that for every element y of the orbit xG, we choose $t(y) \in G$ with the property that $xt(y) = y$. Then it follows immediately from the Orbit Lemma that all such $t(y)$ form a set of coset representatives for G_x in G. It would be inefficient to store all the elements $t(y)$, so instead we construct a *Schreier tree* rooted at x. That is, a subgraph $\mathcal{T}$ of the orbit graph $\mathcal{G}$ containing x which is a tree (when we view the edges as being undirected) with root x. For every $y \in X$, there is a unique minimal path in $\mathcal{T}$ from x to y (again, disregarding the fact that the labeled edges are directed).

5. **Stabiliser** Now that we have a set of coset representatives for G_x, we can use it to compute a generating set for the stabiliser of x in G. This query is based upon *Schreier's lemma*:

   Suppose G is a group with generating set A, and H is a subgroup of G. If U is a set of coset representatives for H in G, and the function $t : G \to U$ maps an element g of G to the representative of Hg, then a generating set for H is given by

   $$\left\{ua\, t(ua)^{-1} : u \in U, a \in A\right\}.$$

   Observe that for $x \in \Omega$ and $H = G_x$, the function $t : G \to U$ does not depend on the choice of element in a coset Hg, so the map $t : \Omega \to G$ given by $t(xg) = t(g)$, is well defined. This indicates how to apply Schreier's Lemma to permutation groups.

6. **Base** Now that we have a stabiliser of a subgroup, we can repeat the process to form a chain of subgroups. A *base* for G is a finite sequence $B = [x_1, \ldots, x_k]$ of distinct points in Ω such that

   $$G_{x_1, x_2, \ldots, x_k} = 1.$$

   Hence, the only element of G which fixes all of the points $x_1, x_2, \ldots, x_k$ is the identity. Clearly every permutation group has a base, but not all bases for a given group are of the same length. If we write $G^{(i)} = G_{x_1, x_2, \ldots, x_i}$, then we have a *chain of stabilisers*

   $$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(k-1)} \geq G^{(k)} = 1.$$

We often require that a base has the additional property that $G^{(i)} \neq G^{(i+1)}$.

A base can be constructed by starting with $B = [x_1]$, and recursively choosing a letter x_i in a nontrivial $G_{x_1,\ldots,x_{i-1}}$-orbit and appending it to B. The construction is finished when $G_{x_1,\ldots,x_i} = 1$.

7. **Non-membership** Here we deal with the complementary query to the first one treated: Prove that the permutation g does not belong to G. Given a base $B = [x_1, \ldots, x_k]$ of G, we have a chain of subgroups
$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(k-1)} \geq G^{(k)} = 1$$
and sets $U^{(i)}$ consisting of coset representatives for $G^{(i+1)}$ in $G^{(i)}$. For we can take $G^{(i)} = G_{x_1,\ldots,x_{i-1}}$ and $U^{(i)} = t(G^{(i)})$ the set of Schreier elements corresponding to a Schreier tree for $G^{(i)}$ rooted at x_{i-1}.

   An element g of G is contained in exactly one coset of $G^{(1)}$ in $G^{(0)}$, so $g = h_1 u_0$ for some unique h_1 in $G^{(1)}$ and u_0 in $U^{(0)}$. By induction, we can show that $g = u_k u_{k-1} \cdots u_1 u_0$ where each $u_i \in U^{(i)}$ is uniquely determined by g. This process, called *sifting* an element, gives a canonical form for the elements of G and underpins most of the more advanced applications of stabiliser chains.

   On the other hand, if g is not in G, then sifting fails because at some stage we get that $x_i h_{i-1}$ is not in the orbit $x_i G^{(i-1)}$, and so h_{i-1} is not in $G^{(i-1)}$. This gives us our proof of non-membership.

8. **Order** The order of a permutation group can now be effectively computed. We use the following *Order Lemma*:

   Suppose G is a permutation group and $B = [x_1, \ldots, x_k]$ is a base for G. Then
$$|G| \quad = \quad \prod_{i=1}^{k} |x_i G^{(i-1)}|.$$

We have currently implemented specialized algorithms in the computer algebra system Gap to compute the necessary information for the above problems. These computations are now employed in the proof planner of ΩMEGA to derive the necessary proofs for the problems. From this we will work towards non-isomorphism proofs for automorphism groups, which will eventually lead up to the full non-isomorphism proofs for graphs.

The work will be jointly carried out by the nodes in Birmingham, Eindhoven, and Saarbrücken. Thereby the Eindhoven node is responsible for generating the certificates in the computer algebra system Gap, whereas the Birmingham and the Saarbrücken node will be jointly responsible for an adequate formalisation of the problems in an intuitive logical formalism in ΩMEGA, and the proof planning including computer algebra computations. Currently the work involves the YVR Martin Pollet.

Exploration in Zariski Spaces

Zariski Spaces were introduced in 1998 in [189]. As they are a fairly new and barely researched domain, they offer the opportunity to apply some of the techniques developed in the Calculemus Network to uncharted mathematical territory.

Before defining a Zariski Space, we first recall the definition of a Zariski Topology of a ring. Let R be a commutative ring with unity, and let $specR$ denote the collection of prime ideals of R. Now for each (possibly empty) subset A of R, the *variety* of A is given by
$$V(A) = \{P \in specR : A \subseteq P\}.$$

It is easily shown that the collection of all such varieties constitutes (the closed sets of) a topology on $specR$, called the Zariski Topology of R, which we denote by $\zeta(R)$. Furthermore, every topology is clearly a semiring, if one takes the operations of addition and multiplication to be set-theoretic intersection and union, respectively.

Now let M be an R-module, and more or less repeat the above process. That is to say, let $specM$ denote the collection of all prime submodules of M, and for each subset B of M, let the variety of B be given by

$$V(B) = \{P \in specM : B \subseteq P\}.$$

Finally, let $\zeta(M)$ represent the collection of all varieties of subsets of M. Then one can show that $\zeta(M)$ is closed with respect to intersections, but not unions, and therefore does not in general form a topology. However, it does form a semimodule over the semiring $\zeta(R)$, where the additive operation in $\zeta(M)$ is taken to be intersection, and scalar multiplication is given by $V(A)V(B) = V(RAB)$. We call this semimodule $\zeta(M)$ the Zariski Space of M.

We plan to transfer techniques developed for the residue class domain to Zariski Spaces. In particular we intend to perform a classification of concrete, small Zariski spaces, for instance over integer modules automatically using the proof planner Multi, computer algebra systems like Singular and MacAulay2, the model generator SEM and the HR concept formation system.

The work will be jointly carried out by the Birmingham and the Edinburgh node. Since one of the authors of the original Zariski Spaces paper, R. L. McCasland, is now a research fellow with the Edinburgh node, we have the unique opportunity to exploit his indepth knowledge and mathematical know-how for our experiments.

Jordan Curve Theorem

The work on the MIZAR formalization of *Jordan curve theorem* is quite advanced. Only some technical lemmas remain to be proved and we hope to complete the proof for arbitrary simple closed curves during the first part of the year 2003.

Continuous Lattices

We intend to formalize in MIZAR a substantial part of the remaining theory in *A Compendium of Continuous Lattices*, [126]. Although the formalization already includes 57 MIZAR articles, there is still a lot of work to do.

As the first step we will concentrate on the theory of function spaces (spaces of continuous functions equipped with the Scott topology) which seems to be the most difficult topic on the project research frontier. It deals with categorial properties unifying the lattice aspects of function spaces with topological ones. Namely, we must address lattice and topological products, top-lattices of maps preserving some lattice properties and top-lattices of continuous topological maps, and correspondence between them in this topic. It is well advanced, but still some theorems are waiting.

As the second step we will formalize the proof of equivalence of Lawson topology and lim-inf topology. It is required by the theory from Chapter V. This will cover Chapter III. Also, the formalization of Chapters IV and V is started. However, the theory from Chapter V depends on the theorems from the first and second step. It needs equivalence of Lawson and lim-inf topologies and formalized results cannot be submitted yet to the MML. For Chapter IV there were and still are several formalizations required to fill the gaps between knowledge assumed in [126] and the state of the *Mizar Mathematical Library*.

We expect a new edition of the monograph which is supposed to be published by Cambridge University Press in March 2003 as ’Continuous Lattices and Domains’. Therefore, our work will also include a revision of *Mizar Mathematical Library* according to the new approach presented in the new edition. We seek partners in the Calculemus project to cooperate within all the above fields.

Bibliography

[1] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS: Journal of Computer and System Sciences*, 61:236–269, 2000.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *IEEE Real-Time Systems Symposium*, pages 2–11, 1993.

[3] A. Armando and C. Ballarin. Maple's evaluation process as constraint contextual rewriting. In B. Mourrain, editor, *ISSAC 2001: July 22–25, 2001, University of Western Ontario, London, Ontario, Canada: Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, pages 32–37, New York, NY 10036, USA, 2001. ACM Press.

[4] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proc. European Conference on Planning, CP-99*, 1999.

[5] A. Armando, A. Coglio, F. Giunchiglia, and S. Ranise. The Control Layer in Open Mechanized Reasoning Systems: Annotations and Tactics. *Journal of Symbolic Computation*, 32(4), 2001.

[6] A. Armando, L. Compagna, and S. Ranise. System Description: RDL—Rewrite and Decision procedure Laboratory. In *Automated Reasoning. First International Joint Conference (IJCAR'01), Siena, Italy, June 18–23, 2001, Proceedings*, volume 2083 of *LNAI*, pages 663–669, Berlin, 2001. Springer.

[7] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[8] A. Armando and T. Jebelean, editors. *Calculemus 99: International Workshop on Combining Proving and Computation*, volume 23(3) of *Electronic Notes in Theoretical Computer Science*, Trento, Italy, 1999. Elsevier.

[9] A. Armando and T. Jebelean, editors. *Calculemus: Integrating Computation and Deduction*, volume 32 (4) of *Special Issue of Journal of Symbolic Computation on Calculemus'99*, October 2001.

[10] A. Armando, M. Kohlhase, and S. Ranise. Communication Protocols for Mathematical Services based on KQML and OMRS. In Kerber and Kohlhase [150].

[11] A. Armando and S. Ranise. Constraint Contextual Rewriting. In *Proc. of the 2nd Intl. Workshop on First Order Theorem Proving (FTP'98), Vienna (Austria)*, pages 65–75, 1998.

[12] A. Armando and S. Ranise. Termination of Constraint Contextual Rewriting. In Kirchner and Ringeissen [156], pages 47–61.

[13] A. Armando and S. Ranise. A Practical Extension Mechanism for Decision Procedures: the Case Study of Universal Presburger Arithmetic. *J. of Universal Computer Science (Special Issue: Formal Methods and Tools)*, 7(2):124–140, 2001.

[14] A. Armando and S. Ranise. Constraint Contextual Rewriting. *Journal of Symbolic Computation. Special issue on First Order Theorem Proving, P. Baumgartner and H. Zhang editors*, 2002.

[15] A. Armando, S. Ranise, and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In L. Fribourg, editor, *CSL-01: Conference on Computer Science Logic*, volume 2142, pages 513–527, Paris, France, 2001. Springer.

[16] A. Armando, M. Rusinowitch, and S. Stratulat. Incorporating decision procedures in implicit induction. *Journal of Symbolic Computation*, 34:241–258, 2002.

[17] A. Armando and D. Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In *AI*IA-TABOO Joint Workshop: 'Dagli Oggetti agli Agenti: Tendenze Evolutive dei Sistemi Software'*, pages 70–75, Parma, Italy, 2000. Reprinted in AI*IA Notizie Anno XIII (2000) vol. 3.

[18] A. Armando and D. Zini. Interfacing Computer Algebra and Deduction Systems via the Logic Broker Architecture. In Kerber and Kohlhase [150], pages 49–64.

[19] A. Asperti, B. Buchberger, and J. H. Davenport, editors. *Mathematical Knowledge Management, Second International Conference, MKM 2003*, Bertinoro, Italy, February 16-18 2003. Springer.

[20] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In Voronkov [260], pages 195–210.

[21] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Efficiently Integrating Boolean Reasoning and Mathematical Solving, 2002. Submitted to Journal of Symbolic Computation.

[22] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements. In Calmet et al. [87].

[23] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Bounded Model Checking for Timed Systems. In D. A. Peled and M. Y. Vardi, editors, *FORTE 2002: Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, pages 243–259, Houston, Texas, 2002. Springer.

[24] S. Autexier, C. Benzmüller, and D. Hutter. Towards a framework to integrate proof search paradigms. SEKI Report SR-03-02, Fachrichtung Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2003. Submitted to a major international conference.

[25] M. Baaz and A. Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, volume 2514 of *LNAI*, Tblisi, Georgia, 2002. Springer.

[26] L. Bachmair, N. Dershowitz, and D.Plaisted. Completion without failure. In Ait-Kaci and M.Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2 Rewriting Techniques, pages 1–30. Academic Press, New York, 1989.

[27] C. Ballarin and M. Kauers. Solving parameter linear systems: an experiment with constant algebraic programming. In *Proc. Eighth Rhine Workshop on Computer Algebra*, 2002.

[28] A. Balluchi, M. D. Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli. Hybrid control in automotive applications: the cut-off control, 1999.

[29] G. Bancerek. Development of the theory of continuous lattices in MIZAR. In Kerber and Kohlhase [150].

[30] G. Bancerek, N. Endou, and Y. Shidama. Lim-inf convergence and its compactness. *Mechanized Mathematics and Its Applications*, 2(**1**):29–35, 2002.

[31] G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in MIZAR: Formalizing recent mathematics. *Journal of Automated Reasoning*, 29(**3**):189–224, 2002.

[32] H. Barendregt and E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28(3):321–336, 2002.

[33] H. Barendregt and A. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, 32:3–22, 2001.

[34] H. Barendregt and H. Geuvers. *Proof Assistants using Dependent Type Systems*, volume 2 of *Handbook of Automated Reasoning*, chapter 18, pages 1149–1238. Elsevier, 2001.

[35] D. W. Barnes and L. A. Lambe. Fixed point approach to homological perturbation theory. In *Proceedings of the American Mathematical Society*, volume 112, pages 881–892, 1991.

[36] R. G. Bartle and D. R. Sherbert. *Introduction to Real Analysis*. John Wiley and Sons, 3rd edition, 1999.

[37] C. Benzmüller, editor. *Systems for Integrated Computation and Deduction – Interim Report of the Calculemus IHP Network,* Seki Technical Report. Saarland University, 2003. `http://www.ags.uni-sb.de/~chris/papers/E5.pdf`.

[38] C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part I)*, number SR-02-07 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E2.pdf`.

[39] C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part II)*, number SR-02-08 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E3.pdf`.

[40] C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part III)*, number SR-02-09 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E4.pdf`.

[41] C. Benzmüller, C. Giromini, and A. Nonnengart. Symbolic Verification of Hybrid Systems supported by Mathematical Services. In Caprotti and Sorge [99]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.

[42] C. Benzmüller, C. Giromini, A. Nonnengart, and J. Zimmer. Reasoning services in the mathweb-sb for symbolic verification of hybrid systems. In *Proceedings of the Verification Workshop - VERIFY'02 in connection with FLOC 2002*, pages 29–39, Kopenhagen, Denmark, 2002.

[43] C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. An Agent-oriented Approach to Reasoning. In Linton and Sebastiani [175].

[44] C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Experiments with an Agent-oriented Reasoning System. In *KI 2001: Advances in Artificial Intelligence*, Vienna (Austria), 2001.

[45] C. Benzmüller and M. Kerber. A Challenge for Automated Deduction. In *Proceedings of IJCAR-Workshop: Future Directions in Automated Reasoning*, Siena (Italy), 2001.

[46] C. Benzmüller and M. Kerber. A Lost Proof. In *TPHOLs: Work in Progress Papers*, Edinburgh (Scotland), 2001.

[47] C. Benzmüller, A. Meier, and V. Sorge. Distributed assertion retrieval. In *First International Workshop on Mathematical Knowledge Management RISC-Linz*, pages 1–7, Schloss Hagenberg, 2001.

[48] C. Benzmüller, A. Meier, and V. Sorge. Bridging Theorem Proving and Mathematical Knowledge Retrieval. In D. Hutter and W. Stephan, editors, *Festschrift in Honour of Prof. Jörg Siekmann*, LNAI. Springer, 2003. To appear.

[49] C. Benzmüller and V. Sorge. Oants – an open approach at combining interactive and automated theorem proving. In Kerber and Kohlhase [150], pages 81–97.

[50] C. Benzmüller and V. Sorge. Agent-based Theorem Proving. In *9th Workshop on Automated Reasoning*, London (GB), March 2002.

[51] P. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. *Fundamenta Informaticae*, 39(1–2):39–57, 1999.

[52] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, LNCS, pages 193–207. Springer Verlag, 1999.

[53] R. J. Boulton and P. B. Jackson, editors. *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*. Springer, 2001.

[54] A. Bove and V. Capretta. Nested general recursion and partiality in type theory. In Boulton and Jackson [53], pages 121–135.

[55] D. Brickley and R. V. Guha. The Resource Description Framework. W3C Recommendation 1.0, World Wide Web Consortium, 2000. Available at `http://www.w3.org/RDF/`.

[56] R. Brown. The twisted eilenberg-zilber theorem. *Celebrazioni Arch. Secolo XX, Simp. Top.*, pages 34–37, 1967.

[57] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.

[58] B. Buchberger. Theorema: Proving, Solving, Computing. In *Proc. Workspop Logic Programming and Non-Monotonic Reasoning*, Schloss Dagstuhl, July 1997. Invited Talk.

[59] B. Buchberger. Theory Exploration Versus Theorem Proving. In Armando and Jebelean [8], pages 67–69.

[60] B. Buchberger. Computer-assisted proving by the pcs-method. In M. Hazewinkel, editor, *Proceedings of the Workshop on Constructive Algebra*, LNCS. Springer, 2000. To appear.

[61] B. Buchberger. Mathematica and Computer Science: A Personal View. In T. Jebelean and V. Negru, editors, *Proceedings of the Second International Workshop on Symbolic and Numeric Algorithms for Scientific Computing*, 2000. Timisoara, Oct. 4-6.

[62] B. Buchberger. Mathematics and computer science - a personal view. *An. Mat. Univ. Timisoara, seria mat.-informatica*, 24(1):3–18, 2000.

[63] B. Buchberger. Theorema Proving For and With Gröbner Bases. In K. Galkowski and E. Rogers, editors, *Proceedings of the Second International Workshop on Multidimensional (nD) Systems, June 27-30, Czocha Castle, Poland*, pages 15–22, 2000.

[64] B. Buchberger. Gröbner Bases and Systems Theory. *Multimensional Systems and Signal Processing*, 12(3/4):223–253, 2001. Special Issue on Applications of Groebner Bases in Multidimensional Systems and Signal Processing (Z. Lin, L. Xu eds.).

[65] B. Buchberger. Logicographic symbols: A new feature in *Theorema*. In *Symbolic Computation – New Horizons*, pages 23–30. Tokyo Denki University Press, 2001. Proceedings of the 4th International Mathematica Symposium, Tokyo Denki University, Chiba Campus, Japan, June 25-27, 2001.

[66] B. Buchberger. The pcs prover in *Theorema*. In Moreno-Díaz et al. [202], pages 469–478.

[67] B. Buchberger. Theorema: A short introduction. *Mathematica Journal*, 8(2):247–252, 2001.

[68] B. Buchberger. Theorema: Extending mathematica by automated proving. In D. Ungar, editor, *Proceedings of PrimMath 2001 (The Programming System Mathematica in Science, Technology, and Education)*, pages 10–11, University of Zagreb, Electrotechnical and Computer Science Faculty, September 27-28 2001.

[69] B. Buchberger. Theorema and mathematical knowledge management. Invited talk at the IMA 2002 Summer Program: Special Functions in the Digital Age, University of Minnesota, Minneapolis, USA, July 22 - August 2 2002.

[70] B. Buchberger, K. Aigner, C. Dupre, T. Jebelean, F. Kriftner, M. Marin, K. Nakagawa, O. Podisor, E. Tomuta, Y. Usenko, D. Vasaru, and W. Windsteiger. Theorema: An Integrated System for Computation and Deduction in Natural Style. In *Proceedings of CADE 98 (International Conference on Computer Aided Deduction), Lindau, Germany, July 5-10*, 1998. Workshop on integration of proving and computing.

[71] B. Buchberger and O. Caprotti, editors. *MKM 2001 (1st International Workshop on Mathematical Knowledge Management)*, Research Institute for Symbolic Computation, Johannes Kepler University, Hagenberg, September 24-26 2001.

[72] B. Buchberger, C. Dupré, T. Jebelean, K. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The *Theorema* Project: A Progress Report. In Kerber and Kohlhase [150].

[73] B. Buchberger, G. Gonnet, and M. Hazewinkel, editors. *Mathematical Knowledge Management (MKM 2001) – Special issue of Annals in Mathematics and Artificial Intelligence,*. Kluwer, 2003. To appear.

[74] B. Buchberger and T. Jebelean, editors. *The 2nd International Theorema Workshop*. Proceedings published as RISC-Report 98-10, June 29-30 1998. Hagenberg, Austria.

[75] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A survey of the *theorema* project. In W. Kuechlin, editor, *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation*, pages 384–391, Maui, Hawaii, July 1997. ACM Press.

[76] B. Buchberger, T. Jebelean, and D. Vasaru. Theorema: A System for Formal Scientific Training in Natural Language Presentation. In *Proceedings of ED-MEDIA 98 (International Conference on Educational Multimedia),Freiburg, Germany*, pages 174–179, June 20-23 1998.

[77] B. Buchberger and F. Lichtenberger. *Mathematics for Computer Science: The Method of Mathematics*. Springer Heidelberg, 1980. 180 pages.

[78] B. Buchberger and S. Maruster, editors. *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'01)*, August 2001.

[79] B. Buchberger, D. Vasaru, and T. Jebelean. The Theorema System: Current Status and the Proving-Solving-Computing Cycle. In *Proceedings of RTETP (Rewriting Techniques and Efficient Theorem Proving)*, Kiev, June 2000.

[80] B. Buchberger and W. Windsteiger. The Theorema Language: Implementing Object- and Meta-Level Usage of Symbols. In *Proceedings of Calculemus 98, Eindhoven, Netherlands*, 1998.

[81] A. Bundy. The use of explicit plans to guide inductive proofs. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction*, number 310 in LNCS, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag.

[82] A. Bundy. The use of proof plans for normalisation. In R. S. Boyer, editor, *Essays in Honor of Woody Bledsoe*. Kluwer, 1991.

[83] A. Bundy and P. Janičić. A General Setting for Flexibly Combining and Augmenting Decision Procedures. *Journal of Automated Reasoning*, 3(28), 2002.

[84] C. Byliński and M. Żynel. Cages - the external approximation of Jordan's curve. *Formalized Mathematics*, 9(**1**):19–24, 2001.

[85] J. Calmet. Intas: Final report. Internal Report: http://iaks-www.ira.uka.de/iaks-calmet/intas.html, 2002.

[86] J. Calmet, C. Ballarin, and P. Kullmann. Integration of deduction and computation. *Applications of Computer Algebra*, pages 15–32, 2001.

[87] J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors. *CALCULEMUS-2002: Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, volume 2385 of *LNAI*. Springer, 2002.

[88] J. Calmet, F. Freitas, and G. Bittencourt. Master-web: An ontology-based internet data mining multi-agent system. In *Proceedings of SSGRR 2001, Computer & e-Business conference*, 2001.

[89] J. Calmet, W. Hausdorf, and W. Seiler. A constructive introduction to involution. In R. Akerkar, editor, *Proc. Int. Symp. Applications of Computer Algebra - ISACA 2000*, pages 33–50. Allied Publishers Limited, 2001.

[90] J. Calmet and K. Homann. Classification of communication and cooperation mechanisms for logical and symbolic computation systems. In K. Shultz and F.Baader, editors, *Frontiers of Combining Systems, Proceedings of FroCoS'96*, pages 221–234. Kluwer Series on Applied Logic, 1996.

[91] J. Calmet and K. Homann. Towards the mathematics software bus. *Theoretical Computer Science*, 187:221–230, 1997.

[92] J. Calmet, K. Homann, and I. Tjandra. Unified domains and abstract computational structures. In J. Calmet and J. Campbell, editors, *International Conference on Artificial Intelligence and Symbolic Mathematical Computing*, volume 737 of *LNCS*, pages 166–177. Springer Verlag, 1993.

[93] J. Calmet, S. Jekutsch, P. Kullmann, and J. Schü. Komet – a system for the integration of heterogeneous information sources. In *10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*. Springer Verlag, 1997.

[94] J. Calmet, P. Kullmann, and M. Taneda. Composite distributive lattices as annotation domains for mediators. *Annals of Math. and AI*, 36, 2002.

[95] J. Calmet and I. Tjandra. A unified-algebra-based specification language for symbolic computing. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*, volume 722 of *LNCS*, pages 122–133. Springer Verlag, 1993.

[96] O. Caprotti, H. Geuvers, and M. Oostdijk. Certified and portable mathematical documents from formal contexts. In Buchberger and Caprotti [71].

[97] O. Caprotti and W. Schreiner. Mathematical Software as Web Services. In Cohen et al. [106].

[98] O. Caprotti and W. Schreiner. Towards A Mathematical Services Description Language. In Cohen et al. [106].

[99] O. Caprotti and V. Sorge, editors. *Calculemus 2002, 10th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning: Work in Progress Papers*, Marseilles, France, June 2002. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.

[100] C. Castellini and A. Smaill. Proof planning for feature interactions: a preliminary report. In Baaz and Voronkov [25].

[101] C. Castellini and A. Smaill. A systematic presentation of quantified modal logics. *Logic Journal of the IGPL*, 10(6), November 2002.

[102] W. Chan, R. J. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. CAV'97*, volume 1254 of *LNCS*, pages 316–327, Haifa, Israel, June 1997. Springer.

[103] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Service Description Language. W3C Recommendation 1.1, World Wide Web Consortium, 2001. Available at `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[104] A. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive!* Springer, 1999.

[105] A. Cohen, S. Murray, M. Pollet, and V. Sorge. Certifying solutions to permutation group problems. Submitted to a major international conference, 2003.

[106] A. M. Cohen, X.-S. Gao, and N. Takayama, editors. *Mathematical Software - ICMS 2002*. World Scientific, August 2002.

[107] S. Colton. *Automated Theory Formation in Pure Mathematics*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 2000.

[108] S. Colton. Making conjectures about maple functions. In Calmet et al. [87].

[109] A. Craciun. The sequence provers in theorema. In Caprotti and Sorge [99]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.

[110] A. Craciun and B. Buchberger. Proving the correctness of the merge-sort algorithm with theorema. In Petcu et al. [216], pages 97–111.

[111] L. Cruz-Filipe. A constructive formalization of the fundamental theorem of calculus. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs, Proceedings of the International Workshop, TYPES 2002, Berg en Dal, NL*, LNCS. Springer, 2003. to appear.

[112] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[113] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[114] L. Dennis and J. Zimmer. Inductive theorem proving and computer algebra in the mathweb software bus. In Calmet et al. [87].

[115] J. Denzinger and S. Schulz. Analysis and representation of equational proofs generated by a distributed completion based proof system. Technical report, University of Kaiserslautern, April 1994.

[116] D. L. Detlefs, G. Nelson, and J. Saxe. Simplify: the ESC Theorem Prover. Technical report, DEC, 1996.

[117] X. Dousson, F. Sergeraert, and Y. Siret. *The Kenzo program*. Institut Fourier, Grenoble, 1999. `ftp://fourier.ujf-grenoble.fr/pub/KENZO`.

[118] Z. M. *et. al*. STeP: The Stanford Temporal Prover. Technical Report CS-TR-94-1518, Stanford University, June 1994.

[119] R. Endsuleit and T. Mie. Protecting co-operating mobile agents against malicious hosts. Internal Report 2002-8, University of Karlsruhe, 2002.

[120] S. Engell, S. Kowalewski, C. Schulz, and O. Stursberg. analysis and optimization of continuous-discrete interactions in chemical processing plants.

[121] J. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: integrated canonizer and solver. In *Proc. CAV'2001*, July 2001.

[122] A. Franke, M. Moschner, and M. Pollet. Cooperation between the Mathematical Knowledge Base MBase and the Theorem Prover Omega. In Caprotti and Sorge [99]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.

[123] The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4*, 1998. `http://www-gap.dcs.st-and.ac.uk/~gap`.

[124] H. Geuvers, R. Pollack, F. Wiedijk, and J. Zwanenburg. A constructive algebraic hierarchy in coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002.

[125] H. Geuvers, F. Wiedijk, and J. Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs, Proceedings of the International Workshop, TYPES 2000, Durham*, number 2277 in LNCS, pages 96–111. Springer, 2001.

[126] G. Gierz, K. Hofmann, K. Keimel, J. Lawson, M. Mislove, and D. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin, Heidelberg, New York, 1980.

[127] F. Giunchiglia. GETFOL: Interactive Multicontext Theorem Proving (abstract). In *Proceedings of IJCAI-93 Workshop on Automated Theorem Proving*, page 43, Chambery, France, 1993.

[128] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning theories: Towards an architecture for open mechanized reasoning systems. In F. Baader and K. U. Schulz, editors, *Frontiers of combining systems (FroCoS-1) : 1st international workshop, Munich, March 26-29, 1996*, volume 3 of *Applied logic series*. Kluwer Academic Publishers, 1996.

[129] F. Giunchiglia, R. Sebastiani, and P. Traverso. Integrating SAT solvers with domain-specific reasoners. In Kerber and Kohlhase [150].

[130] V. K. A. M. Gugenheim. On the chain complex of a fibration. *Illinois Journal of Mathematics*, 16:398–414, 1972.

[131] A. Heneveld, E. Maclean, A. Bundy, A. Smaill, and J. Fleuriot. Towards a formalisation of college calculus. In Kerber and Kohlhase [150].

[132] T. Henzinger and P. Ho. HYTECH: The Cornell Hybrid Technology Tool. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 265–293. Springer-Verlag, 1995.

[133] T. Henzinger and P. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 252–264. Springer-Verlag, 1995.

[134] T. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 377–388. Springer-Verlag, 1996.

[135] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[136] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In *HSCC*, pages 130–144, 2000.

[137] T. Hillenbrand, A. Jaeger, and B. Löchner. System Description: Waldmeister : Improvements in Performance and Ease of Use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE–16)*, volume 1632 of *LNAI*, pages 232–236, Trento, Italy, July 7–10, 1999. Springer Verlag, Berlin, Germany.

[138] R. Horowitz and P. Varaiya. Design of an automated highway system. Proceedings of the IEEE. This issue.

[139] M. Jamnik, M. Kerber, and M. Pollet. Automatic learning in proof planning. Technical Report CSRP-02-3, University of Birmingham, School of Computer Science, March 2002.

[140] M. Jamnik, M. Kerber, and M. Pollet. Automatic learning in proof planning. In F. van Harmelen, editor, *ECAI-2002: European Conference on Artificial Intelligence*, pages 282–286. IOS Press, 2002.

[141] M. Jamnik, M. Kerber, and M. Pollet. LearnOmatic: System description. In Voronkov [260], pages 150–155.

[142] T. Jebelean. Natural proofs in elementary analysis by s-decomposition. Poster presentation at ISSAC 2001: International Symposium on Symbolic and Algebraic Computation, London, Ontario, Canada, July 2001.

[143] T. Jebelean. Natural style predicate logic proving in theorema. Talk at ICAI'01: 5th International Conference on Applied Informatics Eger, Hungary, January 2001.

[144] T. Jebelean. Theorema: A system for the working mathematician. Invited talk at International Symposium "35 years of Automath" Edinburgh, Scotland, 10 - 13 Aug. 2002.

[145] T. Jebelean. Theorema: A system for the working mathematician. Software demonstration at ISSAC'02 (International Symposium for Symbolic and Algebraic Computation, Lille, France, July 2002. The abstract of the talk and the software demo are published electronically on the CD-ROM accompanying the proceedings (ACM Press).

[146] T. Jebelean and B. Buchberger. Theorema: A system for the working mathemtician. Talk at SNSC'01: Symbolic and Numeric Computation, Hagenberg, Austria, September 2001.

[147] T. Jebelean and B. Konev. Using Meta-variables for Natural Deduction in Theorema. In Kerber and Kohlhase [150], pages 160–175.

[148] D. Kapur, D. Musser, and X. Nie. An Overview of the Tecton Proof System. *Theoretical Computer Science*, Vol. 133, Oct. 1994.

[149] M. Kaufmann and J. S. Moore. Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. on Software Engineering*, 23(4):203–213, Apr. 1997.

[150] M. Kerber and M. Kohlhase, editors. *Symbolic Computation and Automated Reasoning – The CALCULEMUS-2000 Symposium*, St. Andrews, UK, August 6–7, 2000 2001. AK Peters, Natick, MA, USA.

[151] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra with Proof Planning. In J. Calmet and C. Limongelli, editors, *Design and Implementation of Symbolic Computation Systems; International Symposium, DISCO '96, Karlsruhe, Germany, September 18-20, 1996; Proceedings*, volume 1128, Berlin;Heidelberg;New York, 1996. Springer Verlag.

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

[153] M. Kerber and M. Pollet. On the design of mathematical concepts. Cognitive Science Research Papers CSRP-02-06, The University of Birmingham, School of Computer Science, May 2002.

[154] M. Kerber and M. Pollet. On the design of mathematical concepts. In B. McKay and J. Slaney, editors, *AI-2002: 15th Australian Joint Conference on Artificial Intelligence*. Springer, LNAI, 2002.

[155] M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming. *Journal of Logic Programming*, 12:335–367, 1992.

[156] H. Kirchner and C. Ringeissen, editors. *Proceedings of Third International Workshop Frontiers of Combining Systems (FROCOS 2000)*, volume 1794 of *LNCS*, Nancy, France, March 22–24 2000. Springer Verlag, Berlin, Germany.

[157] M. Kohlhase. OMDOC: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *Proceedings of AI and Symbolic Computation, AISC-2000*, LNAI. Springer Verlag, 2000.

[158] B. Konev and T. Jebelean. Combining Level-Saturation Strategies and Meta-Variables for Predicate Logic Proving in Theorema. In *Proceedings of IMACS ACA 2000, St.Petersburg, Russia*, June 2000.

[159] B. Konev and T. Jebelean. Solution lifting method for handling meta-variables in theorema. In Buchberger and Maruster [78].

[160] A. Korniłowicz. Properties of left and right components. *Formalized Mathematics*, 8(**1**):163–168, 1999.

[161] F. Kossak. An Interface for Interactive Proving with the Mathematical Software System *Theorema*. Master's thesis, FHS-Hagenberg, 1999.

[162] F. Kossak and K. Nakagawa. User System Interaction Within *Theorema*. In Armando and Jebelean [8], pages 69–82.

[163] J. Kotowicz and Y. Nakamura. Go-board theorem. *Formalized Mathematics*, 3(**1**):125–129, 1992.

[164] P. Kullmann. *Wissensrepraesentation und Anfragebearbeitung in einer logikbasierten Mediatorumgebung*. PhD thesis, University of Karlsruhe, 2001.

[165] G. Kusper. Investigation of binary representations of sat, especially 2-literal representation. In *Conference of PhD Students in Computer Science*, Szeged, Hungary, July 2002.

[166] G. Kusper. Solving the resolution-free sat problem by hyper-unit propagation in linear time. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, Cincinnati, Ohio, USA, May 2002.

[167] T. Kutsia. Unification in the empty and flat theories with sequence variables and flexible arity symbols. Talk at International Joint Conference on Automated Reasoning. Workshop UNIF'01 (Unification), June 2001. Extended abstract appeared in F.Baader, V.Diekert, C.Tinelli, R.Treinen (eds.), Proceedings of 15th International Workshop on Unification.

[168] T. Kutsia. Pattern unification with sequence variables and flexible arity symbols. In M. Ojeda-Asiego, editor, *Proceedings of the Workshop on Unification in Non-Classical Logics*, volume 66 of *Electronic Notes on Theoretical Computer Science*, Malaga, Spain, July 2002. Elsevier Science.

[169] T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols*. PhD thesis, RISC Linz, Johannes Kepler University, 2002.

[170] T. Kutsia. Theorem proving with sequence variables and flexible arity. In Baaz and Voronkov [25], pages 278–291.

[171] T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In Calmet et al. [87], pages 290–304.

[172] T. Kutsia and K. Nakagawa. System Description: Interface between Theorema and External Automated Deduction Systems. In Linton and Sebastiani [175].

[173] S. M. Lane. *Homology*. Springer Verlag, Berlin, Germany, 1994.

[174] G. Laumon and L. Moret-Bailly. *Champs algébriques*, volume 39 of *Ergebnisse der Mathematik*. Springer Verlag, 1999.

[175] S. Linton and R. Sebastiani, editors. *CALCULEMUS-2001 – 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, Siena, Italy, June 21–22 2001.

[176] S. Linton and R. Sebastiani, editors. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, volume 34 (4). Elsevier, 2002.

[177] *Logic, Mathematics and Computer Science: Interactions (LMCS 2002)*, RISC, Schloss Hagenberg, Austria, October 2002. Symposium in Honor of Bruno Buchberger's 60th Birthday, Appears in RISC-report Series Nr. 02-60, ISBN 3-902276-03-7.

[178] J. Lygeros, G. J. Pappas, and S. Sastry. An approach to the verification of the center-TRACON automation system. In *HSCC*, pages 289–304, 1998.

[179] E. Maclean. Automating proof in non-standard analysis (ii). In *Proceedings of ESSLLI 2001*, Helsinki, 2001.

[180] E. Maclean, J. Fleuriot, and A. Smaill. Proof-planning non-standard analysis. In *Proceedings of the 7th International Symposium on Aritifical Intelligence and Mathematics*, Fort Lauderdale, 2002.

[181] Z. Mann, J. Calmet, and P. Kullmann. Testing access to external information sources in a mediator environment. In *Proceedings of the 14th IFIP International Conference on Testing of Communicating Systems*, pages 111–126. Kluwer Academic Publishers, 2002.

[182] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, July 3–5 2000. IEEE Computer Society.

[183] H. Mantel. Unwinding Possibilistic Security Properties. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *LNCS*, pages 238–254, Toulouse, France, October 4-6 2000. Springer.

[184] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–101, Oakland, CA, USA, May 12–15 2002. IEEE Computer Society.

[185] Maple. CAS developed at the University of Waterloo, directed by G. Gonnet, http://www.maplesoft.com.

[186] Mathbroker - A Framework for Brokering Distributed Mathematical services. `http://poseidon.risc.uni-linz.ac.at:8080/index.html`.

[187] Mathematical web services workshop. `http://poseidon.risc.uni-linz.ac.at:8080/index.html`.

[188] Mathematica. CAS developed at Wolfram Research Inc., directed by S. Wolfram, http://www.wolfram.com.

[189] R. L. McCasland, M. E. Moore, and P. F. Smith. An introduction to zariski spaces over zariski topologies. *Rocky Mountain Journal of Mathematics*, 28:1357–1369, 1998.

[190] B. D. McKay. nauty user's guide (version 1.5). Technical Report TR-CS-90-02, Department of Computer Science, Australian National University, 1990.

[191] A. Meier. Tramp: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE–17)*, volume 1831 of *LNAI*, pages 460–464, Pittsburgh, PA, USA, June 17–20 2000. Springer Verlag, Berlin, Germany.

[192] A. Meier, M. Pollet, and V. Sorge. Exploring the Domain of Residue Classes. Seki Report SR-00-04, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, December 2000.

[193] A. Meier, M. Pollet, and V. Sorge. Classifying Isomorphic Residue Classes. In Moreno-Díaz et al. [202], pages 494–508.

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

[195] A. Meier and V. Sorge. Exploring Properties of Residue Classes. In Kerber and Kohlhase [150], pages 175–190.

[196] A. Meier, V. Sorge, and S. Colton. Employing theory formation to guide proof planning. In Calmet et al. [87], pages 275–289.

[197] E. Melis, E. Andres, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: A generic and adaptive web-based learning environment. *Journal of Artificial Intelligence and Education*, 12(4):385–407, 2001.

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

[199] R. Milewski. Fundamental theorem of algebra. *Formalized Mathematics*, 9(**3**):461–470, 2001.

[200] J. Moeller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. In *Electronic Notes in Theoretical Computer Science*, volume 23. Elsevier Science, 2001.

[201] The MONET Home Page. `http://monet.nag.co.uk/cocoon/monet/index.html`.

[202] R. Moreno-Díaz, B. Buchberger, and J.-L. Freire, editors. *Proceedings of the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, volume 2178 of *LNCS*, Las Palmas de Gran Canaria, Spain, February 19–23 2001. Springer Verlag, Berlin, Germany.

[203] M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, 2001.

[204] K. Nakagawa. *Supporting User-Friendliness in the Mathematical Software System Theroema*. PhD thesis, RISC Linz, Johannes Kepler University, 2002.

[205] K. Nakagawa. Variable shape logicographic symbols. In LMCS02 [177]. Symposium in Honor of Bruno Buchberger's 60th Birthday, Appears in RISC-report Series Nr. 02-60, ISBN 3-902276-03-7.

[206] K. Nakagawa and B. Buchberger. Presenting proofs using logicographic symbols. In A. Fiedler and H. Horacek, editors, *Proceedings of the Workshop on Proof Transformation and Presentation (PTP-01 in IJCAR-2001)*, page 11, Siena, Italy, June 2001.

[207] K. Nakagawa and B. Buchberger. Two tools for mathematical knowledge management in theorema. In Buchberger and Caprotti [71].

[208] Y. Nakamura and J. Kotowicz. The Jordan's property for certain subsets of the plane. *Formalized Mathematics*, 3(**2**):137–142, 1992.

[209] Y. Nakamura and A. Trybulec. The first part of Jordan's theorem for special polygons. *Formalized Mathematics*, 6(**1**):49–51, 1997.

[210] G. Nelson and D. Oppen. Fast Decision Procedures Based on Congruence Closure. *J. of the ACM*, 27(2):356–364, April 1980.

[211] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, volume 2283. Springer Verlag, Berlin, Germany, 2002.

[212] A. Nonnengart. A deductive model checking approach for hybrid systems. Technical Report MPI-I-1999-2-006, Max-Planck-Institute for Computer Science, Saarbrücken, Germany, November 1999. Available via http://www.mpi-sb.mpg.de/.

[213] A. Nonnengart and A. Szalas. A fixpoint approach to second-order quantifier elimination with applications to correspondence theory. Technical Report MPI-I-95-2-007, MPII Saarbrücken, Saarbrücken, Germany, 1995.

[214] M. Oostdijk. *Generation and Presentation of Formal Mathematical Documents*. PhD thesis, Eindhoven University of Technology, Sept. 2001.

[215] P. Paule and M. Schorn. A mathematica version of zeilberger's algorithm for proving binomial coefficient identities. *JSC*, 20:973–698, 1995.

[216] D. Petcu, V. Negru, D. Zaharie, and T. Jebelean, editors. *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'02)*, October 2002.

[217] J. P. Pickett and et al. *The American Heritage Dictionary of the English Language. Fourth edition.* Houghton Mifflin Company, Boston, 2000. Available at: http://www.bartleby.com/61/31/J0063100.html.

[218] F. Piroi. Focus windows: A tool for automated provers. In Petcu et al. [216].

[219] F. Piroi. *Proof Simplification in Automated Theorem Proving*. PhD thesis, RISC Institute, ongoing.

[220] F. Piroi and B. Buchberger. Focus windows: A new technique for proof presentation. In Calmet et al. [87].

[221] F. Piroi and B. Buchberger. Focus windows: A new technique for proof presentation. In H. Kredel and W. Seiler, editors, *Proceedings of the 8th Rhine Workshop on Computer Algebra*, Mannheim, Germany, 2002.

[222] F. Piroi and T. Jebelean. Advanced proof presentation in theorema. In Buchberger and Maruster [78].

[223] F. Piroi and T. Jebelean. Interactive proving in theorema. In T. Walsh, editor, *Collected Abstracts of Ninth Workshop on Automated Reasoning, AISB'02*, Imperial College of Science, Technology and Medicine, University of London, England, April 3-5 2002.

[224] W. Pugh. The Omega Test: a fast and practical integer programming algorithm for dependence analysis. *Communication of the ACM*, August 1992.

[225] S. Ranise. Combining generic and domain specific reasoning by using contexts. In Calmet et al. [87].

[226] T. Recio and M. Kerber, editors. *Computer Algebra and Mechanized Reasoning: Selected St. Andrews' ISSAC/Calculemus 2000 Contributions*, volume 32(1/2) of *Journal of Symbolic Computation*, 2001.

[227] D. Redfern. *The Maple Handbook: Maple V Release 5*. Springer Verlag, Berlin, Germany, 1999.

[228] J. D. C. Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with Lambda-Clam. In *CADE'98*, volume 1421 of *LNCS*, pages 129–133, 1998.

[229] J. Robu. Systematic exploration of geometric configurations using mathematica. Talk at SYNASC 2001, Timisoara, Romania, October 2001.

[230] J. Robu. *Geometry Theorem Proving in the Frame of Theorema Project*. PhD thesis, RISC Linz, Johannes Kepler University, 2002.

[231] J. Robu. Geometry theorem proving in the frame of theorema project. Talk at: 4th International Workshop on Automated Deduction in Geometry (ADG 2002), RISC, September 2-4 2002. to appear in Springer LNCS series.

[232] J. Rubio and F. Sergeraert. Constructive Algebraic Topology. In *Lecture Notes Summer School in Fundamental Algebraic Topology*. Institut Fourier, 1997. `http://www-fourier.ujf-grenoble.fr/\~{}sergerar/Summer-School/`.

[233] J. Rubio and F. Sergeraert. Constructive Algebraic Topology. *Bulletin des Sciences Mathématiques*, 126:389–412, 2002.

[234] J. Rubio, F. Sergeraert, and Y. Siret. EAT: Symbolic Software for Effective Homology Computation. Technical report, Institut Fourier, Grenoble, 1997. `ftp://fourier.ujf-grenoble.fr/pub/EAT`.

[235] F. Sergeraert. The computability problem in Algebraic Topology. *Advances in Mathematics*, 104:1–29, 1994.

[236] W. Shih. Homologie des espaces fibrés. *Publications Mathématiques de l'I.H.E.S.*, 13, 1962.

[237] R. Shostak. Deciding Combination of Theories. *Journal of the ACM*, 31(1):1–12, 1984.

[238] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with omega. In Voronkov [260], pages 144–149.

[239] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Irrationality of Square Root of 2 - A Case Study in OMEGA. Submitted to an International Journal, 2002.

[240] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with omega: Sqrt(2) is irrational. In Baaz and Voronkov [25], pages 367–387.

[241] A. Smaill and I. Green. Higher-order annotated terms for proof search. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1275 of *LNCS*, pages 399–414, Turku, Finland, 1996. Springer-Verlag. Also available as DAI Research Paper 799.

[242] A. Smaill and I. Green. Higher-order annotated terms for proof search. In *TPHOLs'96*, volume 1275 of *LNCS*. Springer, 1996.

[243] V. Sorge. Non-Trivial Symbolic Computations in Proof Planning. In Kirchner and Ringeissen [156], pages 121–135.

[244] G. Steel, A. Bundy, and E. Denney. Finding counterexamples to inductive conjectures and discovering security protocol attacks. *AISB Journal*, 1(2), 2002.

[245] G. Steel, A. Bundy, and E. Denney. Finding counterexamples to inductive conjectures and discovering security protocol attacks. In *Proceedings of the Foundations of Computer Security Workshop*, 2002. Appeared in Proceedings of The Verify'02 Workshop as well. Also available as Informatics Research Report EDI-INF-RR-0141.

[246] A. Stump, C. W. Barret, and D. L. Dill. CVC: A Cooperative Validity Checker. In *Proc. CAV'02*, LNCS. Springer, July 2002.

[247] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Journal on Autonomous Agents and Multi-Agent Systems, Kluwer Academic*, 4(4), 2001.

[248] Y. Takeuchi and Y. Nakamura. On the Jordan curve theorem. Technical Report 19804, Dept. of Information Eng., Shinshu University, 500 Wakasato, Nagano city, Japan, April 1980.

[249] The Information Bus Company (Tibco). Tp-tib overview. Technical report, Teknekron Software Systems Inc., 1995.

[250] E. Tomuta. *An Architecture for Combining Provers and its Applications in the Theorema System*. PhD thesis, The Research Institute for Symbolic Computation, Johannes Kepler University, 1998. RISC report 98-14.

[251] E. Tomuta and B. Buchberger. Combining Provers in the Theorema System. In *Proceedings of the Sixth Rhine Workshop on Computer Algebra, March 31.–April 3, Sankt Augustin, Germany*, 1998.

[252] E. Tomuta, D. Vasaru, and M. Marin. Handling Provers Cooperation in Theorema. In B. Buchberger and T. Jebelean, editors, *Proceedings of the Second International Theorema Workshop*, pages 55–75, 1998. RISC report 98-10.

[253] A. Trybulec. Many sorted algebras. *Formalized Mathematics*, 5(**1**):37–42, 1996.

[254] A. Trybulec. Introducing spans. *Formalized Mathematics*, 10(**2**):97–98, 2002.

[255] A. Trybulec and Y. Nakamura. On the components of the complement of a special polygonal curve. *Formalized Mathematics*, 8(**1**):21–23, 1999.

[256] J. Urban. Free order sorted universal algebra. *Formalized Mathematics*, 10(**3**):211–225, 2002.

[257] J. Urban. Order sorted algebras. *Formalized Mathematics*, 10(**3**):179–188, 2002.

[258] D. Vasaru-Dupré. *Automated Theorem Proving by Integrating Proving, Solving and Computing*. PhD thesis, RISC Institute, May 2000. RISC report 00-19.

[259] C. Vogt. *Mathematical Knowledge Management*. PhD thesis, RISC Institute, ongoing.

[260] A. Voronkov, editor. *Proceedings of the 18th International Conference on Automated Deduction (CADE-19)*, volume 2392 of *LNAI*, Copenhagen, Denmark, 2002. Springer.

[261] T. Weibel and G. H. Gonnet. An assume facility for CAS, with a sample implementation for Maple. In J. Fitch, editor, *DISCO*, volume 721 of *Lecture Notes in Computer Science*, pages 95–103. Springer, 1993.

[262] M. Wenzel and F. Wiedijk. A comparison of the mathematical proof languages mizar and isar. *Journal of Automated Reasoning (submitted)*, page 26, 2003.

[263] F. Wiedijk. The fifteen provers of the world. Unpublished Draft available at `http://www.cs.kun.nl/~freek/notes/index.html`.

[264] F. Wiedijk. Mizar light for hol light. In Boulton and Jackson [53], pages 378–393.

[265] F. Wiedijk. Comparing mathematical provers. In Asperti et al. [19].

[266] W. Windsteiger. Building up hierarchical mathematical domains using functors in *mathematica*. In Armando and Jebelean [8], pages 83–102. CALCULEMUS 99 Workshop, Trento, Italy.

[267] W. Windsteiger. A Set Theory Prover in Theorema. In Moreno-Díaz et al. [202], pages 525–539. extended version available as RISC report 01-07.

[268] W. Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications*. PhD thesis, RISC Institute, May 2001.

[269] W. Windsteiger. On a Solution of the Mutilated Checkerboard Problem using the Theorema Set Theory Prover. In Linton and Sebastiani [175].

[270] W. Windsteiger. An automated prover for Zermelo-Fraenkel set theory in Theorema. In LMCS02 [177]. Symposium in Honor of Bruno Buchberger's 60th Birthday, Appears in RISC-report Series Nr. 02-60, ISBN 3-902276-03-7.

[271] W. Windsteiger. An Automated Prover for Set Theory in Theorema. In Caprotti and Sorge [99]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.

[272] S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.

[273] F. Wolter and M. Zakharyaschev. Spatio-temporal representation and reasoning based on rcc-8. KR2000: Principles of Knowledge Representation and Reasoning.

[274] J. Zhang and H. Zhang. Generating Models by SEM. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction (CADE–13)*, volume 1104 of *LNAI*, pages 308–312, New Brunswick, NJ, USA, July 30– August 3 1996. Springer Verlag, Berlin, Germany.

[275] J. Zimmer. Blue Note: A Specification of the ATP Interface in Mathweb, August 2001. Available at `http://www.mathweb.org/~jzimmer/atpspec.ps`.

[276] J. Zimmer, A. Armando, and C. Giromini. Towards Mathematical Agents – Combining MathWeb-SB and LB. In Linton and Sebastiani [175], pages 64–77.

[277] J. Zimmer and C. Benzmüller, editors. *CALCULEMUS Autumn School 2002: Student Poster Abstracts*, number SR-02-06 in SEKI Technical Report, 2002.

[278] J. Zimmer and M. Kohlhase. System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. In Voronkov [260], pages 144–149.