Mathematical Journals as Reasoning Agents: Literature Review

Contents

Mathematical Journals as Reasoning Agents: Literature Review

Florina Piroi, Bruno Buchberger, Camelia Rosenkranz

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Schloss Hagenberg, A4232
Hagenberg, Austria
{F.Piroi, Bruno.Buchberger,Camelia.Rosenkranz}@risc.uni−linz.ac.at

Version 2008−03−02

Abstract

This report reviews the literature relevant for the research project "Math−Agents: Mathematical Journals as Reasoning Agents" proposed by Bruno Buchberger as a technology transfer project based on the results of the SFB Project "Scientific Comput−ing", in particular the project SFB 1302, "Theorema". The project aims at computer−supporting the refereeing process of mathematical journals by tools that are mainly based on automated reasoning and also at building up the knowledge archived in mathematical journals in such a way that they can act as interactive and active reason−ing agents later on. In this report, we review current mathematical software systems with a focus on the availability of tools that can contribute to the goals of the Math−Agents project.

1 Introduction

1.1 The Goals of the Math−Agents Project

For making this technical report self−contained, we summarize here the goals of the project "Math−Agents: Mathematical Journals as Reasoning Agents" proposed in [Buchberger07]:

Mathematical knowledge as currently stored by journals (in printed or electronic form) is not yet accessible to any logical processing. In the past three decades, automated reasoning tech−niques and software−technology have matured to the extent that it seems to be possible to automate or, at least computer−support, the essential knowledge processing activities mathemati-cal journals. We have the technology that should enable us to construct reasoning tools and to re−organize the knowledge contained in any mathematical journal in such a way that questions

of the following type can be answered automatically or, at least, with significant computer−support:

- *Originality* - Is a given mathematical result already "contained" in the journal, i.e. can it be derived from the knowledge in the journal by "easy" (automated) reasoning?

- *Correctness* - Is a result in a paper correct, i.e. can it be formally derived from the available knowledge (using the proof "script" given in the paper)?

- *Importance* - Is the result important, i.e. how, where, and how often can it be used in the context of the available knowledge?

- *Completeness* - Are all "easy" consequences of a given result drawn in the available knowledge context?

- *Similarity* - Search for structurally similar results in all areas of available knowledge! Create a pattern.

- *Complete Knowledge* - Draw all consequences of the available knowledge using the currently available reasoners.

- *Knowledge Reduction* - Given (increasingly sophisticated) automated reasoners, from time to time, the amount of mathematical knowledge to be stored can be reduced to the propositions whose proof is not "easy" (w.r.t. the available automated reasoning power) whereas all other knowledge can be reproduced quickly, by the available automated reasoners, on demand.

- *Generalization* - Extract the abstract structure from available knowledge. For this, domains, schemes, functors, categories are important abstraction tools.

- *Minimal Prerequisites* - Analyze existing mathematical propositions for minimal prerequisites necessary in their proof.

- *Trace History* - Generate the successive versions of a given result (notion, proposition, problem, method) in the knowledge base over time.

- *Re−organize knowledge* - Take (a part of) the given knowledge and re−structure it under a particular perspective ("view").

- *Problem Generation* - What questions can be asked and what problems can be formulated on the basis of the available knowledge.

- *Etc.*.

The Math−Agent project will be carried out in the frame of the *Theorema* project, see [BuchbergerEtAl06, BuchbergerEtAl00, BuchbergerEtAl97], with the goal of extending the *Theorema* system by adding more tools and more sophisticated tools for mathematical knowl−edge management towards achieving the goals listed above.

1.2 Logical and Organizational Knowledge Management Tools

In this report, we consider both logical and organization tools for mathematical knowledge management. The emphasis in on logical tools.

- *Logical tools:* These are tools based on formal logic and automated reasoning and, hence, on the mathematical content of the information processed.

- *Organizational tools:* These are tools based on non−mathematical aspects of the information processed, like the distribution of mathematical knowledge in file systems, nested structuring of mathematical text, accessibility over the web, syntax standards, formats etc.

1.3 Classes of Mathematical Software Systems

In the early days of computing, mathematical software systems were basically *libraries of mathematical algorithms* − at the beginning (~ 1960) mainly numerical algorithms but also algorithms of discrete mathematics, later (~ 1965, 1970) algebraic algorithms, and then also logical algorithms (provers, reasoners).

Starting from ~ 1980 some of the algebraic algorithm libraries, like Mathematica, added a mathematical programming language and various tools for producing mathematical texts together with formulae. These systems were and are often called "computer algebra systems" although, typically, they also contain most of the common numerical algorithms.

In parallel (since ~ 1960), but quite independently, systems that contained logical algorithms ("automated provers", more generally "automated reasoners") were developed. In the early days and until quite recently, most of them concentrated on just one automated proving method, like resolution. Also, computer algebra systems and reasoning systems had little interaction with each other.

Also in parallel, powerful graphical tools were developed and added to the numerical and algebraic software systems.

Since approximately 1985, partly under the influence of the second author, it was felt that computer algebra systems and automated reasoning systems should move closely together and that, in reasoning systems, an emphasis should be put on multi−method systems that combine, in particular, general reasoning methods, like resolution, with special reasoning methods, like cylindrical algebraic decomposition. Also it was felt that tools for organizing and processing mathematical knowledge (definitions, theorems, problems, algorithms) should be added to mathematical software systems: See the editorial of the Journal of Symbolic Computation

[Buchberger85]; the creation of the Calculemus Research Consortium, see [Buchberger99]; the keynote presentation at the "10 Years of Mathematica" Conference [Buchberger98]; and the MKM (Mathematical Knowledge Management) conference series and consortium [Buchberger−Caprotti01].

Hence, current (general and special purpose) mathematical software systems often combine (numerical, discrete, algebraic, logical) algorithm libraries, mathematical programming languages, advanced graphics, and some first MKM tools for organizing and processing mathematical knowledge and, therefore, it is not any more possible and meaningful to classify mathematical software systems in numerical systems, computer algebra systems, automated reasoning systems, etc. Any combination of these features is encountered in current systems. Some of the systems focus on MKM tools only and are meant to provide an organizational frame from which the mathematical contents (algorithms and knowledge) of other existing systems can be called. However, in our view, none of the current mathematical software systems is yet truly comprehensive and universal as a uniform frame for "doing all of mathematics".

Since (fortunately) the boundaries between numerical, algebraic, logical etc. systems are being blurred, in this report, we do not group mathematical software systems into various distinct classes. Rather, we discuss the systems in alphabetic order and, for each system, we compile the information that characterizes the main features like type of algorithm libraries, programming language, reasoning tools, tools for mathematical knowledge management etc., see next section.

1.4 The Structure of the Report

The report lists (hopefully most of) the available mathematical software systems and, for each system, provides the information necessary for judging its appropriateness for the goals to be pursued in the Math−Agent project.

For each system, the information is structured in the following way:

- Technical Information on the System
  - Name of the System/Project and Website
  - Project Leader and Group
  - Main Publications
  - Implementation Language
  - Availability and System Prerequisites (sw and hw)
- Algorithm Libraries
  - Numerical Library
  - Discrete Math Library

- Algebraic Library
- Reasoners

  List and short description of the most important general and special reasoners in the system.

- Graphical Tools and Interfaces

  In particular, graphics that supports reasoning.

- User Language
  - Programming Language

    Mathematical language(s) in which the user can formulate algorithms. (Not the implementation language!)

  - Logic Language for the Formulation of Mathematical Knowledge

    Mathematical language(s) in which the user can formulate mathematical knowledge (definitions, theorems, problems).

  - Mathematical Syntax

    External syntax (syntaxes) for formulating algorithms and knowledge (may be different from the internal syntax!)

- Mathematical Knowledge Bases
  - Available Theories/Knowledge Bases

    Are there already (verified) mathematical theories (consisting of definitions, theorems, problems, algorithms etc.) available in the system?

  - Tools for Retrieval in Mathematical Knowledge Bases

    Given a formula (definition, theorem, problem, algorithm) and a knowledge base. How can one decide wether the formula is already contained in the knowledge base? ... whether equivalent, similar, etc. formulae are in the knowledge base?

  - Tools for Inventing Mathematical Knowledge

Is there support for inventing, in the frame of a given knowledge base, additional definitions, theorems, problems, algorithms? How is the invention supported? Bottom−up? Top−down (by a kind of "lazy thinking" mechanism)? In particular, tools for algorithm invention?

- Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

  Given a (consistent, verified) knowledge base and a new formula (axiom, definition, theorem, problem, algorithm). Is there support for checking whether the new formula is consistent? true? true under additional conditions? etc. In particular, tools for algorithm verification?

- Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

  Given a knowledge base, *complete* it by drawing all "easy but interesting" consequences. ("Easy", a relative notion: An "easy" consequence can be generated by the "strong" automated provers currently available in the system for the given theory. "Interesting", a relative notion: An "interesting" consequence cannot be drawn by "weak" automated provers currently available in the system for the given theory. For example: In the context of elementary analysis, the PCS prover [Buchberger01] is a "strong" prover, i.e. it combines quantifier proving and algebraic inequality solving in a certain way that may be seen as the essence of proving in this area. In contrast, a conditional rewrite prover that only processes rewrite knowledge like limit and differentiation rules, in this context, is a "weak" prover. More on that see [Buchberger06].

  Given a knowledge base, *reduce* it by eliminating all formulae that can be derived from others by "easy" provers. If possible, find minimal prerequisites from which the knowledge can be derived.

  Given a knowledge base, try to *structure and re−structure* it either be organizational means like chapters, sections etc. that are in correspondence with mathematical contents or by mathematical / logical means like schemes, domains, functors, categories (see [Buchberger96], [Buchberger03]).

- (Verified) Programming of Reasoners, Meta−Programming, Quotation

Is it possible for the user to add reasoners by formulating them within the system? Does the system give support for proving the correctness of user−defined reasoners? Is there a way to access the meta−level of the system and to go back and forth between the object and meta−level by an appropriate quoting (or other) mechanism.

- Standardization, Inter−Operability

  Can the algorithm and knowledge libraries of the system be translated back and forth from and to other systems? Do they use standard formats?

- Web Access

  Can the algorithm and knowledge libraries of the system be accessed interactively through the web?

- Example of a Theory Exploration Session

  How does a typical session in the system look like? From which situation do we start, what does the user do? What is the result? We try to provide an example of a session that illustrates the most advanced features in the direction of "mathematical knowledge management". Hence, the example should not only indicate what is possible in the system but it should also indicate that beyond the capabilities shown essentially nothing more sophisticated is possible.

2 ACL2

2.1 Short Description

ACL2 is a proof system from the Boyer−Moore family, used to construct mathematical models of computer hardware and software. It is "a mathematical logic together with a mechani−cal theorem prover" [ACL2Web]. Its most common uses are as a programming/specification/modelling language, a formal mathematical logic, or a theorem prover.

2.2 Technical Information on the System

2.2.1 Name of the System and Website

ACL2 − A Computational Logic for Applicative Common Lisp. (The ACL appears twice in the extended name, that’s why the 2 in ACL2.)

http://www.cs.utexas.edu/users/moore/acl2/

2.2.2 Project Leaders and Group

Matt Kaufmann and J Strother Moore (University of Texas, Austin, USA).
Various other people contributed to ACL2 by writing documentation, tutorials, porting to other operating systems, etc. (see the acknowledgments page on [ACL2Web].

2.2.3 Main Publications

Kaufmann, M. and Moore, J., An Industrial Strength Theorem Prover for a Logic Based on Common Lisp, *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 203−213, 1997.

Kaufmann, M., and Manolios, P., and Moore, J. *Computer−Aided Reasoning: An Approach*, Kluwer Academic Publishers, June, 2000, ISBN 0−7923−7744−3.

Kaufmann, M., and Manolios, P., and Moore, J., editors. *Computer−Aided Reasoning: ACL2 Case Studies*, Kluwer Academic Publishers, June, 2000, ISBN 0−7923−7849−0.

2.2.4 Implementation Language

With the exception of a small amount of Common Lisp functions (mainly for bootstraping), ACL2 is written in its own formal language [KaufmannMoore06].

2.2.5 System Availability and Prerequisites

The system is freely available under the GNU General Public License from its website.

ACL2 runs on Unix, Linux, Windows, and Macintosh operating systems. For other distribu−tions, the system can be built in one of the following Common Lisp implementations: Allegro, GCL (Gnu Common Lisp), Lispworks, CLISP, CMU Common Lisp, SBCL, OpenMCL, and MCL (Macintosh Common Lisp).

2.3 Algorithm Libraries

The system has no libraries of algorithms as computer algebra systems have.

2.3.1 Reasoners

The ACL2 prover is rule driven. The prover uses three important techniques: induction, rewriting and inequality chaining (linear arithmetic). Choosing the induction argument is done by a variety of heuristics. Users can also specify the induction scheme, when necessary. Defini−tions and proved theorems are turned into rewriting rules of one of the three technique kinds − induction, rewriting, arithmetic. The users can specify which kind of rule to be generated, or if no rule should be generated at all.

2.3.2 Graphical Tools and Interfaces

The Emacs editor is typically used to edit forms, which then are submitted to ACL2 via Emacs's *shell* buffer. There are a couples of utilities for Emacs like speeding up the interac−tion with the system and infix printing.

2.4 User Language

2.4.1 Programming Language

ACL2, i.e. a subset of applicative Common Lisp.

2.4.2 Logic Language for the Formulation of Mathematical Knowledge

ACL2 logic. The logic of ACL2 is first−order with total recursive functions providing mathematical induction on the ordinals up to ϵ_0, and two extension principles. It has no quantifi−ers and is un−typed, in the sense that the syntax is un−typed.

2.4.3 Mathematical Syntax

The syntax of ACL2 (i.e. the one of Common Lisp).

2.5 Mathematical Knowledge Bases

The ACL2 distribution includes a set of ‘‘books’’. Books are a collections of definitions and/or theorems and/or other commands that extend the system's reasoning capabilities. To become a book, such a collection of definitions, theorems and commands must be certified by the system. During certification, books may also be compiled, speeding up the execution of the functions (definitions, theorems, commands) defined in it. Books can be included into an ACL2 session using an *include_book* command (without certifying them). Including a certified book into the current session guarantees that the error−free inclusion produces a consistent extension of a consistent logic.

2.5.1 Available Theories and Knowledge Bases

The theories available in books are not systematically organized, rather independent pieces of mathematics which often may be composed for the purpose of a work session. The list of theories below is, most probably, not complete, as with each ACL2 workshop − 6 until now − more ‘‘books’’ with mathematical content are created.

* Arithmetic: contains theorems about absolute value, factorial function, integer division and related function, sum of a list of numbers, rules for arithmetic equality, a proof of the binomial theorem, arithmetic equality and inequality rules, rationals, floating point arithmetic support.
* Support for reasoning about real numbers using non−standard analysis.

* Integer hardware specification: integer arithmetic suitable for hardware modelling
* Ordinals
* Finite set theory
* Data structures: lists, suited to the analysis of recursive, data−parallel algorithms, etc.

2.5.2 Tools for Retrieval in Mathematical Knowledge Bases

2.5.3 Tools for Inventing Mathematical Knowledge

2.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The system's reasoner (see the corresponding subsection in the 'Algorithm Libraries' section).

2.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

2.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

Within the ACL2 system, users can write their own simplifiers which, before being used, must be proved correct by the system.

Each theorem that was proved is transformed to a rule − unless the user explicitly forbid this − which is then used by the system's reasoner.

2.5.7 Standardization, Inter−Operability

2.5.8 Web Access

2.6 Example of a Theory Exploration Session

The ACL2 system interacts with users by a read−eval−print loop. The input language is case insensitive.

The example shown below is taken from the on−line tour available on the system's website. For readability, we have marked the system's answers with '>>' (see below).

Define an append function:

```
ACL2 !>(defun app (x y)
  (cond ((endp x) y)
        (t (cons (car x)
                 (app (cdr x) y)))))
```

```
>> The admission of APP is trivial, using the relation O< (which
>> is known to be well-founded on the domain recognized by O-P)
>> and the measure (ACL2-COUNT X). We observe that the type of APP
>> is described by the theorem (OR (CONSP (APP X Y)) (EQUAL (APP X Y) Y)).
>> We used primitive type reasoning.

>> Summary
>> Form: ( DEFUN APP ...)
>> Rules: ((:FAKE-RUNE-FOR-TYPE-SET NIL))
>> Warnings: None
>> Time: 0.03 seconds (prove: 0.00, print: 0.00, other: 0.03)
>> APP
```

Comments about the system's answer above: In order to be accepted by the system, recursive definitions must have the termination property. Termination must be proved before admitting the definition. The proof, shortly presented in the first paragraph of the above output, is done by ACL2' reasoner by induction on the size of X, using the standard ordering on ordinals less than ϵ_0. The induction scheme is automatically chosen and involves elaborate heuristics. The 'Rules' listed in the 'Summary' of the proof are the ones involved to obtain the proof.

Evaluate APP on some sample input:

```
ACL2 !>(app nil '(x y z))
>> (X Y Z)

ACL2 !>(app '(1 2 3) '(4 5 6 7))
>> (1 2 3 4 5 6 7)

ACL2 !>(app (app '(1 2) '(3 4)) '(5 6))
>> (1 2 3 4 5 6)

ACL2 !>(app '(1 2) (app '(3 4) '(5 6)))
>> (1 2 3 4 5 6)

ACL2!>(let  ((a '(1 2))
        (b '(3 4))
        (c '(5 6)))
    (equal  (app (app a b) c)
        (app a (app b c))))
>> T
```

The last examples in this list suggests that the APP function may be associative. We write a theorem about this:

```
ACL2!>(defthm associativity-of-app
    (equal  (app (app a b) c)
                    (app a (app b c))))
```

and after submitting this to the system we get the following output:

```
>> Name the formula above *1.

>> Perhaps we can prove *1 by induction. Three induction schemes are
>> suggested by this conjecture. Subsumption reduces that number to
two.
```

```
>> However, one of these is flawed and so we are left with one viable
>> candidate.

>> We will induct according to a scheme suggested by (APP A B). If we
>> let (:P A B C) denote *1 above then the induction scheme we'll use
>> is
>> (AND
>>  (IMPLIES    (AND     (NOT (ENDP A))
>>              (:P (CDR A) B C))
>>          (:P A B C))
>>  (IMPLIES (ENDP A) (:P A B C))).
>> This induction is justified by the same argument used to admit APP,
>> namely, the measure (ACL2-COUNT A) is decreasing according to the
relation
>> O< (which is known to be well-founded on the domain recognized
>> by O-P). When applied to the goal at hand the above induction
>> scheme produces the following two nontautological subgoals.
```

Simplification of the Induction Step to T(rue):

```
>> Subgoal *1/2
>> (IMPLIES (AND (NOT (ENDP A))
>>               (EQUAL (APP (APP (CDR A) B) C)
>>                      (APP (CDR A) (APP B C))))
>>          (EQUAL (APP (APP A B) C)
>>                 (APP A (APP B C)))).

>> By the simple :definition ENDP we reduce the conjecture to

>> Subgoal *1/2'
>> IMPLIES (AND (CONSP A)
>>               (EQUAL (APP (APP (CDR A) B) C)
>>                      (APP (CDR A) (APP B C))))
>>          (EQUAL (APP (APP A B) C)
>>                 (APP A (APP B C)))).


>> But simplification reduces this to T, using the :definition APP, the
>> :rewrite rules CDR-CONS and CAR-CONS and primitive type reasoning.
```

Simplification of the Base Case to T(rue)

```
>> Subgoal *1/1
>> (IMPLIES (ENDP A)
>>         (EQUAL (APP (APP A B) C)
>>                (APP A (APP B C)))).

>> By the simple :definition ENDP we reduce the conjecture to

>> Subgoal *1/1'
>> (IMPLIES (NOT (CONSP A))
>>          (EQUAL (APP (APP A B) C)
>>                 (APP A (APP B C)))).

>> But simplification reduces this to T, using the :definition APP and
>> primitive type reasoning.
```

End of the proof:

```
>> That completes the proof of *1.

>> Q.E.D.

>> Summary
>> Form:  ( DEFTHM ASSOCIATIVITY-OF-APP ...)
>> Rules: ((:REWRITE CDR-CONS)
>>         (:REWRITE CAR-CONS)
>>         (:DEFINITION NOT)
>>         (:DEFINITION ENDP)
>>         (:FAKE-RUNE-FOR-TYPE-SET NIL)
>>         (:DEFINITION APP))
>> Warnings:  None
>> Time:  0.27 seconds (prove: 0.10, print: 0.05, other: 0.12)
>>  ASSOCIATIVITY-OF-APP
```

When the theorem is proved, it is also transformed to a rewrite rule and can be used in further proving.

3 ActiveMath

3.1 Short Description

ActiveMath is an interactive learning environment for mathematics [LibbrechtMelis06], based on an extension of OMDoc.It is web-based and user-adaptive.Its building blocks are *content items* holding both mathematical expressions and text (that can contain hyperlinks); they are annotated with mathematical and pedagogical attributes and relations (i.e.metadata). For managing, referencing and searching mathematical information, it mainly uses the OMDoc level of definitions, examples and theorems.

3.2 Technical Information on the System

3.2.1 Name of the System and Website

ActiveMath. http://www.activemath.org/

3.2.2 Project Leaders and Group

The project leader is Erica Melis. The group members are: George Goguadze, Martin Homik, Bruce McLaren, Paul Libbrecht, Alberto Gonzalez Palomo, Oliver Scheuer, Ian Tsigler, Dimitra Tsovalzi, Stefan Winterstein.

3.2.3 Main Publications

Paul Libbrecht, Erica Melis, Methods for Access and Retrieval of Mathematical Content in ActiveMath, Proceedings of ICMS-2006, Ed. Nobuki Takayama, Andres Iglesias, Jaime Gutierrez, LNCS 4151, Springer Verlag GmbH.

E. Melis, J. Siekmann, ActiveMath: An Intelligent Tutoring System for Mathematics, Seventh International Conference Artificial Intelligence and Soft Computing (ICAISC).

E. Melis, J. Büdenbender, G. Goguadze, P. Libbrecht, M. Pollet, C. Ullrich, Knowledge Representation and Management in ActiveMath, Annals of Mathematics and Artificial Intelligence, Special Issue on Management of Mathematical Knowledge, Proceedings of the first Conference on Mathematical Knowledge Management MKM'01, volume 38, no. 1−3, pages 47−64.

3.2.4 Implementation Language

Active Math is implemented as a Java−based web application. Its search engine is based on Lucene, a full-text search engine written in Java.

3.2.5 System Availability and Prerequisites

ActiveMath has an open source license for non−commercial applications. Demo freely available online. Upon request, the full ActiveMath can be downloaded free of charge.

For locally installing ActiveMath one needs:

− a relatively fast machine with at least 512 Mb of RAM, and about 1 Gb of free disk space;
− Java JDK 5;
− and a Subversion (http://subversion.tigris.org/) client.

For running the online available demo one needs:

− Firefox, at least version 1.5 or Internet Explorer 6 (other browsers may work,but are not their primary target).Cookies and Javascript need to be enabled.
− Java Plug−in for your browser
− screen size:1024x768 or better.
− a fast Internet connection (DSL).

3.3 Algorithm Libraries

The ActiveMath exercises execute algorithms from Computer Algebra Systems (CAS) like Yacas [Yacas], Maxima [Maxima], WIRIS [WIRIS]. The called libraries cover the domains listed in Mathematical Knowledge Bases / Mathematical Theories.

3.3.1 Numerical, Discrete, Algebraic, etc. Libraries

3.3.2 Reasoners

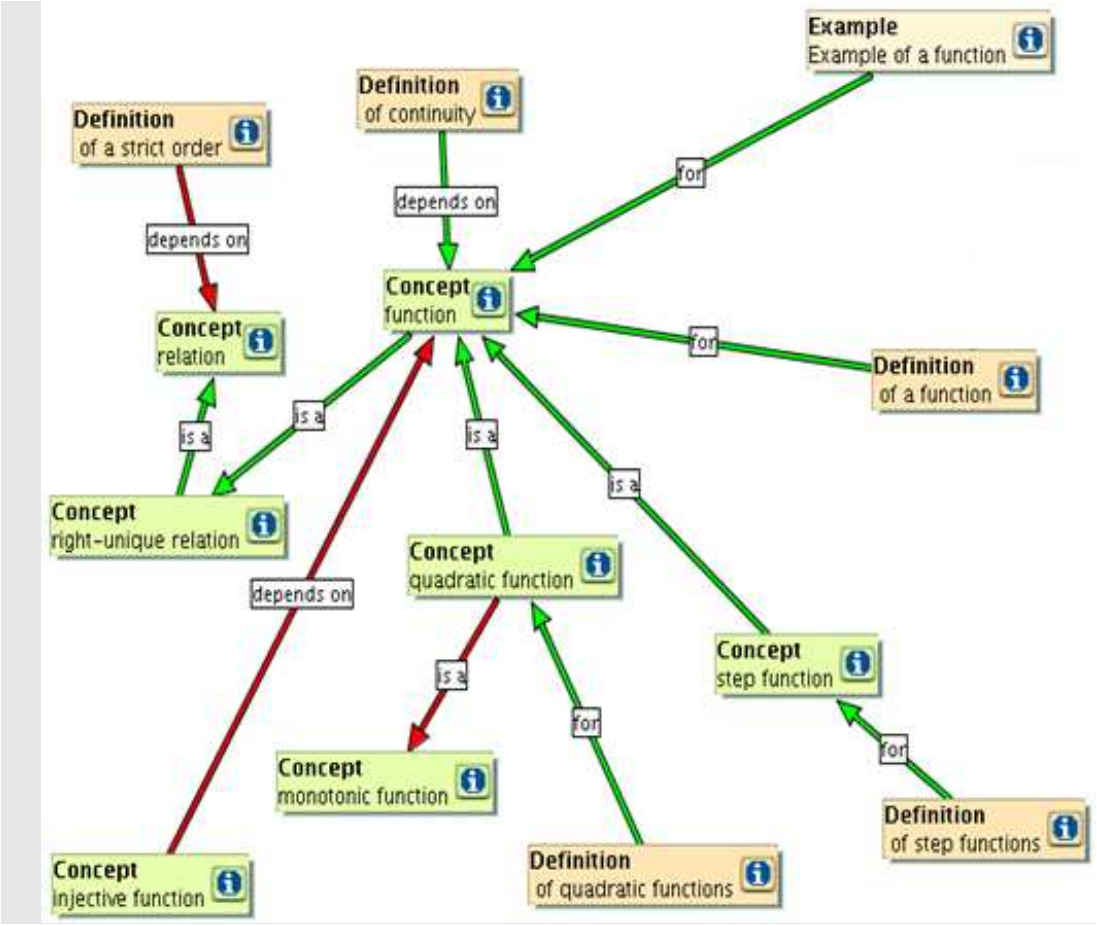
The concept mapping tool iCMap contains restricted versions of propositional reasoning about . For details see Graphical tools.

3.3.3 Graphical Tools and Interfaces

The input for the search for a mathematical term/formula using the graphical input editor WIRIS [WIRIS], a Java application with two−dimensional syntax. The search itself is made on OMDoc objects. For more details see Mathematical Knowledge Bases / Tools for Retrieval in Mathematical Knowledge Bases.

iCMap [MelisEtAl05] is an interactive visual tool, helping students discover relations between mathematical concepts.

Students can play with notions borrowed from the existing ActiveMath courses, creating links between them, and validating them (after the validation, the program colors the correct edges in green, and the incorrect ones in red).



For more details on iCMap see Mathematical Knowledge Bases/ Tools for Verifying Mathe−matical Knowledge.

3.4 User Language

3.4.1 Programming Language

3.4.2 Logic Language for the Formulation of Mathematical Knowledge

3.4.3 Mathematical Syntax

Mathematical knowledge is represented in an extension of the OMDoc format [Kohlhase06].

The building blocks are *content items*, that hold both mathematical expressions and text (that can contain hyperlinks). They are annotated with mathematical and pedagogical attributes and relations.

3.5 Mathematical Knowledge Bases

3.5.1 Available Theories

ActiveMath contains (as stated in [Kohlhase06]) large educational contents for:
Fractions (German)
Differential Calculus (German, English, Spanish) at high-school and first year university level
Operations Research (Russian, English),
Methods of Optimization (Russian)
Statistics and Probability Calculus (German),
Matheführerschein (German)
Calculus (a course from the University of Westminster, London).

3.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Search queries can be combinations of :
Text queries that allow exact phrase searches or fuzzy searches.The latter are based on phonetic and edit−distance fuzziness.
Attribute queries that check whether the mathematical item has certain characteristics. For example,one can search (in the current course) for all items of type ‘‘definition’’, or for all items having the difficulty ‘‘very easy’’.
Mathematical expression queries that are entered by the WIRIS [WIRIS] input editor.

Mathematical expressions are converted into text (linearizing the tree expression by giving depth indication for the components) and then indexed by Lucene [Lucene], a full−featured text−searching library. The actual search in ActiveMath [LibbrechtMelis06] treats both the formal and the informal part of an OMDoc document in the same way. By its ranking heuristic, it sorts the results that the user sees as a list of titles (hyperlinks to the real matches).

3.5.3 Tools for Inventing Mathematical Knowledge

3.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

iCMap [MelisEtAl05] is an interactive concept mapping tool, i.e. a visual representation of a graph whose nodes are abstract symbols, learning objects and theories (as seen in Algorithm Libraries/ Graphical Tools). The edges correspond to relations between nodes. This relations can be :
- relations existing in the underlying OMDoc source (the OMDOC element *theory* has the purpose of gathering notions, definitions and theorems into theories), .
- deductive relations, like *is_equivalent, belongs_to*. Two definitions relating to the same symbol are linked by the *is_equivalent* edge (are considered equivalent) . Also transitivity rules are build in for several types of edge (like *is_a* or *belongs_to).*
- added by the author, as template edges.

The user can construct a concept map and use the validation for all his edges (global verification) or only for one (local verification). The verification is made upon the knowledge base and upon the (independent) authored information existing in the exercise. The user input is matched with the existing ontologies. For example, if the user draws an edge between ***a*** and ***b*** **, iCMap** checks whether there is a relation (in the above meaning) between the object associated with **a** and the object associated with **b.**

In case of failure, explanations are given to the user like "This edge has a wrong type", "This edge is correct, but subsumes several steps. Please elaborate".

3.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

The ActiveMath community uses plain text−editors for editing the textual part of an OMDoc document and jEDITOQMath for editing the mathematical formulae [Libbrechtgross06]. jEDITOQMath takes most of the mathematical LaTeX abbreviations as input and using QMath [QMath] outputs the corresponding OMDoc file.

3.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

3.5.7 Standardization, Inter−Operability

ActiveMath uses knowledge represented in OMDoc, and as such, its repositories are reusable for other applications using the same representation.

3.5.8 Web Access

A demo version of ActiveMath is accessible via the web, through a browser.

3.6 Example of a Theory Exploration Session

ActiveMath is an educational system, and as such does supports learning sessions. These start by logging in and opening a course containing several content items provided by an author. Unknown concepts or symbols can be linked by the authors to their corresponding content item (e.g.their definition) and are thus available to the user. Alternatively, the user can search by the name of a content item. The courses visible to a user are automatically adapted to his learning goals, his knowledge and the previous scenarios. Checking the user exercises is done by evaluating the mathematical input (by a computer algebra system connected to it) and providing correspondingly feedback.

For finding queries composed of text, metadata information (e.g. if the content item is a definition, theorem) and formulae, the user can select the menu "Find" and choose the "Advanced Find". E.g. finding the items that are theorems and that contain the approximate word "convergence" and the term (which is called in ActiveMath formula) $\sum_{i=0}^{n} i^2$. The result will be a

list of links to files with items that are theorems and that contain the word "convergence" and the term $\sum_{i=0}^{n} i^2$ in . The term $\sum_{j=0}^{n} j^2$ will not be identified. (At this moment the formula search capabilities does not work —07.11.2007).

For visualizing the concepts learned and their connections, the user can apply iCMap con−cept maps.

4 Aldor

4.1 Short Description

Aldor is a programming language with an expressive type system well−suited for mathemati−cal computing and which has been used to develop a number of computer algebra libraries [AldorWeb].

4.2 Technical Information on the System

4.2.1 Name of the System and Website

Aldor: A Language for Describing Objects and Relationships.
http://www.aldor.org/

4.2.2 Project Leaders and Group

Leader: Stephen Watt (Ontario Research Centre for Computer Algebra).

Group: Laurentiu Dragan, Oleg Golubitsky, Sandy Huerter, Marc Moreno−Maza, Cosmin Oancea.

4.2.3 Main Publications

S.M.Watt. Aldor. In Handbook of Computer Algebra J. Grabmeier, E. Kaltofen, V. Weispfen·ning (editors), Springer Verlag, Heidelberg 2003. pp. 265−270. ISBN 3−540−65466−6.

Stephen M. Watt, Peter A. Broadbery, Samuel S. Dooley, Pietro Iglio, Scott C. Morrison, Jonathan M. Steinbach, and Robert S. Sutor. AXIOM Library Compiler User Guide. NAG Ltd, Oxford, 1994.

4.2.4 Implementation Language

The compiler for the Aldor language is C (gcc for the Linux environments).

4.2.5 System Availability and Prerequisites

The system is available for non−commercial purposes, under the Aldor Public License Version 2.0. If the sources of the system are downloaded, a gcc compiler is needed.

4.3 Algorithm Libraries

4.4 User Language

4.4.1 Programming Language

The Aldor language

4.4.2 Logic Language for the Formulation of Mathematical Knowledge

4.4.3 Mathematical Syntax

4.5 Mathematical Knowledge Bases

4.5.1 Available Theories and Knowledge Bases

4.5.2 Tools for Retrieval in Mathematical Knowledge Bases

4.5.3 Tools for Inventing Mathematical Knowledge

4.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

4.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

4.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

4.5.7 Standardization, Inter−Operability

The system can compile Aldor code into C and Fortran 77.

4.5.8 Web Access

4.6 Example of a Theory Exploration Session

These examples are taken from [Watt03].

To compute the square root in 6 steps of Newton's method we can write:

```
miniSqrt(x: DoubleFloat): DoubleFloat == {
        r := x;
        r := (r*r + x)/(2.0*r);
        r := (r*r + x)/(2.0*r);
        r := (r*r + x)/(2.0*r);
        r := (r*r + x)/(2.0*r);
        r := (r*r + x)/(2.0*r);
        r := (r*r + x)/(2.0*r);
        r
}
```

This is how a category is defined:

```
define Logic: Category == BasicType with {
        ~:         % -> %;      ++ Logical complement.
        /\:        (%, %) -> %; ++ Logical 'meet', e.g. 'and'.
        \/:        (%, %) -> %; ++ Logical 'join', e.g. 'or'.
        xor:       (%, %) -> %; ++ Exclusive or.
        default (x: %) \/ (y: %): % == ~(~x /\ ~y);
        default xor(x: %, y: %): % == (x /\ ~y) \/ (~x /\ y);
}
```

where [Watt03]:

- A *with* expression forms a category;
- *BasicType* is a previously defined category which this one extends;
- The text beginning with ++ will go into the documentation;
- '%' are later replaced by the domain which belongs to the category. E.g. importing Boolean: Logic will give the operations such as xor: (Boolean, Boolean) −> Boolean;
- *define* makes the value of *Logic* public information. This is necessary in order to know what operations are exported when Logic is used. (Usually only the name and type are public information.)

5 ArXiv

5.1 Short Description

ArXiv is an archive for electronic preprints of scientific papers in the fields of mathematics, physics, computer science and quantitative biology. It can be accessed via Internet and contains over 400.000 papers.

5.2 Technical Information on the System

5.2.1 Name of the System and Website

arXiv (pronounced ’archive’). http://arxiv.org/

5.2.2 Project Leaders and Group

The e−print archive was started in 1991 by Paul Ginsparg. The operation of arXiv is cur−rently funded by the Cornell University and by the National Science Foundation.

5.2.3 Main Publications

McKiernan, Gerry. "arXiv.org: The Los Alamos National Laboratory E−Print Server." The International Journal on Grey Literature 1 (3): 127−138. 2000. http://www.public.iastate.edu/~gerrymck/arXiv.org.pdf

Simeon Warner. Open Archives Initiative protocol development and implementation at arXiv. Expanded version of talk presented at Open Archives Initiative Open Meeting in Washing ton, DC, USA on 23 January 2001. Available on arXiv.org as arXiv:cs/0101027v1.

5.2.4 Implementation Language

Over 30,000 lines of Perl running under Linux, and numerous other programs (TEX, ghost−script, tar, gnuzip, etc.) [Warner01]

5.2.5 System Availability and Prerequisites

Arxhiv.org (including the mirrors) can be freely accessed over the internet.

5.3 Algorithm Libraries

It is a repository of papers. There are no algorithms to be ran.

5.4 User Language

5.5 Mathematical Knowledge Bases

5.5.1 Available Theories and Knowledge Bases

The archive contains over 450.000 e−prints (as of 28 November 2007) from the following domains: Computer Science, Mathematics, Nonlinear sciences, Physics, Quantitative biology, Statistics. Since 2004 there is an endorsement system which is used for submitting papers. I.e. authors that are submitting must be endorsed by authors which already have submitted to arXiv. It is a kind of 'peer−reviewing'.

5.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Search engines via available search pages and portals (for example Front − http://front.math.ucdavis.edu/, Google scholar, Windows Live Academic, eprintweb.org). The searches are textual, by author, title, scientific category, words in abstract or content.

5.5.3 Tools for Inventing Mathematical Knowledge

5.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

5.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

5.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

5.5.7 Standardization, Inter−Operability

5.5.8 Web Access

See the 'Tools for Retrieval ...' subsection above.

5.6 Example of a Theory Exploration Session

6 CoCoA

6.1 Short Description

CoCoA is a program to compute with numbers and polynomials, mainly for computing Gröbner Bases. The CoCoA project began in 1987, as a small Pascal program for Macintosh, experimenting with Buchberger's algorithm for Gröbner bases. In a short time the program was widely used for research and teaching, triggering further implementations, translation into the C programming language, and porting it to other platforms.

An important purpose of the CoCoA program is to provide a "laboratory" for studying computational commutative algebra: it together with Singular and Macaulay 2 form an elite group of highly specialized systems having as their main forte the capability to calculate Gröb− ner bases (citation from [CoCoAWeb]).

6.2 Technical Information on the System

6.2.1 Name of the System and Website

Computations in Commutative Algebra
http://cocoa.dima.unige.it/

6.2.2 Project Leader and Group

Leader: Lorenzo Robbiano (DIMA − Dipartimento di Matematica, Università di Genova, Italy)

Group: John Abbott, Volker Augustin, L. Bazzotti, Anna M. Bigatti, Massimo Caboara, Antonio Capani, G. Dalzotto, A. DelPadrone, S. DeFrancisci, Alessandro Giovini, D. La Mac− chia, Gianfranco Niesi, Dave Perkinson, Alessandro Polverini, F. Rossi, Arndt Willis.

6.2.3 Main Publications

The CoCoATeam. CoCoA: A System for doing Computations in Commutative Algebra. Available at http://cocoa.dima.unige.it/ (last checked on 17.10.2007).

A. Capani and G. Niesi and L. Robbiano. Some Features of CoCoA 3. Comput. Sci. J. of Moldova, 4(3), pages 296−314,1996.

J. Abbott. Challenges in Computational Commutative Algebra. In W. Decker, M. Dewar, E. Kaltofen, and S. Watt, editors, *Challenges in Symbolic Computation Software*, Internationales

Begegnungs− und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

Antonio Capani. The Design of the CoCoA 3 System. PhD Thesis, DISI−TH−2000−02. Universita degli Studi di Genova, Dipartamento di Informatica e Scienze dell'Informazione, February 2000.

See the program's website for further relevant publications.

6.2.4 Implementation Language

C. The new CoCoA system (version 5, not yet released) is implemented in C++.

6.2.5 System Availability and Prerequisites

Free to download from the project's website, under the GPL v3 licence. The system is available for a multitude of platforms: Linux86, Macintosh, SunSolaris, Dec Alpha workstations, Windows, etc).

No prerequisites are needed. If Emacs modes are intended to be used, then Emacs is, of course, required.

6.3 Algorithm Libraries

We could not find a complete list of algorithms implemented in the CoCoA system, therefore, the list below is probably not complete.

6.3.1 Numerical Library

Basic arithmetic algorithms on integer, rationals, modular integers, together with GCD, composition, and factorization.

6.3.2 Algebraic Library

In [Capani00], where CoCoA 3 is described in detail, it is mentioned that the mathematical part of the kernel contains: implementation of basic data structures (coefficients, polynomials, modules); advanced algorithms for computing Gröbner bases, syzygies, rezolution (in CoCoA aka. the "Gröbner Framework"); high level functions like intersection of modules or the Colon operations; powerful special purpose libraries for computing Poincaré series, computation of the solutions of linear algebra problems, management of toric ideals, and an efficient multivariate factorizer.

Additional libraries, written as packages in the CoCoAL language are for:

computing in a cyclic algebraic extension;
computing K−algebra homomorphisms and subalgebras;
applying toric ideals to integer programming;
computing homogeneous generators of an algebra of invariants, and for testing

invariance of a polynomial;
computing the Hilbert−Poincare series of special varieties (Segre, Veronese, Rees);
computing type−vectors associated to Hilbert functions of ideals of points;
package for computing conductor sequence of points;
package for computing normal forms of a matrix, Smith Normal Form (PID)
package for playing Can't Stop and studying strategies;
package for Geometric Control Theory.

6.3.3 Reasoners

There is a package for geometrical theorem−proving in euclidean space (i.e. involving computation).

6.3.4 Graphical Tools and Interfaces

Interacting with the system is made mainly via Emacs (under Linux). For Macintoshes, the CoCoA user interface is based on Mel Park's PlainText (v.1.6). A graphical user interface is also available, where the documents (text files storing CoCoA commands and programs) can be processed, commands executed, etc.

6.4 User Language

6.4.1 Programming Language

The CoCoA system has a full−fledged high level programming language, CoCoAL, which is a Pascal like language. It is complete with loops, branching, scoping of variables, and input/output control [CoCoAMan]. It is strongly typed, each CoCoA expression has a type. There is a partial ordering defined on the types which helps in implementing the notion of subtype.

6.4.2 Logic Language for the Formulation of Mathematical Knowledge

As the system is for computing, one cannot formulate definitions, theorems, etc. like in a book, or like in a proving assistant. The users can formulate (i.e. define, program) algorithms using the CoCoAL language, and compute with them.

6.4.3 Mathematical Syntax

The syntax to formulate algorithms is the one of the CoCoAL, i.e. the programming lan−guage available to the user.

6.5 Mathematical Knowledge Bases

6.5.1 Available Theories and Knowledge Bases

6.5.2 Tools for Retrieval in Mathematical Knowledge Bases

6.5.3 Tools for Inventing Mathematical Knowledge

6.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

6.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

6.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

6.5.7 Standardization, Inter−Operability

6.5.8 Web Access

A general interface between people and CASs (CoCoA in this case) is described in [Capani00, Chapter 6]. The interface is based on the idea of session management, and is imple−mented using the client−server capabilities given by WWW. The user runs some webbrowser with automatic client−server capabilities and submits queries to a machine that runs CoCoA via cgi−applications. We have found no later proof of the further development of this interface, or of its use.

6.6 Example of a Theory Exploration Session

The examples below are taken from the CoCoA manual [CoCoAMan, chapter 2]. The system's output lines can be differentiated from the ones typed in by the user by the fact that they have no semicolon at the end, and a line is printed immediately after them. For readability we have changed the output's font color to blue, but this is NOT a feature of the CoCoA sys−tem. The text following the two dash symbols ('−−') is seen as comments by the system, and therefore not processed. For readability, we have marked the system's answers with '>>'

6.6.1 Some basic examples

Arithmetic with CoCoA

```
(2+3)(1+1);--multiplication,as usual
>>10
------------------------------
2^10;
>>1024
------------------------------
2+2/3;
>>8/3
------------------------------
1.5+2.3;--decimals are converted to fractions
>>19/5
------------------------------
Mod(27,5);
>>2
------------------------------
2*3;
>>6
------------------------------
Fact(4);
>>24
------------------------------
```

Defining variables:

```
A:=3;
2A;
>>6
------------------------------
```

Variable names must start with capital letter, otherwise an error occurs.

```
b:=7;
>>ERROR:parse error in line 12 of device stdin
------------------------------
>>7
------------------------------
B:=7;
A^2+B;
>>16
------------------------------
```

6.6.2 Lists

```
L:=[2,3,‘‘a string’’,[5,7],3,3];--L is now a list
L[3];--here is the 3rd component of L
>>a string
------------------------------
L[4];--the 4th component of L is a list,itself
>>[5,7]
------------------------------
Append(L,‘‘new’’);
L;
>>[2,3,‘‘a string’’,[5,7],3,3,‘‘new’’]
------------------------------
[X^2|X In 1..5];--a useful way to make lists
>>[1,4,9,16,25]
------------------------------
[1,2] >< [3,4] >< [5];--Cartesian product:use a greater-than
--sign ‘‘>’’ and a less-than sign ‘‘<’’ to make
--the operator ‘‘><’’.
>>[[1,3,5],[1,4,5],[2,3,5],[2,4,5]]
------------------------------
```

6.6.3 Rings, Polynomials, Groebner Bases

When a CoCoA session is started, the default ring R = Q[x,y,z] is used automatically. This can be changed with the ’Use’ command. Below we show how to create the Z/(5)[a,b,c] (the coefficient ring of integers mod 5). Once constructed, the user can do various operations within this ring. There are ways to use more than one ring at once.

```
Use S::=Z/(5)[a,b,c];
F:=a-b;
I:=Ideal(F^2,c);
I;
>>Ideal(a^2-2ab+b^2,c)
------------------------------
J:=Ideal(a-b);
I+J;
>>Ideal(a^2-2ab+b^2,c,a-b)
------------------------------
Minimalized(It);--find a minimal set of generators for I+J
>>Ideal(a-b,c)
```

Let r be a root of the equation $x^7 - x - 1$ over the rationals. The minimal polynomial of $(4r - 1)/r^3$ can be found by computing the reduced Groebner basis of the ideal $(x^7 - x - 1,\ x^3 y - 4x + 1)$ with respect to the lexicographic term-ordering with $x > y$. The computed Groebner basis has two elements, the second one being a univariate polynomial which is the minimal polynomial for *r*.

```
Use R::=Q[x,y],Lex;
Set Indentation;--to improve the appearance of the output
G:=GBasis(Ideal(x^7-x-1,x^3y-4x+1));
G;
```

```
>>[160281875715209075944O/3452460823618119936lx−4457540/5875764481y^7
    − 47746460716124220/3452460823618119936ly^6+
    890175715271333840/3452460823618119936ly^5−
    1992534667352220/3452460823618119936ly^4−
    5594389451313946416O/3452460823618119936ly^3−
    5647365436133328098O/3452460823618119936ly^2−
    2797197971202545304O/3452460823618119936ly−

400704689288022689860/3452460823618119936,1/16384y^7−5/16384y^6
    +147/16384y^4+5/128y^3−31/16384y^2+17/128y−20479/16384 ]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Len(G);
>>2
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
F:=16384*G[2];−−clear denominators
F;
>>y^7−5y^6+147y^4+640y^3−31y^2+2176y−20479
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

Factoring a ring can be done as below:

```
Use R::=Q[x,y];
F:=x^12−37x^11+608x^10−5852x^9+36642x^8−156786x^7+−984128x^5+1437157x^4−1422337x^3+905880x^2−333900x
Factor(F);
>>[[x−2,1],[x−4,1],[x−6,1],[x−3,2],[x−5,3],[x−1,4]]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
F:=(x+y)^2*(x^2y+y^2x+3);
F;
>>x^4y+3x^3y^2+3x^2y^3+xy^4+3x^2+6xy+3y^2
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Factor(F);−−multivariate factorization
>>[[x^2y+xy^2+3,1],[x+y,2]]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Use Z/(37)[x];
>>Factor(x^6−1);[[x−1,1],[x+1,1],[x+10,1],[x+11,1],[x−11,1],[x−10,1]]
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

6.6.4 User−Defined Functions

The syntax of a user defined function is:

```
Define<FunctionName>(<argument list>)<Commands>EndDefine;
```

For example, a function that tests wether a given parameter is prime or not looks like:

```
Define IsPrime(X)−− a more complicated function
    If Type(X)<>INT Then Return Error(‘‘Expected INT’’) EndIf;
    I:=2;
    While I^2≤X Do
        If Mod(X,I)=0 Then Return False EndIf;
        I:=I+1;
```

```
    EndWhile;
    Return TRUE;
EndDefine;--end of function definition
```

A test execution of the above defined function:

```
IsPrime(4);
>>FALSE
```

7 Coq

7.1 Short Description

Coq is a proof assistant for the logical framework known as the Calculus of Inductive Constructions. The Coq project started in 1984 as an implementation of the Calculus of Construc tions, at INRIA−Rocquencourt. The originators of the project were Thierry Coquand and Gérard Huet. The first release of the system was made in 1989, and in 1991 was extended to the Calculus of Inductive Constructions. Nowadays, the system is developed under the coordination of the LogiCal project [LogiCal].

On their website, the Coq team state that "Coq is a formal proof management system". Proofs done with the system are checked mechanically, by a relatively small kernel. The system also comes with a functional programming language, provides both interactive proof methods and (semi)decision algorithms, and also has a protocol for connecting to external computer algebra systems or theorem provers [CoQWeb].

7.2 Technical Information on the System

7.2.1 Name of the System and Website

The Coq proof assistant. http://coq.inria.fr/

In French, "coq" means rooster, and it sounds like the initials of the Calculus of Construc− tions CoC on which it is based. (In fact, the first implementation *was* named CoC).

7.2.2 Project Leader and Group

The development of the system is coordinated by the LogiCal project [LogiCalWeb]. The leader of this project is Giles Dowek (Laboratoire d'Informatique (LIX), École polytechnique, France).

As the project has already a long history of development, the project group has modified along the years. As of 2006, the people that contributed to the last release are (according to their reference manual [CoQMan]): Nicolas Ayache, Bruno Barras, Yves Bertot, Pierre Castéran, Jacek Chrzaszcz, Claudio Sacerdoti Coen, Pierre Corbineau, Pierre Courtieu, Jean−Christophe Filliâtre, Julien Forest, Benjamin Grégoire, Hugo Herbelin, Pierre Letouzey, Assia Mahboubi,

Benjamin Monate, Julien Narboux, Jean−Marc Notin, Christine Paulin, Laurent Théry, Mat−thieu Sozeau.

7.2.3 Main Publications

The CoQ Reference Manual. http://coq.inria.fr/doc/toc.html (last checked on 10.10.2007)

Y. Bertot and P. Castèran. *Interactive Theorem Proving and Program Development: Coq'−Art: The Calculus of Inductive Constructions,* ISBN 3−540−20854−2, Springer, 2004.

7.2.4 Implementation Language

Written in Objective Caml (with a bit of C) [OCaml].

7.2.5 System Availability and Prerequisites

Distributed under the GNU Lesser General Public Licence Version 2.1 (LGPL).

The system can be downloaded form the Coq website. Binaries are available for various operating systems (Linux, FreeBSD, MacOsX, Windows). If users opt for the system's sources they need to compile them with Objective Caml.

7.3 Algorithm Libraries

The Coq system does not have libraries of numerical or symbolic algorithms implemented in an execution−efficient way, as, for example, are implemented in CASs.

7.3.1 Reasoners

The Coq's reasoner can be called either in interactive mode or proof editing mode.

The logical reasoning within Coq is done via *tactics* (built from atomic tactics and tactic expressions, or tacticals, i.e. tactic combinators). There are, at least, three levels of atomic tactics. The simplest one implements basic rules of the logical framework. The second level is the one of derived rules which are built by combination of other tactics. The third one imple−ments heuristics or decision procedures to build a complete proof of a goal [CoQTut]. Before a tactic is applied to a goal, the system checks whether some preconditions are satisfied.

During proof development, at each stage, there is a list of goals to prove, which at the beginning has only one element: the proposition to prove itself. The list of goals is updated with each tactic application. Each goal has associated a local context — a number of hypotheses — which is empty at the beginning.

Finished proofs can be stored as defined constants in the environment (because of the Curry−Howard isomorphism). Coq stores proofs as terms (i.e. proof terms) of Calculus of Inductive Constructions.

The tactics in the Coq system cover areas like of propositional and predicate logic, equality (Leibnitz equality), induction (on booleans, natural numbers, on functions defined by primitive recursion, inductive families). Generalized pattern matching is also supported (see [CoQMan], Addendum to the Reference Manual).

There are tactics which, when called, try to do more than one proof step without user interference. The most used is the *auto* tactic, which implements a Prolog−like resolution procedure to solve the current goal. By default, *auto* uses the local context of the current goal and the hints of the database named *core*. The power of the *auto* tactic can be increased by adding lemmata/theorems as hints to be used by the *auto* tactic.

Coq contains also other specific tactics, like:

— *congruence* which implements a decision procedure for ground equalities with uninterpreted symbols.

— *omega*, which is an automatic decision procedure for Presburger arithmetic.

— *ring*, which solves equations upon polynomial expressions of a ring (or semi−ring) structure.

— *fourier*, for solving linear inequations on real numbers using Fourier's method.

— *autorewrite* for rewriting according to given rewriting rule bases.

For a detailed description of the system's tactics and tactical language see [CoQMan].

7.3.2 Graphical Tools and Interfaces

CoqIde and *Pcoq* are two graphical user interfaces to Coq. In *CoqIde* the user input is given in one window and Coq's answers are displayed in a different window. *Pcoq* offers multiple fonts and colors to display formulas and commands, provides a way to structurally edit formulae and commands, makes adding new notation easy. It also supports the 'proof by pointing' method [BertotKahnThery94].

Other graphical interfaces for Coq are ProofGeneral (a generic interface for proof systems based on Emacs) and Emacs modes.

7.4 User Language

7.4.1 Programming Language

Gallina is the specification language of Coq. It is used both to develop formalizations and to prove specifications of programs [CoQMan]. The language combines predicate calculus, prolog and recursion equations in a logically sound way [Huet92]. See also the subsection below.

For writing new tactics one can use the LTac language [Delahaye00], or the Objective Caml language, the language in which the Coq system is written.

7.4.2 Logic Language for the Formulation of Mathematical Knowledge

The language used to develop formal axiomatizations is called Gallina. Such a development consists of a sequence of *declarations* and *definitions*. During a formalization session, the user can call Coq commands which are not part of the formal development, but are used for informa−tion requests (for example). [CoQMan]

A declaration associates a *name* (identifier) with a *specification*. Basically, there are three kinds of specifications: logical propositions, mathematical collections, and abstract types [CoQMan].

See some examples in the "Example of a Theory Exploration Session" section below.

7.4.3 Mathematical Syntax

The syntax used is the syntax of Gallina. See the "Programming Language" and "Logic Language for the Formulation of Mathematical Knowledge" subsections above.

7.5 Mathematical Knowledge Bases

7.5.1 Available Theories and Knowledge Bases

Coq's standard library contains the following [CoQWeb]:

* Init: The core library (automatically loaded when starting Coq)
* Logic: Classical logic and dependent equality
* Arith: Basic Peano arithmetic
* NArith: Binary positive integers
* ZArith: Binary integers
* QArith: Rational numbers
* Reals: Formalization of real numbers
* Sets: Sets (classical, constructive, finite, infinite, powerset, etc.)
* Relations: Relations (definitions and basic results)
* Wellfounded: Well−founded Relations
* Setoids
* Bool: Booleans (basic functions and results)
* Lists: Polymorphic lists, Streams (infinite sequences)
* FSets: Modular implementation of finite sets/maps using lists
* IntMap: An implementation of finite sets/maps as trees indexed by addresses
* Strings Implementation of string as list of ascii characters
* Sorting: Axiomatizations of sorts.

There are several contributions to the library made by users [CoQWeb] in Mathematics (Logic, Algebra, Real Analysis and Topology, Geometry, Arithmetic and Number Theory, Combinatorics and Graph Theory, Category Theory), Computer Science (Lambda−Calculi et al, Formal Languages Theory and Automata, Decision Procedures and Certified Algorithms, etc.),

Logical Puzzles and Entertainment. For a complete list and a description of these libraries see the project's website.

Another significant corpus of formalized mathematics is stored into the C−CoRN library (The Constructive Coq Repository at Nijmegen) [C−CoRNWeb]. Because it is independent from the Coq system, we treat this library as a separate entity in the chapter about C−CoRN.

7.5.2 Tools for Retrieval in Mathematical Knowledge Bases

For loaded libraries (i.e. knowledge is available in the local context), some Coq search commands are provided. For example, *Search* and *SearchAbout* look for name and type of all theorems and objects in the current context, respectively, whose statements contain a given identifier. *SearchPattern* and *SearchRewrite* involve patterns in their search for theorems, *Locate* displays the full name of the given identifier, together with the name of the module (file) where it is defined.

Another search tool for the Coq standard library *and* the user contribution library is *Whelp*. This is an experimental searching and browsing tool and requires a browser to work. Whelp was developed as part of the HELM and MoWGLI projects, at the University of Bologna [AspertiEt−Al04]. The Whelp search commands — given in Coq top−level or CoqIde — will open a browser window displaying the search results. The search expression can involve regular expressions (when searching for names), patterns (where holes in the pattern are represented by the underscore character), valid identifiers, or terms (when searching for statements that can be used to prove the term by instantiation). See [AspertiEtAl04] for more on this search tool.

7.5.3 Tools for Inventing Mathematical Knowledge

7.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The Coq's reasoner checks every piece of mathematical knowledge that is worked on with the system. Theorems can be added to the local context only if they have a successful proof (i.e. a correct type). The contributions to the library also must be successfully checked by Coq.

7.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

7.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

LTac, the tactic language of Coq, allows users to write their own tactics. The tactics are checked by the Coq type checker which verifies that the term built by a tactic is of the theorem type the user wants to apply it to [Delahaye00].

Additionally, the system has a *Quote* tactic, which is used to do proofs by reflection [Co−QMan, 10.7].

7.5.7 Standardization, Inter−Operability

There is a Coq package, written by David Delahaye and Micaela Mayero, that imports Maple computation tools. It exports, from Maple, the functions simplify, factor, expand, normal and provides Coq tactics with the same names (i.e. Simplify, Factor, etc.). The manipulations done by these (imported) tactics are proved with the *Field* tactic [CoQMan, 20]. Given a term *t*, if *t'* is the simplified/factorized term, in order to replace *t* by *t'* one must prove that *t = t'*, which is done by the *Field* tactic. For this reason, for the term *t* to Simplify, Factorize, etc., a field theory must be declared for its type.

There is an OpenMath phrasebook for the Coq library, but we've found no reference to a publication about this.

7.5.8 Web Access

The Coq library can be browsed on−line on the Helm website (http://helm.cs.unibo.it/). A web interface to Coq has been implemented by Loïc Pottier and is available at http://wims.unice.fr/wims/wims.cgi?module=U3/logic/logicoq . Using this interface, one can insert the content of a Coq file and submit it for verification. It is only usable for small problems.

Another web based GUI to CoQ is described in [Kaliszyk06] and can be tried at http://prover.cs.ru.nl/ .

7.6 Example of a Theory Exploration Session

Where not otherwise stated, the examples are taken from [CoQTut]. The system outputs are marked with '>>'

7.6.1 Basics

Starting the system (command line)

```
unix:~>coqtop
>> Welcome to Coq 8.0 (Mar 2004)
>> Coq<
```

Leaving the system is done with the *Quit* command.

Declare a natural number *n*:

```
Coq<Variable n:nat.
>> n is assumed
```

Let's declare that *n* is positive:

```
Coq<Hypothesis Pos_n:(gt n 0).
>> Pos_n is assumed
```

7.6.2 Simple Logic Proof

Declare some variables of type Prop:

```
Coq<Variables A B C:Prop.
>> A is assumed
>> B is assumed
>> C is assumed
```

Let us now prove that (A −> B −> C) −> (A −> B) −> A −> C. This is done by using the command *Goal* followed by the conjecture we want to verify:

```
Coq<Goal (A→B→C)→(A→B)→A→C.
>> 1 subgoal

>> A:Prop
>> B:Prop
>> C:Prop
>>   ============================

>>     (A→B→C)→(A→B)→A→C
```

Now we find ourselves in the proof mode, with the goal displayed under the line. Tactics are now available to the user. For instance, the *intro* tactic is applicable to any goal which is an implication, by moving the left part of the implication to the list of local hypotheses:

```
Coq<intro H.
>> 1 subgoal

>> A:Prop
>> B:Prop
>> C:Prop
>> H:A→B→C
>>   ============================

>>     (A→B)→A→C
```

Several introductions may be done in one step:

```
Coq<intros H' HA.
>> 1 subgoal

>> A:Prop
>> B:Prop
>> C:Prop
>> H:A→B→C
>> H':A→B
>> HA:A
>>   ============================

>>   C
```

The current goal, may be obtained from hypothesis H, provided that A and B are true. We use the *apply* tactic, and we end with two subgoals (the current goal has the local context, i.e. local assumptions, listed in full):

```
Coq<apply H.
>> 2 subgoals

>> A:Prop
>> B:Prop
>> C:Prop
>> H:A→B→C
>> H':A→B
>> HA:A
>>   ============================

>>   A
>> subgoal 2 is:

>>     B
```

The current goal is available as the HA hypothesis:

```
Coq<exact HA.
>> 1 subgoal

>> A:Prop
>> B:Prop
>> C:Prop
>> H:A→B→C
>> H':A→B
>> HA:A
>>   ============================

>>   B
```

Now H' applies:

```
Coq<apply H'.
>> 1 subgoal

>> A:Prop
>> B:Prop
>> C:Prop
>> H:A→B→C
>> H':A→B
>> HA:A
>>   ============================

>>   A
```

And we may now conclude the proof as before, with *exact* HA. We can also just state that the current goal is solvable from the current local assumptions:

```
Coq<assumption.
>> Proof completed.
```

The proof is now finished. We may either discard it, by using the command *Abort* which returns to the standard Coq top−level loop (we remind here that we were in a proof mode) without further ado, or else save it as a lemma in the current context, under name say trivial_lemma:

```
Coq<Save trivial_lemma.
>> intro H.
>> intros H' HA.
>> apply H.
>> exact HA.
>> apply H'.
>> assumption.
>> trivial_lemma is defined
```

7.6.3 Simple Search

```
Coq<Require Import ZArith.
Coq<SearchAbout[Zmult Zplus "distr"].
>> weak_Zmult_plus_distr_r:
>>  forall (p:positive) (n m:Z),
>>  (Zpos p*(n+m))%Z=(Zpos p*n+Zpos p*m)%Z
>> Zmult_plus_distr_r:
>>  forall n m p:Z,(n*(m+p))%Z=(n*m+n*p)%Z
>> Zmult_plus_distr_l:
>>  forall n m p:Z,((n+m)*p)%Z=(n*p+m*p)%Z
>> OmegaLemmas.fast_Zmult_plus_distr_l:
>>  forall (n m p:Z) (P:Z→Prop),
>>  P (n*p+m*p)%Z→P ((n+m)*p)%Z

Coq<Require Import Arith.
Coq<SearchPattern (_+_=_+_).
>> plus_comm: forall n m:nat,n+m=m+n
>> plus_Snm_nSm:forall n m:nat,S n+m=n+S m
>> plus_assoc: forall n m p:nat,n+(m+p)=n+m+p
>> plus_permute: forall n m p:nat,n+(m+p)=m+(n+p)
>> plus_assoc_reverse:forall n m p:nat,n+m+p=n+(m+p)
>> plus_permute_2_in_4:forall n m p q:nat,n+m+(p+q)=n+p+(m+q)
```

A pattern that requires that the same expression must occur on two different places in an expression:

```
Coq<SearchPattern (?X1 + _ = _ + ?X1).
>> plus_comm: forall n m:nat,n+m=m+n
```

Show the full name of the given identifier (*nat*) and (consequently) the Coq module in which is definded.:

```
Coq<Locate nat.
>> Inductive Coq.Init.Datatypes.nat
```

8 C−CoRN

8.1 Short Description

The Constructive Coq Repository at Nijmegen, C−CoRN, aims at building a computer based library of constructive mathematics, formalized in the theorem prover Coq [C−CoRNWeb]. It aims at being a test−bed for investigating the process of formalizing mathematics and how digitally stored formalized mathematics can be interacted with, managed. The knowledge in this library is formalized constructively, because of practical reasons (the authors are well acquainted with the Coq system) and because constructive mathematics offers executable algorithms for the functions that are proved to exist. The project grew out of the FTA project (the Fundamental Theorem of Algebra project) whose achieved goal was to formalize the constructive proof of the FTA theorem, together with all the knowledge needed for this proof.

8.2 Technical Information on the System

8.2.1 Name of the System and Website

The Constructive Coq Repository at Nijmegen.
http://c−corn.cs.ru.nl/index.html

8.2.2 Project Leaders

Henk Barendregt and Herman Geuvers (Foundations of Mathematics and Computer Science, Radboud University, Nijmegen, The Netherlands).

The following people have, until now, contributed to the C−Corn library: Henk Barendregt, Vince Bárány, Luís Cruz−Filipe, Herman Geuvers, Mariusz Giero, Rik van Ginneken, Dimitri Hendriks, Sébastien Hinderer, B. W. M. Kirkels, Pierre Letouzey, Iris Loeb, Lionel Mamane, Milad Niqui, Russell O'Connor, Randy Pollack, Nickolay V. Shmyrev, Bas Spitters, Dan Synek, Freek Wiedijk, Jan Zwanenburg.

8.2.3 Main Publications

L. Cruz−Filipe. Constructive Real Analysis: a Type−Theoretical Formalization and Applica−tions, PhD Thesis. University of Nijmegen, April 2004.

Other, project related publications deal with formalizing mathematics and not the system description. A list of these publication can be seen at the project's website (http://c−corn.cs.ru.nl/pub.html).

8.2.4 Implementation Language

None. The library is a collection of text files.

8.2.5 System Availability and Prerequisites

The library can be freely downloaded from the website.

None mentioned, but as it is used together with the Coq system, the same system prerequisites as for Coq are to be considered.

8.3 Algorithm Libraries

8.4 User Language

We only note here that the library was created using the Coq system, using the tactics defined there. The authors of the library also created new Coq tactics using the LTac language available in Coq. See the section on the Coq system description.

8.5 Mathematical Knowledge Bases

We only list here the theories and knowledge bases available in the system. For the (possibly) available tools for mathematical knowledge management see the section on CoQ.

8.5.1 Available Theories and Knowledge Bases

The C−CoRN library has over 3000 formalized lemmas and over 800 definitions. The available theories are (according to their website):

Algebraic Hierarchy: An axiomatic formalization of the most common algebraic structures, including setoids, monoids, groups, rings, fields, ordered fields, rings of polynomials, real and complex numbers.

Model of the Real Numbers: Construction of a concrete real number structure satisfying the previously defined axioms.

Fundamental Theorem of Algebra: A proof that every non−constant polynomial on the complex plane has at least one root.

Real Calculus: A collection of elementary results on real analysis, including continuity, differentiability, integration, Taylor's theorem and the Fundamental Theorem of Calculus.

8.5.2 Web Access

The library can be browsed on−line on the Helm website (see the chapter on Helm of this report).

8.6 Example of a Theory Exploration Session

See the section on Coq.

9 DLMF

9.1 Short Description

DLMF or the **Digital Library of Mathematical Functions** [MillerYoussef03] intends to be the (online and printed) successor of Abramowitz and Stegun' s ''Handbook of Mathematical Functions'' [AbramowitzStegun65], a standard reference for engineers throughout several decades. The project, headed by the National Institute of Standards and Technology (NIST), creates documents coded in LaTeX, using special macros for producing a handbook on paper as well as a dynamic website.This website will provide mathematical formulae, 3D interactive graphics, methods of computation, references, links to software, and an equation search capability.

9.2 Technical Information on the System

9.2.1 Name of the System and Website

DLMF (Digital Library of Mathematical Functions)
http://dlmf.nist.gov

9.2.2 Project Leaders and Group

DLMF is an international project, with members from USA and European Universities, and it does not have a general project leader.

General Editor:
Dan Lozier, Mathematical and Computational Sciences Division, ITL , NIST.
Mathematics Editor:
Frank Olver, Institute for Physical Science and Technology, University of Mary−land, College Park, Maryland.
Physical Sciences Editor:
Charles Clark, Electron and Optical Physics Division, Physics Laboratory, NIST.
Information Technology Editor:
Ron Boisvert, Mathematical and Computational Sciences Division, ITL, NIST.
Associate Editors:
Associate Editors for Special Functions
Dick Askey,University of Wisconsin,Madison.
Nico Temme,CWI,Amsterdam.
Associate Editors for Physics

Michael Berry,University of Bristol,U.K.
Leonard Maximon,George Washington University.
Associate Editor for Chemistry
Bill Reinhardt,University of Washington,Seattle.
Associate Editor for Combinatorics and Number Theory
Morris Newman,University of California,Santa Barbara.
Associate Editor for Computer Algebra
Peter Paule,Johannes Kepler University,Linz
Associate Editor for Statistics
Ingram Olkin,Stanford University.

9.2.3 Main Publications

Bruce Miller and Abdou Youssef, Technical Aspects of the Digital Library of Mathematical Functions, Annals of Mathematics and Artificial Intelligence, Volume 38, pp. 121−136, 2003.

Qiming Wong and Bonita Saunders, Web−Based 3D Visualization in a Digital Library of Mathematical Functions, in Proceedings of Web3D 2005 (10th International Conference on 3D Web Technology), ACM SIGGRAPH, 2005.

9.2.4 Implementation Language

For the 3D graphics of DLMF the Virtual Reality Modeling Extensible 3D Language (VRML/X3D) has been used. All other functionalities seem to be programmed in Java.

9.2.5 System Availability and Prerequisites

Mock−up available online. By 2008, the whole project will be available online.
For viewing the VRML graphics, one needs a VRML browser such as BS Contact, Cortona, Cosmoplayer.

9.3 Algorithm Libraries

The user has no direct contact to an algorithm library.

9.3.1 Reasoners

9.3.2 Graphical Tools and Interfaces

Most chapters of the DLMF will have a "Graphs and Visualization" section with 2D and 3D figures. The user will interact with the 3D images by using **MathViewer** [WongSaunders05]. MathViewer is a VRML/3D application that allows on top of customary VRML browser con−trols (rotate, zoom, pan that permit the examination of a 3D object from an arbitrary direction) also more advanced capabilities. The three main controls are:

Cutting Plane Control: helps the user examine the intersection of a plane with a function surface as the plan moves through the surface.

Axis and Labels Control: the user can select the axis and labels preferred.

Color Map Control: used for manipulating the colors of the visualizations of the real-valued or complex-valued functions.

Scale Control: that manages the screen length of the surface in the x,y and z direction.

Also several viewpoints of the objects are available: "back", "right", "front", "left" and "top".

For several examples, please check http://ovrt.nist.gov/projects/vrml/dlmf/ (last checked on 26.11.07).

9.4 User Language

9.4.1 Programming Language

9.4.2 Logic Language for the Formulation of Mathematical Knowledge

It is not clear which logical language is used.

9.4.3 Mathematical Syntax

The mathematical formulae are stored, for now, in LaTeX(according to their website). There are two versions for presenting them on the web:

- An HTML version that is generated by a combination of hand written HTML and HTML generated from LaTeX (using images for mathematical formulae or terms) by a customized version of LaTex2HTML.

- A XML/MathML version that is generated by a translator (LaTeXML) conceived and implemented as part of the DLMF project. The mathematical formulae here are represented by content MathML. This translation is a semi-automatical one, most ambiguities in the sources can be eliminated by augmenting the LaTeX source with e.g. declarations of variable types.

9.5 Mathematical Knowledge Bases

9.5.1 Available Theories and Knowledge Bases

Mathematical functions from the following theories are included:

Algebraic and Analytic Methods
Asymptotic Approximations
Numerical Methods
Elementary Functions
Gamma Function
Exponential Integral, Logarithmic Integral, Sine and Cosine Integrals
Error Functions, Dawson’ s Integral, and Fresnel Integrals
Incomplete Gamma and Related Functions
Airy and Related Functions
Bessel Functions
Struve and Related Functions
Confluent Hypergeometric Functions
Parabolic Cylinder Functions
Legendre Functions and Associated Legendre Functions
Hypergeometric Function
Generalized Hypergeometric Functions and Meijer G − Function
q − Hypergeometric Functions
Orthogonal Polynomials
Elliptic Integrals
Theta Functions
Riemann Theta Functions
Jacobian Elliptic Functions
Weierstrass Elliptic and Modular Functions
Bernoulli and Euler Numbers and Polynomials
Zeta and Related Functions
Functions of Number Theory
Combinatorial Analysis
Mathieu Functions and Hill’ s Equation
Lamé Functions
Spheroidal Wave Functions
Heun Functions
Painlevé Transcendents
Functions of Matrix Argument
Coulomb Wave Functions
3 j, 6 j, 9 j Symbols
Integrals with Coalescing Saddles
Statistical Methods & Distributions
Computer Algebra

9.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The **equation search capability** in DLMF is understood as a layer on top of textual search. Mathematical information is preprocessed in the following way [Youssef05, Youssef04]:

- *Textualization*: The mathematical information is transformed into structured text.
For example, $x^a + y_b - t$ is encoded as x super a endsuper plus y sub b endsub minus t.
- *Flattening and scoping*: The scope of quantifiers (linguistic constructs that bind variables) is made explicit and their (typically two-dimensional) syntax is linearized as in the previous step.
For example, $\int_0^1 y \, \mathrm{dy}$ becomes integral sub 0 endsub super 1endsuper integrand y endinte−grand dy.
- *Normalization*: The mathematical equation is brought into a normal form with respect to certain algebraic and notational rules.
For example, $\mathrm{a\,b^{-1}\ c\,d^{-1}}$ has the normal form frac {ac} {bd}, and x_ 2^3 and x^3 _ 2 both have the normal form x sub 2 endsub super 3 endsuper.

The preprocessed mathematical information is then indexed by the underlying text search system. A **query** (input in a simplified LaTeX-like query language) goes through the same process as above and triggers a search in the index [Youssef07, YoussefShatnawi06].
Besides this equation search, one can also search for keywords like ‘‘Bessel Function’’.

9.5.3 Tools for Inventing Mathematical Knowledge

9.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

9.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

9.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

9.5.7 Standardization, Inter−Operability

The DLMF project is a web-based project, as such can be accessed easily over the internet (Its graphics, though, require the use of VRML-capable browsers).

9.5.8 Web Access

For now, a mock−up of DLMF is available online at the webpage mentioned above.

9.6 Example of a Theory Exploration Session

The user of DLMF can browse the whole library of mathematical functions available or can search for a function, either by its name "Bessel function" or by an equation (for more details see Mathematical Knowledge Bases / Tools for Retrieval in Mathematical Knowledge Bases). Once the function is found, one can see the actual formula and its properties (where it is referenced, informations about its components) and in several cases 2D or 3D visualizations of the function with MathView (see Mathematical Knowledge Bases/Graphical Tools).

For example, Euler's Beta function $B(a, b) = \int_0^1 t^{a-1}(1-t)^{b-1}\,\mathrm{dt} = \frac{\Gamma(a)\,\Gamma(b)}{\Gamma(a+b)}$ includes among its properties:

Notations:

B(a,b): Beta function,

Γ(z): Gamma function (with a link to the file containing the definition of Gamma function),

a: real or complex variable and b: real or complex variable

(with a link to the file containing the definition of real variable and complex variable),

Referenced By: the link to the formulae that are using the function (in this case Euler Beta Integral)

The webpage is not yet fully functional online, so we can not give a more detailed example.

10 FDL

Note: It is recommended that the chapter on Nuprl is read before this one.

10.1 Short Description

The FDL project aims at creating a digital library of algorithms and constructive mathematics usable for program and software construction. It has, therefore, two complementary goals: a) create a basic logical infrastructure for a global digital library (a Logical Library), and b) create new formal computational content using the logical library.

The library contains definitions, theorems, theories, proof methods, and articles about topics in computational mathematics, as well as books assembled form all these. The objects stored are created with the proving systems MetaPRL [Metaprl], Nuprl [NuPRLBook] and PVS [OwreEtAl96].

Currently, only an FDL prototype is available.

10.2 Technical Information on the System

10.2.1 Name of the System and Website

Formal Digital Libraries. http://www.nuprl.org/FDLproject/

10.2.2 Project Leaders and Group

The leader of the project is Robert L.Constable (Dean of Faculty of Computing and Informa−tion Science, Computer Science Department, Cornell University).

The group working on the FDL project consists of: Stuart Allen, Rich Eaton, Christoph Kreitz, Lori Lorigo, Eli Barzilay, Alexei Kopylov, Jason Hickey, Xin Yu, James Caldwell, John Cowles, Vitali Khaikine, Christoph Jechlitschek.

10.2.3 Main Publications

Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. *FDL: A Prototype Formal Digital Library − Description and Draft Reference Manual.* Technical report, 16 June 2004. Department of Computer Science, Cornell−University, Ithaca, NY, 14853−7501.

10.2.4 Implementation Language

The FDL library is implemented as a distributed system and as a transactional database of formal mathematics using the notion of abstract object identifiers. The implementation language is (as in the Nuprl case) Lisp and ML.

10.2.5 System Availability and Prerequisites

Same as Nuprl (see the corresponding chapter).

10.3 Algorithm Libraries

10.4 User Language

As a system that has emerged from Nuprl, see the corresponding part of the chapter on Nuprl for more details.

10.5 Mathematical Knowledge Bases

10.5.1 Available Theories and Knowledge Bases

The contents of the FDL library are represented by a common basic data structure: *objects*. Objects are abstract terms that are associated with a *kind*, a variety of *properties*, and possibly with *extra data* [AllenEtAl02].

In addition to the theories which come from the Nuprl system (see the respective chapter of this document), the system contains libraries from the PVS system (the theories from PVS prelude, the complete libraries on bitvectors, finite sets, arrays, number and graph theory, real analysis, etc. − status as of 2003), and MetaPRL [Metaprl]. FDL also contains a collection of formal algorithmic knowledge − algorithms together with proofs of their correctness (see the website for more details).

10.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Only basic kind of information retrieval are now implemented [AllenEtAl02]:

- − Search for lemmata containing a specified list of object names;
- − Search for objects modified within a given time specification
- − Search for objects by name

10.5.3 Tools for Inventing Mathematical Knowledge

10.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The FDL prototype system is currently connected to the reasoners and inference engines of Nuprl, JProver [SchmittEtAl01], and PVS [OwreEtAl96]. The connection is done via API (Application Programming Interface) modules. The intention is that more and more proof engines will be connected to the library.

10.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

In the FDL prototype there are a number of operations that allow structuring and re−structur−ing of mathematical knowledge:

- − Arrange objects in folders and theories;
- − Move and rename object;
- − Create links to objects;
- − Deactivate and re−activate objects;
- − Remove objects and links.

Among the goals of the project mentioned in the project’s description more elaborate library operations are mentioned to be of use:

- Pruning the theories around main theorems;
- Mining a theory for new relationships, connections and proof methods;
- Finding dependencies among theories.

The web interface to the system (see the 'Web Access' subsection below) also does information analysis, creating articles for the web presenting the formal content at different levels of detail.

10.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

As a system that has emerged from Nuprl, FDL has means for programming tactics (meta−programming) and do proofs about them. See the Nuprl chapter.

10.5.7 Standardization, Inter−Operability

Supports representation of terms in XML. It can also communicate with various proof engines via the MathBus standard [MathBus]. Another data format supported by FDL is the compressed ASCII format.

10.5.8 Web Access

There is a website where users could browse the content of the library, however, at every try over a period of time, the respective pages have returned 'Page not found' errors. (http://www.nuprl.org/FDLProject/fdl_online.html last checked 21.11.2007).

10.6 Example of a Theory Exploration Session

An FDL session starts with five windows: a navigator , an ML top−loop, a window for the library, a refiner, and the editor process [Kreitz03]. The user interacts with the library via the navigator window which has three parts: A command zone which contains buttons (see [Kreitz03] for a detailed description of these buttons); A statistics zone which displays directory statistics; A navigation zone which displays a linear portion of the library, one object per line.

The ML Top Loop window provides a command interface to the editor process, refiner process, and library process.

In spirit, the FDL session windows are very similar with the Nuprl set of windows. For a detailed on browsing and manipulating the formal content of FDL see [Kreitz03].

11 GAP

11.1 Short Description

GAP is a system for computational discrete algebra, with particular emphasis on Computational Group Theory. Has been started in 1986 at Lehrstuhl D für Mathematik at Reinisch−Westfällische Technische Hochschule Aachen (RWTH). After 1997, the development was coordinated at the University of St Andrews. Since 2005 there are GAP Centers (at RWTH Aachen, Technische Universität Braunschweig, Colorado State University in Fort Collins, and University of St. Andrews) which coordinate the further development and maintenance of the system.

The system provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the GAP language as well as large data libraries of algebraic objects. It used in research and teaching for studying groups and their representations, rings, vector spaces, algebras, combinatorial structures, and more [GAPWeb].

11.2 Technical Information on the System

11.2.1 Name of the System and Website

GAP − Group, Algorithms and Programming
http://www.gap−system.org/

11.2.2 Project Leader

GAP is an international cooperation of many people, including user contribution. They refer to themselves as the GAP Group. Part of this group are the GAP authors, Module authors and maintainers, Support group and the Council group.

There is a GAP Council consisted of senior mathematicians and computer scientists which provide advice for the future developments of GAP and also functions as an editorial board for GAP packages submitted by users. The current chair of the council is Edmund Robertson (University of St. Andrews). The leader of the consortium that develops GAP is Steve Linton (University of St. Andrews).

For a complete list of people in the GAP group see the GAP web site.

11.2.3 Main Publications

V. Felsch and J. Neubüser. An algorithm for the computation of conjugacy classes and centralizers in p−groups. *Symbolic and Algebraic Computation* (proceedings of EUROSAM ’79, Marseille, 1979), edited by E. W. Ng, LNCS, Vol. 72, Springer, Berlin, 1979, pp. 452−465.

Steve Linton and Thomas Breuer. The GAP 4 Type System: Organising Algebraic Algorithms. In Oliver Gloor, editor, ISSAC98:Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, pages 38−45. ACM Press, 1998.

The GAP Manual, see website : http://www.gap−system.org/~gap/Doc/manuals.html (last checked on 08.10.2007)

11.2.4 Implementation Language

It has a kernel, written in C, which provides the user with

* the GAP language,
* an interactive environment for developing and using GAP programs,
* memory management,
* a set of programming tools for testing, debugging, and timing algorithms, and
* fast versions of time critical operations for various data types.

11.2.5 System Availability and Prerequisites

The system is distributed under the terms of the GNU Public License. The copyright for the core part of the GAP distribution is by the GAP Group and the copyright of redistributed packages remains with their authors. The current version is 4.4.9

There are no specific system prerequisites. Runs on Windows, Linux and MacOS operating systems. Some packages included in the system's distribution require, however, other software (like a LaTeX installation for generating documentation, Evince, Tcl/Tk for visualization of semigroups).

11.3 Algorithm Libraries

The GAP system has functions for elementary number theory, most of them dealing with the group of coprime integers. It can compute with integers and rational numbers, Gaussian numbers, p−adic numbers. It also knows how to handle elements of finite fields, (multivariate) polynomials, rational functions, different kinds of group elements (like permutations, matrices), as well as many algebraic structures.

A detailed list of GAP packages implementing various combinatorial, algebraic, discrete, etc. algorithms can be found at [GAPWeb].

11.3.1 Reasoners

11.3.2 Graphical Tools and Interfaces

XGAP package is a front end for mathematical operations on subgroup lattices.

11.4 User Language

11.4.1 Programming Language

The GAP programming language is interpreted and can be compiled. It is an un−typed imperative programming language with functions as first class objects and some extra built−in data types such as permutations and finite field elements. The language supports a form of object−oriented programming, similar to that supported by languages like C++ and Java [GAP−Web, GAP4Man].

It can be used interactively at the keyboard or to write programs to be saved and then exe−cuted. Such programs can easily be modified and rerun. The language features:

* Pascal−like control structures,
* automatic memory management including garbage collection,
* streams,
* flexible list and record data types,
* built−in data types for key algebraic objects,
* automatic method selection building on a mechanism for automatically choosing the highest ranked method for a certain operation, depending on the current state of all its arguments, so that GAP objects representing mathematical objects may gain knowledge about themselves during their lifetime resulting in better methods being chosen later on.

It is possible to compile the interpreted language into C for better performance.

11.4.2 Logic Language for the Formulation of Mathematical Knowledge

The mathematical knowledge is formulated using the GAP language (un−typed, imperative programming language).

11.4.3 Mathematical Syntax

The syntax used to formulate algorithms and knowledge is the syntax of the GAP program−ming language.

11.5 Mathematical Knowledge Bases

11.5.1 Available Theories and Knowledge Bases

The GAP system has a collection of libraries containing special types of groups. Mainly, these libraries were constructed with the help of algorithms. The libraries contain groups them−selves (as set of generators) and/or the number of groups of certain orders.

Small Groups Library (by E. A. O'Brien, B. Eick, and H. U. Besche). Contains all groups of order up to 2000 (except 1024) and some infinite series of groups characterized by the prime

number decomposition of their orders. (’Small’ refers to group of small order, i.e. below a certain bound). The algorithms used to generate this library were not checked (human errors may occur in them, e.g. typos). The results were cross−checked with existing data, i.e. with group classification done by other people [BescheEickOBrien99].

Basic Groups Library: provides generic functions to construct cyclic groups, abelian groups, elementary abelian groups, dihedral groups, extraspecial groups, alternating groups, symmetric groups, Mathieu groups, Suzuki groups, and Ree groups.

Classical Groups Library: provides generic functions to construct general linear groups, special linear groups, general unitary groups, special unitary groups, symplectic groups, general orthogonal groups, special orthogonal groups, projective general linear groups, projective special linear groups, projective general unitary groups, projective special unitary groups, and projective symplectic groups.

Perfect Groups Library (by D. Holt, W. Plesken, and V. Felsch): provides, up to isomor−phism, a list of all perfect groups whose sizes are less than 10^6(excluding certain sizes). The groups are stored by presentations

Transitive Permutation Groups Library (by A. Hulpke): currently contains representa−tives for all transitive permutation groups of degree at most 30.

Primitive Permutation Groups Library (by C. Roney−Dougal and many other people): includes primitive permutation groups of degree less than 2500 up to permutation isomorphism

Irreducible Solvable Matrix Groups Library (by M. Short): contains the irreducible solvable subgroups of GL(n,q) for qn up to 243.

IRREDSOL Library (by B. Hoefling): provides a library of all irreducible solvable sub−groups of GL(n,q), up to conjugacy, for small values of n and q.

Integral Matrix Groups Library (by W. Plesken, B. Souvignier, G. Nebe, and V. Felsch) contains Q−class representatives of all irreducible maximal finite integral matrix groups of dimension up to 31 and Z−class representatives of those among these groups which are of dimension at most 11 or of dimension 13, 17, 19, or 23.

CrystCat Library (by V. Felsch, F. Gähler) provides a catalog of crystallographic groups of dimensions 2, 3, and 4 which covers most of the data contained in the book Crystallographic groups of four−dimensional space by H. Brown, R. Bülow, J. Neubüser, H. Wondratschek, and H. Zassenhaus (John Wiley, New York, 1978).

AClib Library (by K. Dekimpe, B. Eick) contains a library of almost crystallographic groups and a some algorithms to compute with these groups.

Tables of Marks Library (by G. Pfeiffer and T. Merkwitz): tables of marks for many almost simple groups.

CTblLib Library (by T. Breuer) contains the GAP Character Table Library

Lie Algebras Library (by W. A. de Graaf and T. Breuer) provides generic functions to construct free Lie algebras, full matrix Lie algebras, and simple Lie algebras of types A, B, C, D, E, F, G, H, K, S, and W, as well as the irreducible modules of semisimple Lie algebras in characteristic 0.

11.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The libraries provide selection functions for retrieving groups by their properties (like order). Some libraries have more retrieval functions, like, in the case of SmallGroups, returning the catalogue number of a group G (the order of the group should be covered by the library). Further example: IRREDSOL can identify in the library a group to which a given one is conjugate.

11.5.3 Tools for Inventing Mathematical Knowledge

11.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

11.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

11.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

11.5.7 Standardization, Inter−Operability

One of GAP's packages (If, written by M. Costantini) has translators to/from the following systems:

* CoCoA (COmputations in COmmutative Algebra),
* Macauly 2 (algebraic geometry research),
* Singular (CAS for Polynomial Computations),
* Plural (CAS for Non−commutative Polynomial Computations),
* Kant/Kash (Computational Algebraic Number Theory),
* Pari/GP (fast computation in number theory),
* ARIBAS (an interactive interpreter for big integer arithmetic and multi−precision floating point arithmetic),
* Risa/Asir (general purpose CAS),
* MuPAD (mathematical expert system for doing symbolic and exact algebraic computations),
* Maple (commercial, general purpose CAS),
* Mathematica (commercial, general purpose CAS),
* YACAS (general purpose CAS for symbolic manipulation of mathematical expressions)

It also has a package providing an OpenMath phrasebook for GAP (written by M. Constan−tini andA. Solomon) , meant to facilitate communication with other systems that understand OpenMath, and a package (written by R.A. Wilson, R.A. Parker, S.Nickerson, J.N. Bray, T.Breuer) that provides an interface to the Atlas of Group Representations − a database that comprises representations of many almost simple groups and information about their maximal subgroups [AtlasWeb].

11.5.8 Web Access

11.6 Example of a Theory Exploration Session

The examples are taken from the website of Centre for Interdisciplinary Research in Compu−tational Algebra (Circa), University of St. Andrews (http://www−circa.mcs.st−and.ac.uk/gap.html , last checked on 08.10.2007). The (short) listing below gives an idea about using GAP and its language. The system's outputs are marked with '>>'.

11.6.1 Arithmetic

```
(2+3)*(7−5);
>> 10
2*3/66;
>> 1/11
2^100−1;
>> 1267650600228229401496703205375
Sqrt(99) in Integers;
>> false
```

11.6.2 Lists

Define a list by range:

```
r_1:=[1..10];
>> [1.. 10]

5 in r_1;
>> true

Length(r_1);
>> 10
```

GAP can use ranges to produce an arithmetic progression. For example:

```
r_2:=[1,3..17];
>> [1,3.. 17]
```

Display the elements of the list:

```
Elements(r_2);
>> [1,3,5,7,9,11,13,15,17]
```

GAP can also define lists by enumerating all its elements. It can perform a form of arithmetic with lists.

```
2*[1..9]-1;
>> [1,3,5,7,9,11,13,15,17]
```

Add or subtract lists -- even of different lengths

```
[1,2,3]+[4,5];
>> [5,7,3]
```

11.6.3 Sets

GAP sometimes considers lists as sets, but needs to do some converting before applying the set operations. Namely, it will only count list of integers as sets if they are ordered increasingly and have no repetitions.

```
t:=[1,2,2,3,4,5,6];
>> [1,2,2,3,4,5,6]

IsSet(t);
>> false
```

Make the list *t* a set:

```
tt:=Set(t);
>> [1,2,3,4,5,6]
```

Take another list:

```
s:=[1,8,6,2];; ss=Set(s);
>> [1,2,6,8]

Intersection(s,t);
>> [1,2,6]

IsEqualSet(Intersection(s,t),Intersection(ss,tt));
>> true

Union(s,t);
>> [1,2,3,4,5,6,8]
```

11.6.4 Loops

Two examples, the second one using a conditional

```
for i in[1..10] do
    Print(i," ");
od;
>> 1 2 3 4 5 6 7 8 9 10

for i in[1,3.. 19] do
    if IsPrimeInt(i) then Print(i," is prime ","\n");fi;
od;
>> 3 is prime
>> 5 is prime
>> 7 is prime
>> 11 is prime
>> 13 is prime
>> 17 is prime
>> 19 is prime
```

11.6.5 Functions

The following code defines a function that simply squares the argument it is given.

```
squared:=function(n);
    return n^2;
end;
>> function(n)... end
```

If you want to see the function written out fully, you can ask GAP to Print it.

```
Print(squared);
>> function (n)
>>  ;
>>  return n^2;
>> end
```

Call the function we just defined:

```
squared(12345);
>> 152399025
```

Another way of defining functions:

```
cubed:=n→n^3;
>> function(n)... end

Print(cubed);
>> function (n)
```

```
>>  return n^3;
>> end

cubed(123456);
>> 1881640295202816
```

We can produce lists of cubes as follows:

```
List([1..10],cubed);
>> [1,8,27,64,125,216,343,512,729,1000]
```

11.6.6 Solving Mathematical Problems

Problem:

Show that the dihedral group of order 24 has a subgroup of order *k* for every *k* dividing 24.

Let *G* be a group of order *n* which does not have a subgroup of order *k* for every *k* dividing *n*. Find all such groups of order less than 50.

[Hint:ConjugacyClassesSubgroups(G) will give a list of conjugacy classes of subgroups of G and Representative will allow the selection of one subgroup from each conjugacy class.]

Solution:

(The text between the input and output is not generated by GAP, but it's humanly written explanations.)

```
G:=DihedralGroup(24);
>> <pc group of size 24 with 4 generators>

lis:=ConjugacyClassesSubgroups(G);
l:=List(lis,Representative);
>> [Group( <identity>of...)^G,Group( [ f1*f2 ] )^G,Group( [
f3*f4 ] )^G,Group( [ f1 ] )^G,Group( [ f4 ] )^G,Group( [
f1,f3*f4 ] )^G,Group( [ f1*f2,f3*f4 ] )^G,Group( [
f2*f4^2,f3*f4 ] )^G,Group( [ f4,f3 ] )^G,Group( [ f4,f1*f2 ]
)^G,Group( [ f4,f1 ] )^G,Group( [ f1,f2*f4^2,f3*f4 ] )^G,Group(
[ f4,f3,f1 ] )^G,Group( [ f4,f3,f1*f2 ] )^G,Group( [ f4,f3,f2 ]
)^G,Group( [ f4,f3,f1,f2 ] )^G]

List(l,Size);
>> [1,2,2,2,3,4,4,4,6,6,6,8,12,12,12,24]

DivisorsInt(24);
>> [1,2,3,4,6,8,12,24]
```

Hence there is a subgroup of that order for each divisor.

For the next part we first loop over group orders up to 50 and then loop over the number of groups of that order.

nfac is the number of distinct divisors of the group order
lis is a list of representatives, one for each conjugacy class of subgroups
nsub is the number of distinct orders of subgroups

```
for gporder in[2..50] do
    for i in[1..NumberSmallGroups(gporder)] do
        G:=SmallGroup(gporder,i);
        nfac:=Length(DivisorsInt(gporder));
        lis:=List(ConjugacyClassesSubgroups(G),Representative);
        nsub:=Length(Set(List(lis,Size)));
        if nfac<>nsub then Print("Order ",gporder," group
number ",i,"\n");fi;
    od;
od;
>> Order 12 group number 3
>> Order 24 group number 3
>> Order 36 group number 3
>> Order 36 group number 9
>> Order 36 group number 11
>> Order 48 group number 3
>> Order 48 group number 50
```

Problem: Finding groups with certain properties.

There are exactly two groups of order p^2, p a prime. Both are abelian, one is cyclic.

Find the smallest value of n such that n is not the square of a prime, yet there are exactly two groups of order n with both groups abelian and one cyclic.

Solution:

Some easy observations:

1. There is always one cyclic group of order n for every n. It is abelian so we need only need to check that for a given n there are exactly 2 groups and that "IsAbelian" (a GAP function, a.n.) has the same value for each.

2. We need to jump over the squares of primes. These are precisely the numbers with 2 equal factors.

3. We could jump over other vales of n which we know will not work, such as primes. However we don't bother to write this into our code.

```
found:=0;;
n:=2;;
while found<5 do
    n:=n+1;
    if Length(Factors(n))=2 and Factors(n)[1]=Factors(n)[2] then
        n:=n+1;
    fi;
    glist:=AllSmallGroups(Size,n);
    ablist:=List(glist,IsAbelian);
    if Length(ablist)=2 and ablist[1]=ablist[2] then
        found:=found+1;
        Print(n,"\n");
    fi;
od;
>> 45
```

```
>> 99
>> 153
>> 175
>> 207
```

12 GiNaC

12.1 Short Description

GiNaC is specifically built to be a fast symbolic engine for complex computations in quan−tum field theory [BauerEtAl02]. The available operations are to be performed within the C++ language. GiNaC's main users are physicists

12.2 Technical Information on the System

12.2.1 Name of the System and Website

GiNaC − recursive abbreviation for GiNaC is Not a CAS.
http://www.ginac.de/

12.2.2 Project Leaders and Group

The project has no named leader. Active coders are: Chris Dams, Vladimir V. Kisil, Richard Kreckel (Institute of Physics, Johannes−Gutenberg−University, Mainz, Germany), Alexei Sheplyakov, Jens Vollinga.

12.2.3 Main Publications

C. Bauer, A. Frink, and R. Kreckel. Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language. *J. Symbolic Computation* (2002) 33, 1−12.

J. Vollinga. "GiNaC − Symbolic computation with C++". *Nucl.Instrum.Meth.* A559 (2006) 282−284. Also available at arXiv.org as arXiv:hep−ph/0510057 v1. 5 Oct. 2005.

12.2.4 Implementation Language

C++

12.2.5 System Availability and Prerequisites

Distributed under the GPL licence. There are binary packages available for a variety of Linux operating systems. As a prerequisite, it needs the CLN library (see the chapter on CLN of this document). If the sources are downloaded, a C++ compiler should be available on the system.

12.3 Algorithm Libraries

12.3.1 Numerical Library

Uses the CLN library, which is a library for efficient computations with all kinds of numbers in arbitrary precision. See http://www.ginac.de/CLN/ for more details.

12.3.2 Algebraic Library

GiNaC has classes that describe and manipulate algebraic symbols, truncated power series, symbolic functions, matrices and vectors of expressions, non−commutative objects, polynomial manipulations, etc.

12.3.3 Reasoners

12.3.4 Graphical Tools and Interfaces

12.4 User Language

12.5 Mathematical Knowledge Bases

12.6 Example of a Theory Exploration Session

We present a small C++ program that uses the GiNaC library [BauerEtAl02]. It implements the Hermite polynomial:

```
  1  #include <ginac/ginac.h>
  2  using namespace GiNaC;
  3
  4  ex HermitePoly(const symbol & x, int n)
  5  {
  6       const ex HGen = exp(−pow(x,2));
  7       // uses the identity H_n(x) == (−1)^n exp(x^2)
(d/dx)^n exp(−x^2)
  8       return normal(pow(−1,n) * HGen.diff(x, n) / HGen);
  9  }
 10
```

```
11  int main(int argc, char **argv)
12  {
13       int degree = atoi(argv[1]);
14       numeric value = numeric(argv[2]);
15       symbol z("z");
16       ex H = HermitePoly(z,degree);
17       cout << "H_" << degree << "(z) == "
18            << H << endl;
19       cout << "H_" << degree << "(" << value << ") == "
20            << H.subs(z==value) << endl;
21       return 0;
22  }
```

After compilation, we call it with 11 and 0.8 as command line arguments. As a result it will print the 11th Hermite polynomial together with its numerically value at z = 0.8:

```
 1  H_11(z) ==
-665280*z+2217600*z^3-1774080*z^5+506880*z^7-56320*z^9+2048*z^11
 2  H_11(0.8) == 120773.8855954841959
```

13 HELM

13.1 Short Description

The aim of the HELM project is "to study and develop a technological infrastructure for the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge'' [AspertiEtAl00]. In a nutshell, it has means to import formal libraries, to transform their formal content into an intermediate, MathML content representation format, and to make it available on the internet for browsing and searching. The existing libraries (which we call 'the Helm library') contain data imported from the Coq and the NuPRL systems. Important tools used by Helm (like search within the library, proof checking) have been developed in the frame of the European IST project MoWGLI [MoWGLI].

13.2 Technical Information on the System

13.2.1 Name of the System and Website

Hypertextual Electronic Library of Mathematics (HELM), http://helm.cs.unibo.it/

13.2.2 Project Leaders and Group

Leader: Andrea Asperti. Project members: Irene Schena, Luca Padovani, Ferruccio Guidi, Claudio Sacerdoti Coen, Stefano Zacchiroli and Enrico Tassi.

13.2.3 Main Publications

A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. Towards a Library of Formal Mathematics. Technical Report of TPHOLS'2000 Conference, Portland, Oregon, USA, August 2000.

Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, Stefano Zacchiroli. User Interaction with the Matita Proof Assistant,
Journal of Automated Reasoning, Special Issue on User Interfaces for Theorem Proving, Volume 39 , Issue 2 , August 2007, pages: 109 − 139.

Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli, A Content Based Mathematical Search Engine: Whelp, Proceedings of TYPES 2004, Ed. C. Paulin−Mohring, and B. Werner, LNCS 3839, pages 17—32, Springer Verlag .

13.2.4 Implementation Language

HELM is implemented in PERL, OCaml and Java.

13.2.5 System Availability and Prerequisites

The software developed by HELM is free and can be redistributed and/or modified under the terms of the GNU General Public License.

To interact with the Whelp search engine a browser is needed.

The Matita prover [Matita] requires several tools and libraries: Caml (Objective Caml compiler, version 3.09 or above, Findlib version 1.1.1 or above, OCamlExpat, GMetaDom, OCamlHTTP, LablGTK version 2.6.0 or above) and MySQL (+OCaml MySQL) are needed. Also a GTK+ widget for rendering MathML documents [MathML] (GTKMathView, LablGtk−MAthView) and an extension for rendering texts with features of source code editors (Gtk−SourceView , LablGtkSourceView) are needed.

The Ocamlnet collection of libraries for application-level Internet protocols is needed. Also the ulex (Unicode lexer generator for OCaml) and CamlZip (OCaml library to access .gz files is needed). For installation details please see the project's webpage.

13.3 Algorithm Libraries

13.3.1 Numerical, Discrete, Algebraic, etc., Libraries

13.3.2 Reasoners

Matita [Matita, AspertiEtAl07, AspertiEtAl07a] is a standalone prover, whose development has been intertwined with the development of the Helm project. It is an interactive, tactic−based prover. For proving a certain goal, the system automatically retrieves the appropriate assumptions from the existing knowledge base. The retrieval is based on the signature and context of the goal. The basic automation tactic in Matita is the *auto* tactic, which boils down to an iterated use of *apply* (modus ponens). Other tactics which the prover uses are *intro* (∀-introduction), *elim* (induction), *cut* (forward reasoning), *simplify*.

A recent extension of the prover is the addition of the paramodulation tactic: taking an equational goal and using a given−clause algorithm [NieuwenhuisRubio01], Matita returns all known equational facts from the library (and the local context). For more details see [Asperti−Tassi07].

13.3.3 Graphical Tools and Interfaces

Whelp [AspertiEtAl04] is a web−based application, allowing browsing and searching of mathematical formulae. The initial development was done in the frame of the Mowgli project [MOWGLI]. The user inputs in a textbox her search query, and selects the search option she prefers (locate, match, elim or hint) and gets back a list of pages (each of them containing MathML elements) that fit the query. For an example of how the search works see the **Tools for Retrieval** subsubsection in the **Mathematical Knowledge Bases** subsection below.

Matita [AspertiEtAl07] is an interactive prover meant as an interface between the user and the mathematical knowledge library of HELM. It also provides authoring facilities and a user interface for browsing, adding mathematical knowledge, indexing and searching mathematical objects in the HELM library.

Matita's authoring interface functions in a similar fashion to the ProofGeneral [Aspinall00] generic interface. It contains a window for script editing, one for open goals that need to be proved and another one for messages. The rendering of mathematical expressions is done in a user−friendly manner with a MathML-compliant GTK+ widget, also developed by the HELM team.

The mathematical knowledge can be browsed by using hyperlinks. One can access directly the mathematical objects (i.e. definitions, theorems etc.) or browse through the theories (which are structured collections of explanatory text and mathematical objects, assembled by an author for presentational purposes). The mathematical information can be viewed as raw semantic encoding, as MathML formula or presented as HTML (the HTML is generated on the fly from the MathML encoding).

The user can add its own document (written in LaTeX or in CoQ) to the content of the library. These documents will be then translated to MathML, with the help of HERMES [Anghelache04] and a script converting CoQ mathematical formulae to MathML. The indexing and searching mechanism is done in the same way as in Whelp.

ProofWeb [Kaliszyk06] is a web interface for Matita (as well as Coq and Isabelle), with a similar functionality as ProofGeneral.

13.4 User Language

13.4.1 Programming Language

The user can formulate new proof scripts using the procedural proof language of Matita, which is based on the approach used by the LCF [GordonEtAl79] theorem prover. The tactics (statements of this language) "are collected in textual scripts that are interpreted one statement at the time" [AspertiEtAl07]. New tactics are formed from existing tactics and tacticals.

13.4.2 Logic Language for the Formulation of Mathematical Knowledge

The underlying logic language presupposed by Helm is the Calculus of Inductive Construc−tions (CIC), the same logic underlying CoQ. In Matita, the proof terms are represented as λ-terms of CIC.

13.4.3 Mathematical Syntax

Helm (and also Matita) uses an extension of content MathML for encoding both mathematical expressions and proofs [GoguadzeEtAl].

13.5 Mathematical Knowledge Bases

13.5.1 Available Theories

HELM imports libraries from Coq and C−CoRN [AspertiEtAl06], and, experimentally, from NuPRL. It transforms the representation into an intermediate MathML content representation [MathML], and makes it available on the internet for browsing and searching. Among the libraries available now online are: Constructive Algebra (from C−CoRN [C−CoRNWeb], theories necessary for proving the Fundamental Theorem of Algebra), Gröbner Basis, Zermelo-Frankel Set Theory, Arithmetic, Booleans, Reals, Exact Real Arithmetic, Automata Theory, Geometry, etc. See also the sections on Coq and C−CoRN of this document.

13.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Helm's search engine is called Whelp. It has its own syntax, which, in most of the queries, is disambiguated into suitable terms of Coq's Calculus of Inductive Constructions.

Whelp is based on the metadata model described in [GuidiCoen03, Coen04, Asperti−Selmi04]. Metadata is used for indexing mathematical notions and marking input/output types as well as the components in the body (hypotheses, conclusion, consistency proof of the notion). For the metadata, only the constant symbols of the original mathematical formula are taken into account. For example, from

$$\forall_{n:\mathrm{nat}} \; \forall_{m:\mathrm{nat}} \; \forall_{p:\mathrm{nat}} \; n * (m + p) = n * m + n * p$$

the automatically extracted metadata contains only the constants nat, =, $*$, + and their position (hypothesis/conclusion) in the formula.

Search is then not performed on the stored knowledge, but only on the metadata. Thus one may view the metadata as taking the role of indexes in a database, allowing quick searches. The search facilities described in [AspertiEtAl04] are as follows:

• *Locate* is the simplest search. The user introduces a string regular expression, and the tool outputs the list of all formulae that have this string as ‘‘name".

• *Hint* retrieves all theorems that can be applied to derive the current goal. This means, given a theorem $t \Rightarrow t_2 \Rightarrow \ldots t_n \Rightarrow t$ and a goal g, it checks whether there exists a substitution θ such that $t\,\theta = g$. This is simulated in HELM by metadata filtering.

• *Match* amounts to reverting the operation of indexing, looking for terms matching the metadata set, which are automatically computed from the user input.

• ***Elim*** gives the induction axioms for a domain. It is used for inductive proofs of properties of functions and relations over algebraic types. For example, giving *nat* as input, will return the usual induction principle on natural numbers as a result.

The Matita prover uses Whelp for automatically retrieving, from the whole library, a subset of theorems worth considering as assumptions in the process of proving the current goal.

13.5.3 Tools for Inventing Mathematical Knowledge

13.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The Matita prover has proof checking capabilities (implemented by the system kernel which is a CIC type−checker).

13.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

Matita ensures the consistence of the mathematical library during the formalization by using the mechanisms of invalidation and regeneration [AspertiEtAl07]. A mathematical concept and all concepts depending on it are invalidated when the concept is changed or removed, and need to be regenerated to verify if they are still valid.

13.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

13.5.7 Standardization, Inter−Operability

HELM is based on MathML. As such, mathematical knowledge encoded by HELM can be easily presented on the web, using MathML plug−ins.

The proof terms of HELM and CoQ are compatible: mathematical formulae/terms can be exported from CoQ to become part of the library of HELM (and thus Matita).

13.5.8 Web Access

The knowledge base of HELM can be browsed and searched online, on the project's web page.

13.6 Example of a Theory Exploration Session

The user can browse or search online in the knowledge base of HELM at http://helm.cs.unibo.it/. Browsing is done by listing the directories containing the available mathematical theories (there is e.g. a directory for CoQ, another for NuPRL, yet another for C−CoRN). Each mathematical theory is in its turn a directory, and each mathematical formula is presented, using MathML, in a user-friendly, two-dimensional syntax. Searching is done by calling the tool Whelp, described above in this section.

We give here an example of a Matita session [Asperti07].

One can define the naturals:

```
inductive nat:Set \def
    |O:nat
    |S:nat \to nat.
```

and also the sum of natural numbers (plus(n, m) = $\begin{cases} m & \leftarrow \quad n = 0 \\ S(p, m) & \leftarrow \quad n = S(p) \end{cases}$) as:

```
let rec plus n m \def
    match n with
    [ O \Rightarrow m
    | (S p) \Rightarrow S (plus p m)].
```

The theorem $n = 0 + n$ can be proved by applying the tactics *intros* (assuming the hypothesis of the statement, in this case n : nat), *simplify* (n + 0 is equal to n), *reflexivity* (n = n), and *qed* (that rechecks the proof).

```
theorem plus_O_n: \forall n:nat. n=O+n.
    intros.simplify.reflexivity.
qed.
```

For proving that the successor of $n + m$ is n plus the successor of m one uses induction (the *elim* tactic).

```
theorem plus_n_Sm : \forall n,m:nat. S (n+m) = n+(S m).
    intros.elim n.
        simplify.reflexivity.
        simplify.rewrite < H.reflexivity.
qed.
```

14 Hol 4

14.1 Short Description

HOL is an automated proof system for higher order logic. It is an environment in which theorems can be proved and, at the same time, proof tools can be implemented [HolWeb]. Over the years several versions of HOL were implemented, like HOL88 from Cambridge, HOL90 from Calgary and Bell Labs, HOL98 from Cambridge, Glasgow and Utah. HOL 4 is a successor of these, mainly based on HOL98 and incorporating ideas from HOL Light (which has a simpler logical core and has little legacy code, see [HolLightWeb]). There is a HOL2000 initiative which tries to put together the design of the next generation of HOL prover.

14.2 Technical Information on the System

14.2.1 Name of the System and Website

Hol 4, Kananaskis 4. http://hol.sourceforge.net/

14.2.2 Project Leaders and Group

There are two project managers: Michael Norrish, Konrad Slind.

The group of developers contain over 20 persons, for a complete and current list of developers see http://sourceforge.net/project/memberlist.php?group_id=31790 .

14.2.3 Main Publications

M. J. C. Gordon and T. F. Melham, editors. Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press, 1993. (Although a description for Hol88, the book is describing design principles used in the current HOL system.)

14.2.4 Implementation Language

Standard ML (The Moscow ML implementation).

14.2.5 System Availability and Prerequisites

Hol 4 is open source project with a BSD−style licence that allows its free use in commercial products. It is available for the following operating systems: 32−bit MS Windows (NT/2000/XP), All POSIX (Linux/BSD/UNIX−like OSes), Win2K, WinXP. For compiling the sources one needs a local installation of Moscow ML.

14.3 Algorithm Libraries

14.3.1 Numerical, Discrete, Algebraic, etc. Libraries

14.3.2 Reasoners

The deductive system of the HOL logic is specified by eight rules of inference: Assumption introduction, Reflexivity, Beta conversion, Substitution, Abstraction, Type instantiation, Discharging an assumption, Modus ponens. In addition to these rules, there are five axioms which axiomatize the HOL logic [HolLogic].

HOL 4 has reasoners for first order logic (mesonLib − model−elimination method − and metisLib − resolution method), a propositional tautology prover, an interface to external binary decision diagram engines, a symbolic checker, and a library that supports call−by−value evaluation of HOL functions by deductive steps. It also has a rewriting engine − simpLib − which is especially recommended during interactive theorem proving. The arithmetic libraries provided contain a suite of definitions, theorems, and automated inference support.

14.3.3 Graphical Tools and Interfaces

Interacting with the system is done mainly via Emacs. As the system grew more and more popular, the interest for graphical interfaces to HOL was high. There are graphical tools implemented by users of the system, we name here just a small subset, pointing the reader to look for more, if interested. For example xhol [SchubertBiggs94] implements a X Windows based front end to HOL with a graphical display of the active proof tree. Several users have implemented window inferencing in HOL. For example, [LangbackaEtAl95] describes such an interface which supports window inference: i.e. point and click to do proof refinement. Another similar tool was implemented by J. Grundy and is described in [Grundy91].

14.4 User Language

14.4.1 Programming Language

The ML language. HOL Logic constructs can be used.

14.4.2 Logic Language for the Formulation of Mathematical Knowledge

The logic of HOL is a variant of Church’s simple theory of types.

14.4.3 Mathematical Syntax

The syntax of HOL Logic [HolLogic, HolTut].

14.5 Mathematical Knowledge Bases

14.5.1 Available Theories and Knowledge Bases

The core theories of HOL 4 contain notions about booleans, pairs, disjoint sums, numbers (numerals, integers, rationals, reals, probability theory, n−bit words), sequences, collections (sets, multisets, relations, finite maps), partial orders, a theory of polynomials over the reals, temporal logic, computational tree logic and μ−calculus.

Along the time, users of the system have formalized big chunks of mathematics for the purpose of proving some specific, interesting theorems. All these formalizations are not central−ized anywhere. We give here just a few examples of such formalizations.

Real numbers have been formalized in the frame of J. Harrison's PhD thesis [Harrison96].

The Flyspeck project, led by T. Hales, using HOL light, aims at formalizing the corpus of mathematical knowledge to prove The Kepler Conjecture.

For other formalizations see, for example, [Fox05], [Agerholm94], [BlackWindley98], [BowenGordon95] the proceedings of the TPHOL conference series.

14.5.2 Tools for Retrieval in Mathematical Knowledge Bases

14.5.3 Tools for Inventing Mathematical Knowledge

14.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

14.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

14.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

Reflection in HOL is not considered explicitly, though there are means to program inference rules (i.e. tacticals).

14.5.7 Standardization, Inter−Operability

There are various translators between Hol and other systems. See, for example, [GordonEt−Al06], which describes an interface to ACL2, [Harrison96a] which describes a Mizar mode for HOL, or [NaumovEtAl01] which describes a proof translator to/from NuPRL. In [Harrison−Thery93] a general mechanism of linking HOL to a CAS is described. Implementations follow−ing this description were mainly done to connect HOL to Maple.

14.5.8 Web Access

14.6 Example of a Theory Exploration Session

Working with HOL4 is done in an read−eval−print loop. Most of the users work using an Emacs mode for HOL4. The examples below are taken from the tutorial available on HOL4 website. The ML prompt is '−', here the user must type her input. The answers from the system are prompted by '>'.

Here is the welcome window when starting Hol, and a first command where we add 1 to the list [2,3,4,5].

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−
       HOL−4 [Kananaskis 4 (built Fri Apr 12 15:34:35 2002)]
       For introductory HOL help, type: help "hol";
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−
[loading theories and proof tools ************* ]
[closing file
"/local/scratch/mn200/Work/hol98/tools/end−init−boss.sml"]
− 1 :: [2,3,4,5];
> val it = [1, 2, 3, 4, 5] : int list
```

'it' is a special variable that stores the last system output.

```
− val l = it;
> val l = [1, 2, 3, 4, 5] : int list
− tl l;
> val it = [2, 3, 4, 5] : int list
− hd it;
> val it = 2 : int
− tl(tl(tl(tl(tl l))));
> val it = [] : int list
```

Below we define a function, zip, which converts a pair of lists ([a,b,c,...], [x,y,z, ...]) to a list of pairs [(a,x), (b, y), (c, z), ...]

```
− fun zip(l1,l2) =
    if null l1 orelse null l2 then []
    else (hd l1,hd l2) :: zip(tl l1,tl l2);
> val zip = fn : 'a list * 'b list −> ('a * 'b) list
− zip([1,2,3],["a","b","c"]);
> val it = [(1, "a"), (2, "b"), (3, "c")] : (int * string) list
```

We want prove $\forall x$, x divides 0. For this we open the arithmetic theory and define an arith−metic rewrite tactic, specialized from the rewriter provided by bossLib.

```
− open arithmeticTheory;
  ...
```

```
- val ARW_TAC = RW_TAC arith_ss;
> val ARW_TAC =
    fn
    : thm list -> term list * term ->
       (term list * term) list * (thm list -> thm)
```

We define now the predicate ’divides’:

```
- val divides = Define ’divides a b = ?x. b = a * x’;

Definition has been stored under "divides_def".
> val divides = |- !a b. divides a b = ?x. b = a * x : thm
```

and with the following command we tell the system to treat ’divides’ as right associative.

```
- set_fixity "divides" (Infixr 450);
```

We start now the proof of ∀*x*, *x* divides *0*. The ’g’ in the given command tells the HOL proof manager to start a new proof. The exclamantion sign stands for ∀, the question sign (further below in the proof) stands for ∃.

```
- g ’!x. x divides 0’;

> val it =
    Proof manager status: 1 proof.
    1. Incomplete:
         Initial goal:
         !x. x divides 0
     : proofs
```

We expand the definition of ’divides’:

```
- e (ARW_TAC [divides]);

OK..
1 subgoal:
> val it =
     ?x’. (x = 0) \/ (x’ = 0)
```

We instantiate the existential quantifier by hand:

```
- e (EXISTS_TAC ’’0’’);

OK..
1 subgoal:
> val it =
     (x = 0) \/ (0 = 0)
```

Last, simplify the goal, which finishes the proof.

```
- e (ARW_TAC []);
```

```
OK..
Goal proved.
|- (x = 0) \/ (0 = 0)
Goal proved.
|- ?x'. (x = 0) \/ (x' = 0)
> val it =
     Initial goal proved.
     |- !x. x divides 0
```

15 HR

15.1 Short Description

HR invents concepts and theorems in mathematical domains like finite algebras / groups / rings, graph theory and number theory.The underlying idea is the formation of new conjectures, which are then passed to an external resolution theorem prover (usually Otter [McCune94]): If the proof goes through, the conjecture is adopted as a theorem; otherwise, a model finder (often MACE [McCune03]) is called for drawing counterexamples from it.

Each concept is an operation or a relation on the carrier set of a certain model of interest (e.g.a fixed finite group), represented by a combination of a "data table'' (also called Cayley table in the case of groups) and a formula with an indication of its free variables (the "arguments" of the operation or relation). For example, the concept of multiplication in $\mathbb{Z}_6$ is described by a table containing rows like [4, 2, 2] and [3, 3, 5] together with the formula $[n, a, b] : a \mid n \wedge b \mid n \wedge a * b = n$.

15.2 Technical Information on the System

15.2.1 Name of the System and Website

HR, http://wwwhomes.doc.ic.ac.uk/~sgc/hr/ (the earlier versions at http://www.dai.ed.ac.uk/homes/simonco/research/hr/).

15.2.2 Project Leaders and Group

HR was developed by Simon Colton.

15.2.3 Main Publications

Simon Colton, The HR Program for Theorem Generation Automated Deduction, In CADE−18 : 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27−30, 2002. Proceedings, 2392, pages 285−290. http://www.doc.ic.ac.uk/~sgc/papers/colton_cade02.pdf

Simon Colton, Alan Bundy and Toby Walsh, T. On the Notion of Interestingness in Auto−mated Mathematical Discovery, In International Journal of Human−Computer Studies, Aca−demic Press, 2000. http://www.doc.ic.ac.uk/~sgc/papers/colton_ijhcs00.pdf

Simon Colton, Automated Theory Formation in Pure Mathematics, PhD. Thesis, Department of Artificial Intelligence, University of Edinburgh, 2001. Distinguished Dissertations Series, Springer Verlag, 2002. ISBN 1852336099.

15.2.4 Implementation Language

HR is implemented in Java. The previous implementation of HR was in prolog (sicstus objects).

15.2.5 System Availability and Prerequisites

The following programs are needed: Otter, MACE, LaTeX, Dot and xgraph.

15.3 Algorithm Libraries

15.3.1 Numerical, Discrete, Algebraic, etc. Libraries

15.3.2 Reasoners

15.3.3 Graphical Tools and Interfaces

Since HR was not available for testing, we are not aware how its graphical user interface looks like/works.

15.4 User Language

15.4.1 Programming Language

15.4.2 Logic Language for the Formulation of Mathematical Knowledge

HR deals with first−order mathematical formulae.

15.4.3 Mathematical Syntax

HR has its own custom-tailored syntax.

15.5 Mathematical Knowledge Bases

15.5.1 Available Theories and Knowledge Bases

In **HR** the mathematical domains of finite algebras/groups/rings, graph theory and number theory were explored. It is not clear if these theories are further available, for a user to extend or inspect.

15.5.2 Tools for Retrieval in Mathematical Knowledge Bases

15.5.3 Tools for Inventing Mathematical Knowledge

Concept formation works as follows:

- Initial concepts are given axiomatically (like the multiplication in a group) or by an explicit definition (like divisibility in number theory).The corresponding data tables are either supplied by the user as key examples or generated by MACE[MACE].
- Production rules [BundyEtAl98] are applied to the existing concepts.For example,universal quantification (with respect to the first argument,say) produces an $n-1-$ary predicate from an $n-$ary one.
- Measures of interestingness are used for ranking the concepts on a heuristic basis [ColtonEtAl00], for instance "parsimony'' by measuring the relative size of its data table. Note that these measures are recomputed at various stages in the concept/theory formation cycle.
- Filtering the eventually accepted concepts works by computing a total score computed from various measures of interestingness;these are subsequently used for creating conjectures and new concepts.

Examples of concepts invented in group theory include: Abelian group, cyclic group, order of elements, central elements.

The next step is ***theory formation***. HR takes the newly obtained concept and forms conjec−tures about it:

- If the set of models is empty, a *non−existence conjecture* is made (i.e.the concept definition is inconsistent with the axioms). For instance, the concept of non−trivial idempotent elements in groups does not have a model, thus the conjecture $\nexists_a (a \neq \mathrm{id} \wedge a * a = a)$.
- If the sets of models coincide for the newly created concept and an already existing one,an equivalence conjecture is made.In the case of idempotent elements, HR notices that for all group examples it has, the idempotent elements are the identity, thus generat−ing the conjecture $a * a = a \Leftrightarrow a = \mathrm{id}$.

- If neither of the above two cases applies, HR adds the concept to the theory and looks for subsumption conjectures.These are asserted about concepts whose set of models is contained in the set of models of the new concept. For each such case, HR conjectures that the definition of the new concept is implied by the definition of an old one.

The theories produced in this fashion will include concepts, theorems that relate concepts and proofs of these theorems. Improvements of HR allow using Otter as a filter for newly obtained theorems [ColtonMaple02], inputting initial concepts in Maple format, and substituting Otter and MACE by humans.

15.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

15.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

15.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

15.5.7 Standardization, Inter−Operability

HR is currently connected to the MathWeb software bus [ZimmerEtAl02], being thus available as a mathematical service to the other system connected to the software bus (like Blicksem, Otter, Mace, Maple, CoCoA).

15.5.8 Web Access

15.6 Example of a Theory Exploration Session

Since HR was not available for testing, we could not use it in order to generate an exemplify−ing theory exploration session.

16 ILTP

Note: It is recommended the reader first looks at the chapter on TPTP of this document.

16.1 Short Description

The Intuitionistic Logic Theorem Proving (ILTP) library provides a platform for testing and benchmarking automated theorem proving (ATP) systems for first−order and propositional intuitionistic logic [RathsEtAl07].

16.2 Technical Information on the System

16.2.1 Name of the System and Website

Intuitionistic Logic Theorem Proving
http://www.iltp.de/

16.2.2 Project Leaders and Group

Thomas Raths, Jens Otten. Both are from the University of Potsdam, Department of Com−puter Science, Germany.

16.2.3 Main Publications

Thomas Raths, Jens Otten, Christoph Kreitz. The ILTP Problem Library for Intuitionistic Logic. *Journal of Automated Reasoning*, 38:261−271, 2007.

16.2.4 Implementation Language

The library is a collection of text files, the syntax of the stored knowledge is the one of Prolog.

16.2.5 System Availability and Prerequisites

To be installed and used, the library needs some version of Prolog available on the operating system.

16.3 Algorithm Libraries

16.4 User Language

The authors of the library also maintain it. Users who want to add benchmark problems are advised to send them to the ILTP authors.

16.4.1 Programming Language

Prolog

16.4.2 Logic Language for the Formulation of Mathematical Knowledge

First order logic (with formulae written in first order form or conjunctive normal form).

16.4.3 Mathematical Syntax

First order logic (with formulae written in first order form or conjunctive normal form).

16.5 Mathematical Knowledge Bases

16.5.1 Available Theories and Knowledge Bases

A major source of the problems stored in ILTP come from the TPTP library (see the corre−sponding chapter in this document).

The problems cover the following domains: agents, general algebra, computing theory, commonsense reasoning, geometry, graph theory, group theory, homological algebra, knowl−edge representation, logic calculi, management, miscellaneous, natural language processing, number theory, planning, puzzles, set theory, software creation, software verification, syntactic, topology, constructive geometry, non−clausal group theory, and intuitionistic syntactic.

16.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Uses the tptp2X utility of TPTP (see the respective chapter).

16.5.3 Tools for Inventing Mathematical Knowledge

Uses the tptp2X utility of TPTP (see the respective chapter).

16.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

16.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

Uses the tptp2X utility of TPTP (see the respective chapter).

16.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

16.5.7 Standardization, Inter−Operability

Uses the tptp2X utility of TPTP (see the respective chapter).

16.5.8 Web Access

16.6 Example of a Theory Exploration Session

To get a feeling of how ILTP problems look like see the chapter on TPTP.

17 Infty

17.1 Short Description

The Infty project is a collaboration of several universities and research institutes in Japan which investigates and develops new systems for automatic processing of scientific information. They focus on digitization of scientific documents, user interfaces to scientific information, and accessibility (especially for the visual impaired) to scientific information. The underlined capability of the programs developed in the frame of this project is the treatment of formal text [KanahoriSuzuki06].

17.2 Technical Information on the System

17.2.1 Name of the System and Website

Infty Project. Research Project on Mathematical Information Processing Mathematical Document Recognition and Analysis, User Interface,Accessibility of Scientific Documents.

http://www.inftyproject.org/en/index.html

17.2.2 Project Leaders and Group

Project leader: Masakazu Suzuki (Faculty of Mathematics and Graduate School of Mathematics, Kyushu University, Japan)

Project members: Mitsushi Fujimoto (Fukuoka University of Education), Mamoru Fujiyoshi (The National Center for University Entrance Examinations), Toshihito Kanahori (Tsukuba University of Technology), Fukashi Kawane (Nihon University), Toshihikko Komada (Nihon University), Christopher Malon (Faculty of Mathematics, Kyushu University), Nobuyuki Ohtake (Tsukuba University of Technology), Seiichi Uchida (Kyushu University), Katsuhito Yamaguchi (Nihon University), Chen Yuan (Kyushu University), Kenji Hamada (DIGITALNAUTS.,Co.Ltd), Masayuki Naka (Tsukuba University of Technology), Misa Tachibana (Kyushu University).

17.2.3 Main Publications

M.Suzuki, T.Kanahori, N.Ohtake, K.Yamaguchi. An Integrated OCR Software for mathematical Documents and Its Output with Accessibility, *Computers Helping people with Special Needs*, 9th International Conference ICCHP2004, Paris, July 2004, Lecture Notes in Computer Sciences 3119, Springer (2004) pp.648−655.

T.Kanahori, M.Fujimoto and M.Suzuki. Authoring Tool for Mathematical Documents − Infty −, *3rd International Conference MKM2004*, Bialowieja, Poland, 2004, Sept. Online Proceeding.

See the project's website for papers on specific areas of the project.

17.2.4 Implementation Language

Information not available.

17.2.5 System Availability and Prerequisites

The programs developed within this system are: InftyEditor, InftyReader and ChattyInfty. All run only under Microsoft Windows (InftyReader only under XP!), and need as prerequisite LaTeX fonts; ChattyInfty requires Internet Explorer version 6, and the Microsoft speech API, version 4. A scanner is also needed.

With the exception of InftyEditor, the programs is usable free of charge for 30 days after the installation. Afterwards, a license must be bought from Science Accessibility Net : http://www.sciaccess.net/en/InftyReader/index.html.

See the descriptions of the programs in the Graphical tools subsection below.

17.3 Algorithm Libraries

No mathematical algorithm libraries are available.

17.3.1 Graphical Tools and Interfaces

InftyEditor is an authoring tool of mathematical documents, which eases the input of mathematical expressions [SuzukiEtAl03]. In its newer version it includes a handwriting recognition interface. Mathematical formulae written with the InftyEditor can be computed using various computer algebra systems with an OpenXM [MaekawaEtAl01] communication controller [KanahoriEtAl04].

InftyReader is an OCR program that recognizes scanned images of printed scientific documents (including mathematical formulae). It outputs the recognition results in various formats (XML for InftyEditor, LaTeX, MathML, Braille).

ChattyInfty is a math−expression editor for visually disabled persons to access, to write or to edit scientific documents. It is developed by incorporating a function of speech output into "InftyEditor" by making use of Microsoft speech API (description available on the project's website).

17.4 User Language

17.5 Mathematical Knowledge Bases

17.5.1 Available Theories and Knowledge Bases

The Infty project provides databases of mathematical symbols (i.e. characters, words, mathematical expressions) with different fonts, styles, etc. (stored as images, with text data attached to them) for use in character distinctions within character recognition programs [SuzukiEtAl05]. There are over 600.000 characters available in these databases.

17.5.7 Standardization, Inter−Operability

InftyEditor's Input from LaTeX source and Output into LaTeX, HTML, MathML and UEBC (Unified English Braille Codes). Via the OpenXM communication protocol [MaekawaEt-Al01], it can use computer algebra systems for computation [KanahoriEtAl04].

17.5.8 Web Access

On the project's website, an online recognition test site of InftyReader is available.

18 Isabelle

18.1 Short Description

Isabelle is a generic proof assistant which allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols (description found on their website).

18.2 Technical Information on the System

18.2.1 Name of the System and Website

Isabelle. http://isabelle.in.tum.de/ and http://www.cl.cam.ac.uk/research/hvg/Isabelle/Cambridge/

18.2.2 Project Leaders and Group

Project leaders: Larry Paulson (Cambridge University) and Tobias Nipkow (TU Munich)

Project group: Jean Martina, Tom Ridge, Susmit Sarkar, Peter Sewell, Rok Strniša (at Cambridge). Clemens Ballarin, Stefan Berghofer, Sascha Böhme, Amine Chaieb, Florian Haftmann, Alexander Krauss, Steven Obua, Christian Urban, Makarius M. Wenzel, Boris Borisov, Lukas Bulwahn, Tuan Dao, Markus Dörschmidt, Martin von Gagern, Johannes Hölzl, David Leuschner, Tobias Rittweiler (Munich).

18.2.3 Main Publications

Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. *Isabelle/HOL. A Proof Assistant for Higher−Order Logic*. LNCS, vol. 2283, 2002. Springer.

18.2.4 Implementation Language

ML.

18.2.5 System Availability and Prerequisites

The system is free of charge, and the current version is available − as binaries − for Linux and Mac OS X. To run it, one needs a Standard ML environment (they recommend Poly/ML) and the Proof General user interface with a working version of Emacs. Optionally a LaTeX installation is needed (for document preparations), a Java runtime environment (version 1.4 or later) for showing graphs, and external theorem provers.

18.3 Algorithm Libraries

18.3.1 Numerical, Discrete, Algebraic Libraries

18.3.2 Reasoners

Isabelle has a classical reasoner which is an ML functor that accepts certain information about logic and returns a suite of automatic tactics [Paulson07a]. Each tactic takes a collection of rules and executes a proof procedure. The classical reasoner contains generic tactics, not restricted to first−order logic. It can be instantiated for new logics, the Isabelle system includes the FOL, ZF, HOL and HOLCF logics.

18.3.3 Graphical Tools and Interfaces

Isabelle has a graph browser tool for visualizing theory dependency graphs. The tool is written in Java and can be used both as a standalone application and as an applet [Wenzel−Berghofer07]. The graph browser has two sub−windows: a directory tree window, and the graph window where the graph itself is displayed. In both windows, nodes (i.e. theories) can be collapsed or expanded to hide or show information.

There are various graphical interfaces to Isabelle. The most used is Proof General [Aspinall00], an interface based on Emacs. Within Proof General, the X−Symbol package provides several input methods to enter mathematical characters like ∈ in the text Another Isabelle graphical interface is which XIsabelle [CantOsols99], which uses the script language Tcl and the Tk Toolkit, and allows theory browsing, proof by clicking (avoiding to learn Isa−belle commands), advice on possible proof steps. In [TheryEtAl92] an interface for Isabelle is implemented using the Centaur programming environment.

18.4 User Language

18.4.1 Programming Language

To extend the Isabelle/Pure logic (see the subsection below) one remains within the Isabelle system, i.e. the Isabelle/Isar extension which hides the implementation language. To implement proof procedures, ML is used.

18.4.2 Logic Language for the Formulation of Mathematical Knowledge

Isabelle comes with several logics (which are hierarchies of theories) already implemented [Paulson07b]. All of them are formulated within Isabelle's meta logic, Isabelle/Pure:

− HOL (Higher Order Logic, and not the HOL system) is currently the most developed object−logic;

− ZF set theory, built on FOL (first−order logic);

− CCL is Martin Coen's Classical Computational Logic (for deriving programs from proofs, built upon classical FOL;

− LCF is a version of Scott's Logic for Computable Functions, built upon classical FOL;

− HOLCF, a version of lcf, defined as an extension of HOL;

− CTT is a version of Martin−Löf's Constructive Type Theory, with extensional equality. Universes are not included.

− Cube is Barendregt's λ−cube;

− several logics based upon the sequent calculus;

− LK is classical first−order logic as a sequent calculus;

− Modal implements the modal logics T, S4, and S43;

− ILL implements intuitionistic linear logic.

18.4.3 Mathematical Syntax

The syntax needed is the one of the logic involved to write the mathematical notions.

18.5 Mathematical Knowledge Bases

18.5.1 Available Theories and Knowledge Bases

The Archive of Formal Proofs is a collection of proof libraries, examples, and larger scientific developments, mechanically checked in the theorem prover Isabelle. It is organized in the way of a scientific journal. Submissions are refereed. [KleinEtAl07]. The library is under GNU GPL licence. It contains formalizations of theories starting form AVL Trees, Topology, Lazy Lists, to Fermat's Last Theorem for Exponents 3 and 4 and the Parametrization of Pythagorean Triples and Hensel's Lemma. For a complete list of the archive's content see its website.

Additionally, each of the logics included in Isabelle is stored as a formal theory.

A growing library of useful theories is distributed with the system. This library currently contains:

− Accessible−Part: The accessible part of a relation

− Sets and functions

− Big O notation

− Continuity an iterations of set transformers

− Implementation of natural numbers by integers

− Implementation of finite sets by lists

− Pi and function sets

- Multisets and properties of multisets
- Pairs of natural numbers
- Natural numbers with infinity
- Nested environments
- Permutations
- Primality on natural numbers
- Quotient types
- Binary words
- Zorn's Lemma
- Order on product types, on characters
- Proving equalities in commutative rings
- List prefixes, postfixes, lexicographic orders.

18.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The Archive of Formal Proofs mentioned above has a textual Google powered search.

The Iabelle system has a search engine that help users extract knowledge from the database. An important limitation of this tool is that it is limited at search in the currently loaded theory. The engine can search using patterns [NipkowEtAl07]. Some examples of simple patterns:

```
length
"_ # _ = _ # _"
"_ + _"
"_ * (_ - (_::nat))"
```

18.5.3 Tools for Inventing Mathematical Knowledge

18.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The Isabelle reasoner is available for proving mathematical knowledge relative to existing one.

18.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

18.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

The inference rules that users write can be first proved by Isabelle's reasoner. The rules are usually written as theorems, with indications to the system how they should be used in proving: as rewrite rules, as simplification rules, etc.

18.5.7 Standardization, Inter−Operability

The system has tools for translating Isabelle proof documents into LaTeX , and Html, and tools for code generating (for code in functional programming languages − SML, OCaml, Haskell) out of logical specifications. An interface to the CAS Maple is described in [BallarinEt−Al95].

18.5.8 Web Access

A web based GUI to Isabelle is described in [Kaliszyk06] and can be tried at http://prover.cs.ru.nl/ and is actually an interface for several provers (Coq, Isabelle, Matita, Lego and Plastic). Another web based GUI to Isabelle is described in a recent master thesis [Halvorsen07], but we found no web address to try it out.

18.6 Example of a Theory Exploration Session

Most of the users of Isabelle invoke the Proof General interface to the proving system. A work session within Iabelle means, actually, creating theories (i.e. a collection of types, func−tions, theorems). The general format of a theory T is

```
theory T
imports T1,· · · , Tn
begin
     declarations, definitions, and proofs
end
```

where *T1* , . . . , *Tn* are names of existing theories that T is based on, *declarations, defini−tions, proofs* are newly introduced concepts and proofs about them. The *Ti*s are parents of T, the knowledge stored in them is directly available in T [NipkowEtAl07].

The examples in this section are taken from the tutorial on Isabelle/HOL [NipkowEtAl07].

The theory ToyList defined below is based on the PreList theory, which contains many things but lists. It defines a datatype *list* which also introduces two constructors, Nil and Cons. Then it declares two functions, app(end) and rev(erse). The Cons constructor and app function have also an infix notation (’#’ and ’@’ respectively). The definitions of the two functions declared earlier follows, where the infix notation is used.

```
theory ToyList
imports PreList
begin
datatype ’a list = Nil                          ("[]")
                 | Cons ’a "’a list"            (infixr "#" 65)

consts app :: "’a list => ’a list => ’a list" (infixr "@" 65)
       rev :: "’a list => ’a list"

primrec
"[] @ ys        = ys"
"(x # xs) @ ys = x # (xs @ ys)"
```

```
primrec
"rev []         = []"
"rev (x # xs)   = (rev xs) @ (x # [])"
```

We want now to prove that reversing twice gives back the original ([simp] in the declaration below tells Isabelle that the theorem, should it be proved, should be used as a simplification rule:

```
theorem rev_rev [simp]: "rev(rev xs) = xs"
```

Isabelle's response will be:

```
1. rev (rev xs) = xs
```

We apply the induction tactic on *xs*:

```
apply(induct_tac xs)
```

And the reply from the system is given as a list of 2 goals, one for the base case and one for the step case:

```
1. rev (rev []) = []

2. ∧a list.
      rev (rev list) = list ⟹ rev (rev (a # list)) = a # list
```

We try now to solve the two goals automatically and call

```
apply(auto)
```

This will simplify the first goal, and the second one will become:

```
1. ∧a list.
      rev (rev list) = list ⟹ rev (rev list @ a # []) = a #
list
```

Here it is clear that the prover cannot continue without some additional lemma. We abandon the proof above and try to prove the lemma:

```
oops
lemma rev_app [simp]: "rev(xs @ ys) = (rev ys) @ (rev xs)"
```

To prove this lemma we use induction on the *xs* variable, but, again, we will see that we need another lemma, which in turn, to be proved will need another lemma. We list here the final script with all the lemmata − together with the proof tactics. The theorem we wanted to prove originally is last in this script.

```
lemma app_Nil2 [simp]: "xs @ [] = xs"
apply(induct_tac xs)
apply(auto)
```

```
done

lemma app_assoc [simp]: "(xs @ ys) @ zs = xs @ (ys @ zs)"
apply(induct_tac xs)
apply(auto)
done

lemma rev_app [simp]: "rev(xs @ ys) = (rev ys) @ (rev xs)"
apply(induct_tac xs)
apply(auto)
done

theorem rev_rev [simp]: "rev(rev xs) = xs"
apply(induct_tac xs)
apply(auto)
done
end
```

19 Macaulay 2

19.1 Short Description

Macaulay 2 is a computer algebra system devoted to supporting research in algebraic geometry, commutative algebra, and their applications [Macaulay2]. It aims at supporting efficient computation with a wide variety of mathematical objects. The system is based on its predecessor, Macaulay, was written by Dave Bayer and Mike Stillman. It is not a version 2 of Macaulay, but a fresh start. The Macaulay system is not further developed anymore.

19.2 Technical Information on the System

19.2.1 Name of the System and Website

Macaulay 2
http://www.math.uiuc.edu/Macaulay2/

19.2.2 Project Leaders and Group

The authors of the system are: Daniel R. Grayson (Professor Emeritus, retired, Department of Mathematics, University of Illinois) and Michael E. Stillman (Dept. of Mathematics, Conell University, Ithaca).

There is no group continuously working on the system (except for the two authors). There are several contributors to the system: Wolfram Decker, Neil Epstein, Anton Leykin, Harrison Tsai, Gregory Smith, Amelia Taylor, Carolyn Yackel, Bart Snapp.

19.2.3 Main Publications

D. Eisenbud, D.R. Grayson, M. Stillman, and B. Sturmfels, editors. *Computations in Alge−braic Geometry with Macaulay 2*, Algorithms and Computation in Mathematics Series, Volume 8, ISBN 978−3−540−42230−3. Springer, 2000.

19.2.4 Implementation Language

It has a core of mathematical algorithms implemented and compiled in C++ .

19.2.5 System Availability and Prerequsites

Macaulay 2 is copyright by Daniel R. Grayson and Michael E. Stillman. The system can be used under the terms of the GNU General Public License, version 2. It is available for a variety of operating systems.

As prerequisites, we mention Emacs (for Linux), Emacs for Windows (for the Windows operating system).

19.3 Algorithm Libraries

19.3.1 Numerical Library

Although not specified in the documentation of Macaulay 2, there must be some algorithms that do computations with numbers (especially integers). From other authors, the system includes a package that provides multiple precision arithmetic: the GNU MP4 package by Torbjörn Granlund, John Amanatides, Paul Zimmermann, Ken Weber, Bennet Yee, Andreas Schwab, Robert Harley, Linus Nordberg, Kent Boortz, Kevin Ryde, and Guillaume Hanrot.

19.3.2 Algebraic Library

The packages that come with Macaulay 2 include:

− a parser for polynomials;
− algorithms for D−modules;
− eliminating specified variables, and computing Sylvester resultant;
− an implementation of the Double Description Method (of Fourier, Dines and Motzkin) for converting between these two basic representations for convex cones;
− computing generic initial ideals of ideals in a polynomial ring;
− integral closures;
− creating lexicographic ideals and lex−plus−powers (LPP) ideals;
− lattice reduction (Lenstra−Lenstra−Lovasz bases);
− compute with points in affine and projective spaces;
− computations with components of ideals, including minimal and associated primes, radicals, and primary decompositions of ideals;
− algorithms for computing Rees algebras and integral closure of ideals;
− computations in the representation ring of GL(n);
− manipulating simplicial complexes.

The system also incorporates some code from other authors:

− the package SINGULAR−FACTORY2 which provides for factorization of polyno−mials (by G.−M. Greuel, R. Stobbe, G. Pfister, H. Schoenemann, and J. Schmidt);
− the package SINGULAR−LIBFAC3 uses FACTORY to enable the computation of characteristic sets (by M. Messollen).

19.3.3 Reasoners

19.3.4 Graphical Tools and Interfaces

19.4 User Language

19.4.1 Programming Language

The Macaulay 2 language is an interpreted language. Every object has a type (like Type, String, HashTable, Ring, etc). Expressions can include function calls, control structures (*for*, *while* loops), function definitions, and operator expressions. The engine implements rings, ring elements, and matrices as instances of low−level types, high−level types are based on these ones.

19.4.2 Logic Language for the Formulation of Mathematical Knowledge

Knowledge formulation is reduced to defining (i.e. programming) algorithms and executing them. The language used for this is the Macaulay 2 language.

19.4.3 Mathematical Syntax

The one of the Macaulay 2 language.

19.5 Mathematical Knowledge Bases

19.5.1 Available Theories and Knowledge Bases

19.5.2 Tools for Retrieval in Mathematical Knowledge Bases

19.5.3 Tools for Inventing Mathematical Knowledge

19.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

19.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

19.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

19.5.7 Standardization, Inter−Operability

The system has functions to translate expressions from the Macaulay2 language into html, tex, mathML. The TeXmacs system (http://www.texmacs.org) has an interface to Macauly 2. This interface allows sending commands to and receiving results from Macauly 2 from within the TeXmacs editor.

19.5.8 Web Access

19.6 Example of a Theory Exploration Session

The introductory examples are taken from [EisenbudEtAl00, preface], the Gröbner Bases examples are taken from the on−line documentation of the system [Macaulay2].

The output is textual, and the exponents of the variables are aligned with spaces, in the Macaulay 2 working environment − Emacs. For readability we have typed in the answers using Mathematica's 2−dimensional representation.

19.6.1 Introductory Examples

Macaulay 2 behaves like a standard computer algebra system, such as Mathematica or Maple: the user enters mathematical expressions at the keyboard, and the program computes the value of the expression and displays the answer [EisenbudEtAl00].

The input prompts are of the form 'i<number> : '. The output is displayed to the right of labels of the form 'o<number> = ' and, additionally, to the right of the labels of form 'o<number> : ' the type of the output is displayed. For example:

```
i1:3/5+7/11
   68
o1=--
   55
o1:QQ
```

(The symbol QQ above denotes the class of all rational numbers).

Define a quotient ring of a polynomial ring R over the rational numbers :

```
i3:R=QQ[x,y,z]/(x^3-y^3-z^3)
o3=R
o3:QuotientRing
```

Compute in the ring:

```
i4:(x+y+z)^3
o4 =
 3 x^2 y + 3 x*y^2 + 2 y^3 + 3 x^2 z + 6 x*y*z + 3 y^2 z + 3 x*z^2 + 3 y*z^2 + 2 z^3
o4 :
 R
```

Matrices over the ring are created as:

```
i5:b=vars R
o5 =  | x y z |
o5 : Matrix R^1  <---R^3

i6:c=matrix {{x^2,y^2,z^2}}
o6 =  | x2 y2 z2 |
o6 : Matrix R^1  <---R^3
```

Modules over the ring:

```
i7:M=coker b
o7 = cokernel | x y z |
o7 : R - module, quotient of R^1

i8:N=ker c
o8 = image {2} |  x   0   -y2  -z2  |
           {2} | -y  -z2  x2   0   |
           {2} | -y  y2   0    x2  |
o8 : R - module, submodule of R
```

Projective resolutions of modules are done with the command:

```
i9:res M
o9 = R^1 <-- R^3 <-- R^4 <-- R^4 <-- R^4
     0       1       2       3       4
o9 : ChainComplex
```

19.6.2 Computing Gröbner Bases

To compute the Groebner basis of an ideal ($x^2 y,\ xy^2 + x^3$) in the polynomial ring in four variables we proceed as follows:

Define the field:

```
i1:KK=ZZ/32003
o1=KK
o1:QuotientRing
```

The polynomial ring is:

```
i2:R=KK[x,y,z,w]
o2=R
o2:PolynomialRing
```

and the ideal is:

```
i3:I=ideal(x^2*y,x*y^2+x^3)
o3 = ideal (x^2 y, x^3 + x * y^2)
o3 : Ideal of R
```

The Gröbner basis of I under the graded reverse lexicographic order is:

```
i4:J=gens gb I
o4 =  | x2y  x3+xy2  xy3 |
o4 : Matrix R^1 <--- R^3
```

In Macaulay2, monomial orders are associated with a polynomial ring. For example, to specify the lexicographic order we use:

```
i5:R=KK[x,y,z,w,MonomialOrder ⇒Lex]
o5=R
o5:PolynomialRing
```

and:

```
i6:I=substitute(I,R)
o6 = ideal (x^2 y, x^3 + x * y^2)
o6 : Ideal of R

i7:gens gb I
o7 = | xy3 x2y x3 + xy2 |
o7 : Matrix R^1 < ---R^3
```

The Groebner basis is the same, since this is a small example. The polynomials are sorted in ascending monomial order by their lead terms, but otherwise the two Groebner bases are the same here.

20 Magma

20.1 Short Description

Magma is a system designed to solve computational hard problems in algebra, number theory, geometry and combinatorics. It provides a mathematically rigorous environment for computing with algebraic, number−theoretic, combinatoric, and geometric objects. The design of Magma emphasizes structural computation, i.e., the ability to construct canonical representa−tions of structures [MagmaWeb].

20.2 Technical Information on the System

20.2.1 Name of the System and Website

MAGMA Computational Algebra System
http://magma.maths.usyd.edu.au/magma/

20.2.2 Project Leader

Magma is produced and distributed by the Computational Algebra Group within the School of Mathematics and Statistics of the University of Sydney. No information about a project leader is available.

The current members of the development team (not all members of the Computational Algebra Group at Sidney) are: Geoff Bailey, John Cannon, Steve Donnelly, Claus Fieker, Damien Fisher, Sergei Haller, Michael Harrison, Allan Steel, Damien Stehl'e, Nicole Sutherland, Bill Unger, John Voight, Greg White ([MagmaWeb], HTML Help Document, Acknowledgments). Many other mathematicians have contributed to the system. For a complete list see the webpage.

20.2.3 Main Publications

Wieb Bosma, John Cannon, and Catherine Playoust: The Magma algebra system. I. The user language., J. Symbolic Comput. 24(3−4), 235−265, 1997.

20.2.4 Implementation Language

It has a kernel implemented in C. The rest of the system is implemented in the Magma language (see the User Language subsection below).

20.2.5 System Availability and Prerequisites

Magma is a non−commercial system, however, they recover the costs of distribution and support by charging its users. The subscription costs depend on the processor type, users (students, institution types), country. It is available for the most used operating systems and architectures (Linux, Mac OS X, Windows, Sparc Solaris, Macintosh, Alpha).

20.3 Algorithm Libraries

The information about the libraries listed below is taken from [MagmaOverview].

20.3.1 Groups, Semigroups and Monoids Libraries

The group theory is one of the traditional strengths of the Computational Algebra Group at the University of Sidney. The system offers most of the significant algorithms for finite groups and finitely represented groups: Permutation groups, Matrix groups, Finitely presented groups, Generic abelian groups, Finitely−presented abelian groups, Polycyclic groups, Soluble groups, Finite p−groups, Automorphism groups, Groups defined by rewrite systems, Automatic groups, Groups with elements given as straight−line programs, Black−box groups, Braid groups, Congruence subgroups of PSL(2, $\mathbb{R}$). The system also offers an extensive machinery for the representation theory of groups.

Magma has algorithms for finitely represented semigroups and monoids defined by rewrite systems.

20.3.2 Rings and Fields

Magma can understand fields (mainly local and global arithmetic fields), and their rings of integers and valuation rings. Numeric computations are done within these data structures.

• **The rational field $\mathbb{Q}$ and its ring of integers $\mathbb{Z}$**. Under this category, the system offers algorithms for: Multiple precision integer arithmetic; Integer multiplication and division via classical, Karatsuba, Toom and Schönhage−Strassen FFT methods; Greatest common divisor via Weber Accelerated GCD and Schönhage algorithms; Extended Greatest common divisor via Lehmer and Schönhage algorithms; Alternative representation of integers in factored form; Arithmetic functions: Jacobi symbol, Euler ϕ function, etc.

• **Residue class rings of $\mathbb{Z}$**. Arithmetic; square root, all square roots; Testing elements for: nilpotency, primitivity, regularity, zero−divisor; Order of a unit; Gcd and lcm; Location of a primitive element; Unit group; Functor from additive group to an object in the category of abelian groups; One or all square roots of an element

• **Primality and factorization.**

• **Univariate polynomial rings.** Includes algorithms for: Ring creation and ring operations; Creation of special polynomials; Arithmetic with polynomials; GCD and factorization; Arithmetic with ideals; Residue class rings of univariate polynomial rings.

• **Finite fields.** Contains algorithms for: Construction of fields, subfields, etc.; Computing traces and norms, orders of elements, characteristic polynomials, etc.; Roots and polynomial factorization; Discrete logarithms;

• **Galois rings.** Creation, basic operations with Galois rings.

• **Number fields and their orders** (general number fields, quadratic fields, and cyclotomic number fields). Includes algorithms for: Arithmetic of elements, construction of equation, maximal, arbitrary orders; Computation of Hilbert class fields; Orders and fractional ideals; Invariants; Diophantine equations; Automorphism; Class Field Theory; Quadratic and cyclotomic fields.

• **Rational function fields.** Creation of rational function fields, ring predicates, arithmetic, numerator and denominator, evaluation, derivative, etc.

• **Algebraic function fields.** Retrieval of information defining the field**;** Computation of basis;Construction of a function field with an extended constant field; etc.

• **Valuation rings.**

• **Real and complex fields.** Arithmetic, square root, continued fraction expansion of a real number, trigonometric functions, hyperbolic functions and their inverses, logarithm, dilogarithm, exponential, Bernoulli numbers, logarithmic integral, exponential integral, etc.

• **Newton polygons.** Construction; Finding faces, vertices and slopes; Algorithms for computing Puiseux expansions

• **Local rings and fields.** Construction; Arithmetic; Polynomial factorization.

• **Power series rings and Laurent series rings.** Arithmetic in these fields; Inversion of units; Derivative, integral; Square root, valuation; Exponentiation, composition, convolution, etc.

• **Lazy power series rings.** Creation of rings and elements; Arithmetic of elements; Retrieval of coefficients; Printing some specified terms of a series; Simple predicates on series; Derivative, integral and evaluation of series.

• **Algebraically closed fields.** Automatic extension of the field by the roots of any polynomial over the field, and operations on conjugates of roots; Basic arithmetic; All standard algorithms for rings over generic fields work over such fields; Minimal polynomial; Simplification of the field; Construction of the corresponding absolute field together with the isomorphism; Pruning of useless variables and relations.

20.3.3 Commutative Algebra

Under this title, the Magma system works with the following rings: Multivariate polynomial rings; Ideal theory of multivariate polynomial rings; Affine algebras; Modules over affine algebras.

The basic computational problems for commutative rings, which the Magma system treats, include: A canonical form for elements; Efficient arithmetic; A canonical representation (i.e., standard basis) for ideals; Arithmetic with ideals; Formation of quotient rings; Ideal decomposition, i.e., primary decomposition; The study of modules over rings.

20.3.4 Linear Algebra and Module Theory

The system can deal with: Matrix operations; Vector spaces; Free modules; Modules over Dedekind domains

• **Matrices.** Magma offers algorithms for: Representation of matrices; Arithmetic; Echelon form and nullspaces; Canonical forms.

• **Sparce matrices.**

• **Vector spaces.** Construction; Arithmetic; Subspaces and quotient spaces; Bases; Homomorphisms; Quadratic forms.

• **Free modules.** Basic operations; Homomorphisms.

• **Modules over Dedekind domains.** Creation of modules; Arithmetic with module elements; Submodules and quotients of modules; Determinant, dimension, pseudo−generators; Equality of modules, membership; Intersection of submodules; etc.

20.3.5 Lattices and Quadratic Forms

• **Lattices.** Constructions and operations; Algorithms for investigating properties of lattices; Reduction of lattices; Automorphisms; Neighbors and Genera; G−lattices.

• **Binary Quadratic Forms.** Prime form, random form; Reduction of forms; Composition and powering; Enumeration of reduced forms and reduced orbits; Treatment of fundamental and nonfundamental discriminants; etc.

20.3.6 Algebras

Defining algebras in Magma is made in terms of finite presentations, structure constants or as a matrix algebra. The algorithms for this area include algorithms for handling: Finitely presented associative algebras; General finite dimensional algebras (defined by structure constants); Finite dimensional associative algebras (defined by structure constants); Quaternion algebras; Group algebras; Matrix algebras; Finite dimensional Lie algebras (defined by structure constants); Quantized enveloping algebras (aka quantum groups).

20.3.7 Representation Theory

The main topics covered by this group of algorithms are: Modules over an algebra; K[G]−modules; Representations of groups; Character theory; Invariant theory.

20.3.8 Homological Algebras

• **Basic algebras** (finite dimensional algebras over a field). Algorithms: Creation from a sequence of projective modules and a path tree for each module; Creation of the basic algebra corresponding to the group algebra of a p−group over GF (p); Arithmetic; Extension and restriction of the coefficient ring; Tensor product; Opposite algebra; etc.

• **Chain complexes.** Algorithms for: Creation of a complex from a list of A−modules; Subcomplexes and quotient complexes; Operations on complexes: Splice, shift, direct sum; Exact extensions, zero extensions; Dual of a complex; Homology groups of a complex; Boundary maps; Construction of chain maps between complexes; Composition of chain maps; Image, kernel and cokernel of a chain map; etc.

20.3.9 Lie Theory

The Magma system has algorithms that treat: Coxeter matrices, Coxeter graphs, Cartan matrices, Dynkin digraphs, and Cartan names; Root systems; Root data; Coxeter groups; Coxeter groups as permutation groups; Reflection groups; Groups of Lie type; Finite−dimensional Lie algebras (See section on Algebras).

20.3.10 Algebraic Geometry

Magma has machinery for studying general algebraic varieties and families of special curves. The major categories include: Schemes and maps of schemes; Rational scrolls; Zero−dimensional schemes; Algebraic curves; Function fields and differentials of curves; Divisor groups and places of curves; Resolution graphs and splice diagrams; Graded rings and geometric databases; Plane conic curves and general rational curves; Elliptic curves; Hyperelliptic curves; Module of supersingular points; Modular forms; Modular symbols; Brandt modules; Modular curves; Modular Abelian Varieties.

20.3.11 Differential Galois Theory

• **Differential Rings and Fields.** Includes algorithms for: Construction of the rational differential field and the more general differential ring; Coercions, arithmetic and functionality for elements as for the underlying ring; Changing the derivation of a differential ring.; Extending the constant ring of a differential ring; Wronskian matrix and Wronskian determinant; The differential constant field of a rational differential field; Ring and field extensions of differential rings and fields; Construction of a differential ideal; Quotient rings, rings and field of fractions of differential rings and fields.

• Differential Operator Rings. Includes algorithms for: Creation of a differential operator ring; Coercion, arithmetic and simple predicates for elements; Accessing coefficients of elements; Changing the derivation of a differential operator ring; Changing the operator ring by extending the constant ring; Making a differential operator monic; Adjoint of an operator; Applying an operator to an element of its base ring; Euclidean algorithms, left and right (extended) GCD, (extended) left LCM; Companion matrix of an operator; etc.

20.3.12 Finite Incidence Structures

The algorithms in this category include: Counting functions; Partitions and tableaux; Graphs — directed and undirected; Networks; Incidence structures; Designs; Finite planes; Incidence geometry.

20.3.13 Error Correcting Codes

The coding theory module in Magma treats linear codes over finite fields, linear codes over Galois rings (including special functionality for codes over $\mathbb{Z}_4$), additive codes over finite fields, and quantum stabilizer codes.

20.3.14 Cryptography

Magma contains algorithms for creating and analyzing pseudo−random bit sequences (in general in the GF(2) universe). Some functions, like Berlekamp−Massey, apply also to sequences defined on arbitrary rings.

20.3.15 Reasoners

20.3.16 Graphical Tools and Interfaces

20.4 User Language

20.4.1 Programming Language

The language in which users can formulate algorithms is an imperative one with standard imperative−style statements and procedures. Among other features of the language we mention that it has dynamic typing, a simple notation for constructing sets and sequences, operations for sets and sequences − with emphasis on efficiency.

20.4.2 Logic Language for the Formulation of Mathematical Knowledge

The system is used for writing (programming) algorithms and execute them. It has no means of writing down theorems, definitions, etc. in a book− or paper−like style.

20.4.3 Mathematical Syntax

The Magma programming language.

20.5 Mathematical Knowledge Bases

20.5.1 Available Theories and Knowledge Bases

The theories available in Magma are called 'Mathematical Databases', and, typically, such a database contains complete classifications of structures of some given type, up to a specified bound. Some of these databases are an integral part of the algorithms in Magma [MagmaOver−view]. We enumerate below the databases currently available in Magma (state as of 2006). For details on the content of these databases and on their authors, see [MagmaOverview].

• **Group Theory:** Small Groups (the same that is available in the GAP system); The ATLAS Database (also included in GAP); Almost Simple Groups; Simple Groups; Perfect Groups; Transitive Groups; Primitive Groups; Permutation Representations; Irreducible Matrix Groups; Irreducible Soluble Groups; Finite Groups of Rational Matrices; Quaternionic Matrix Groups; Matrix Representations.

• **Number Theory:** Cunningham Factorizations; Irreducible polynomials; Conway polynomi-als; Galois Polynomials.

• **Algebraic Geometry:** Cremona database of Elliptic Curves; Stein−Watkins Database of Elliptic Curves; K3 Surfaces; 3−folds.

• **Topology:** Fundamental Groups of 3−manifolds.

• **Incidence Structures:** Simple Graphs; Strongly Regular Graphs; Hadamard Matrices; Skew Hadamard Matrices.

• **Linear Codes:** Best Known Binary Linear Codes; Best Known F_3Linear Codes; Best Known F_4Linear Codes.

20.5.2 Tools for Retrieval in Mathematical Knowledge Bases

20.5.3 Tools for Inventing Mathematical Knowledge

20.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

20.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

20.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

20.5.7 Standardization, Inter−Operability

20.5.8 Web Access

It has an on−line demo page, a calculator, based on code distributed under GPL and originally written by William Stein. The user can paste a Magma expression in the input field on the demo page. The output of the Magma computation will be shown in the output field.

20.6 Example of a Theory Exploration Session

The session examples below are taken from [CannonPlayoust96]. Generally, the system reacts at user inputs, i.e. users type in their commands, ended with ';', when a prompt ('>') is available, and the system immediately displays the results.

```
> 2+6;
8
```

Not all user inputs cause a response from the system. A missing semicolon tells the system there is more to be typed in by the user.

```
>P<x> :=PolynomialRing(IntegerRing());
>P;
Univariate Polynomial Algebra in x over Integer Ring

>(x^6−5*x^2+2)*(17*x^3−1);
17*x^9−x^6−85*x^5+34*x^3+5*x^2−2

>Factorization(x^8−1);
[
<x−1,1>,
<x+1,1>,
<x^2+1,1>,
```

```
<x^4+1,1>
]
```

Help about symbols can be obtained by typing '?' followed by the word one needs the help for.

The following example computes the area of the triangle when the lengths *a, b, c* of its sides are given (where 'gt' is 'greater than'):

```
>a:=6;b:=8;c:=10;
> if ((a+b) gt c) and ((b+c) gt a) and ((c+a) gt b) then
>           s:=(a+b+c)/2;
>           "Area is",Sqrt(s*(s-a)*(s-b)*(s-c)),"square units.";
>else
>           "Triangle is degenerate.";
>end if;
Area is 24.000000000000000000000000 square units.
```

When users write longer programs, they can write them in separate files and then load them with the following command:

```
load "filename";
```

Sequences are bracketed with square brackets, and sets with curly brackets. Sets are unordered, and elements can occur only once.

```
>t:={(-11)^2,(-7)^2,(-5)^2,(-3)^2,3^2,5^2,7^2,11^2};
>q:=[(-11)^2,(-7)^2,(-5)^2,(-3)^2,3^2,5^2,7^2,11^2];
>t, q;

{9,25,49,121}
[121,49,25,9,9,25,49,121]
```

Another possibility to define sets/sequences is to use scoping. For example, *t* could have been defined also as:

```
>t:={n^2:n in[-11..11 by 2]|IsPrime(n)};
{9,25,49,121}
```

A cartezian product can be defined as:

```
>{ <a,b,c> :a,b,c in[1..10]|a^2+b^2 eq c^2};
{ <6,8,10>, <4,3,5>, <3,4,5>, <8,6,10>}
```

Users can define both functions and procedures. Procedures have no return value. The function below computes $f(n, q) = \Pi_{i-1}^{n}(q^n - q^{i-1})$:

```
>f:=func<n,q|&*[q^n-q^(i-1):i in[1..n]]>;
>f(5,3);
475566474240
```

Here is a function that returns two values:

```
>counting:=function(n,r)
>                  x:=Factorial(n);
>                  y:=Factorial(r);
>                  z:=Factorial(n-r);
>                  p:=x div z;
>                  c:=p div y;
>                  return p,c;
>end function;
>per,com:=counting(5,2);
>per,com;
20 10
```

21 Maple

21.1 Short Description

Maple [MapleWeb] is a computer algebra system, an all purpose mathematical tool. It contains a computational engine with fully integrated numeric and symbolic computations.

21.2 Technical Information on the System

21.2.1 Name of the System and Website

Maple. http://www.maplesoft.com

21.2.2 Project Leaders and Group

The system is developed by Maplesoft in Waterloo, Ontario, Canada.

21.2.3 Main Publications

See the system's manuals.

21.2.4 Implementation Language

The user interfaces of Maple (since version 9) are implemented in C (Classic Worksheet) and Java (Standard Worksheet) respectively.

More generally, Maplets are Maple−based applets. Most applets found on the WWW are written in Java. Maplets are written in Maple; each Maplet element is, in turn, a Java Swing class. The Maple kernel is implemented in C/C++.

For the Maple libraries the implementation information is not available.

21.2.5 System Availability and Prerequisites

The system is commercial, available for a variety of platforms.

21.3 Algorithm Libraries

The body of mathematics covered by the system's algorithms is very large. This subsection lists a part of them. Additionally, there are many user−written packages and tools which (because of time restriction) we do not mention here.

21.3.1 Numerical Library

Maple has algorithms and packages for: Numerical routines, optimized to take advantage of hardware specifics, Approximations (Continued fractions, Floating−point arithmetic), Discrete Transforms, Integer Functions(e.g. Gaussian Integers), Interpolation and Curve Fitting, Intervals, Maple Numerics (floating-point arithmetic with extensions to symbolic data, to complex numerics), Graph theory, Algorithms for visualization of graphs, Differential geometry, Differential equation solving, Differential Algebraic equations, Polynomial real root finder, etc.

21.3.2 Discrete Math Library

Examples of algorithms in this area: Combinatorial functions (e.g. permutations, combinations, partitions), Combinatorial structures, Discrete Transforms (Fourier, Inverse Fourier Transform), Graph Theory, Summation and Difference Equations.

21.3.3 Algebraic Library

The most important algorithms in this category; Polynomials, Factorization and Root Finding, Gröbner bases, Matrix Polynomial Algebra (algebraic manipulation of matrices of polynomials), Orthogonal Polynomials, Polynomials Ideals, Rational Normal Forms, Skew Polynomials, etc.

21.3.4 Reasoners

Written by a group of researchers, there is a Maple−PVS interface available which can be downloaded from http://www.dcs.qmul.ac.uk/~hago/Maple−PVS/. Through this interface, a Maple user can access PVS from a Maple worksheet and use the PVS proof checking capabilities and libraries [MaplePVS].

21.3.5 Graphical Tools

Maple provides the plots and plottools packages for creating a variety of graphics in 2D and 3D. Animation is also supported.

21.4 User Language

21.4.1 Programming Language

Maple has its own procedural programming language.

21.4.2 Logic Language for the Formulation of Mathematical Knowledge

Not known

21.4.3 Mathematical Syntax

The syntax of Maple’s programming language.

21.5 Mathematical Knowledge Bases

21.5.1 Available Theories and Knowledge Bases

Tutorials available in Maple: PolynomialIdeals, Combinatorics, Optimization, Statistics, Vector Calculus.

There exists several e−books (or interactive books) and study guides created by the Maple users, using Maple, that have a corpus of mathematical knowledge in them. Mostly are available for a fee.

Through the Maple−PVS interface [MaplePVS], the PVS libraries are also available.

21.5.2 Tools for Retrieval in Mathematical Knowledge Bases

21.5.3 Tools for Inventing Mathematical Knowledge

21.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

21.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

21.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

21.5.7 Standardization, Inter−Operability

The system has packages for code generation (C, Fortran, Visual Basic®, JavaT, MATLAB®) and connectivity to Excel®, MATLAB, C, Java, Fortran. The TeXmacs system (http://www.texmacs.org) has an interface to Maple. This interface allows sending commands to and receiving results from Maple from within the TeXmacs editor.

21.5.8 Web Access

21.6 Example of a Theory Exploration Session

A Maple session (as in most of the CAS available) is a sequence of input−output actions: the user types in a command, evaluates it (send it in to the computation engine) and the system returns an answer (marked with '>>'). Here are a few examples:

Some numerical calculations:

```
2 ^ 100
>> 1267650600228229401496703205376
```

```
evalf(2^100)
 >> 1.26765 × 10^30
```

```
1/3+2/7
 >> 13/21
```

Some symbolic computations

```
3x−x+2
>> 2x+2
```

```
(x+2y+1)(x-2)^2
```
$$(x + 2\,y + 1)\,(x - 2)^2$$

```
expand(%)
```
$$x^3 - 3\,x^2 + 2\,y\,x^2 - 8\,y\,x + 8\,y + 4$$

Integration

```
integrate(x^n,x)
```
$$\frac{x^{n+1}}{n+1}$$

```
integrate(1/(x^4-a^4),x)
```
$$-\frac{1}{4}\,\frac{\ln(x+a)}{a^3} - \frac{1}{2}\,\frac{\arctan\left(\frac{x}{a}\right)}{a^3} + \frac{1}{4}\,\frac{\ln(x-a)}{a^3}$$

Sums

```
sum(i^2,i)
```
$$\frac{1}{3}\,x^3 - \frac{1}{2}\,x^2 + \frac{1}{6}\,x$$

Example of a function definition:

```
f:=n→  n!;
```

or

```
f :=proc(n) n! end;
```

22 Mathematica

22.1 Short Description

Mathematica is an all−purpose mathematical software package which integrates numeric and symbolic calculations, graphics, and has a powerful programming language.

22.2 Technical Information on the System

22.2.1 Name of the System and Website

Mathematica. http://www.wolfram.com/

22.2.2 Project Leaders and Group

The system is developed by Wolfram Research Inc. IL USA.

22.2.3 Main Publications

See the sytem's manuals.

22.2.4 Implementation Language

C/C++, Java and Mathematica's language.

22.2.5 System Availability and Prerequisites

The product is commercial, available for the most operating systems. For some applications of Mathematica several other programs must be installed (e.g. some Java virtual machine).

22.3 Algorithm Libraries

The body of mathematics covered by the system's algorithms is very large. This subsection lists a part of them.

22.3.1 Numerical Library

Mathematica can do numerical evaluation to any precision. The types of numbers built−in the system are integers, reals, rationals, and complex. The algorithms in this area deal with: Primes and their distribution, Integer factorization, Divisibility and divisors, Congruences, Partitions, Representations of sums and powers, Recurrence and sum functions, Digit representations, Analytic number theory, Algebraic number theory, Combinatorial functions.

22.3.2 Discrete Math Library

The system has algorithms for dealing with: lists, tuples, sets, permutations, strings and digits, graphs and trees, lattices and knot data, cellular automata and turing machines.

22.3.3 Algebraic Library

Mathematica has algorithms for: Operations on vectors, matrices, Linear systems, Minimization problems, Sparse arrays, etc. It knows how to operate with polynomial elements, polynomial systems, and has algorithms for polynomial algebra like: factoring and decomposition of polynomials, division, finding generic solutions for variables.

22.3.4 Calculus

Total and partial derivatives, integration, integral transformations, power series, symbolic solutions to equations, numerical calculus.

22.3.5 Reasoners

22.3.6 Graphical Tools

It has a powerful engine for displaying graphics of mathematical objects.

22.4 User Language

22.4.1 Programming Language

The Mathematica language is a symbolic language which covers a broad range of programming paradigms. It provides support for pattern matching, procedural programming, functional programming, programming by rules.

22.4.2 Logic Language for the Formulation of Mathematical Knowledge

22.4.3 Mathematical Syntax

The syntax of the Mathematica programming language. It supports two dimensional input of mathematical expressions.

22.5 Mathematical Knowledge Bases

22.5.1 Available Theories and Knowledge Bases

The system itself does not have repositories of mathematical knowledge. However, on the company's website, several repositories are available. The ones we mention here are The Functions Site and MathWorld. There is also a library of Mathematica−related materials which contains books, articles, Mathematica programs, etc. together with bibliographical information. In most of the cases, this information if freely available (see http://library.wolfram.com/).

The Functions Site [FunctionsSite] contains over 80000 formulae defining functions from the following categories: Elementary Functions, Constants, Bessel, Airy, Struve Functions, Integer Functions, Polynomials, Gamma, Beta, Erf, Hypergeometric Functions, Elliptic Integrals, Elliptic Functions, Zeta Functions and Polylogarithms, Mathieu Functions, Complex Components, Number Theory Functions, Generalized Functions.

The MathWorld repository is an on−line mathematical encyclopedia developed and maintained by Eric Weisstein, with many contributors from the academic world. It has over 12000 entries from the following domains: Algebra, Applied Mathematics, Calculus and Analysis, Discrete Mathematics, Foundations of Mathematics, Geometry, History and Terminology, Number Theory, Probability and Statistics, Recreational Mathematics, Topology. For more details see [MathWorld].

22.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The Functions Site can be searched either by a google−based search engine or a mathematical content−based search. The MathWorld repository only has a full−text search capability available.

22.5.3 Tools for Inventing Mathematical Knowledge

22.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

Most of the functions on the Function Site have been verified using the command FullSimplify on test equations.

22.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

22.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

22.5.7 Standardization, Inter−Operability

The system provides translators to LaTeX, Html, and MathML. Via MathLink, it can connect to Java, C/C++, .Net.

22.5.8 Web Access

The company that develops Mathematica, also has a webMathematica product, which allows users who bought it to create websites with interactive calculations and visualizations, websites which integrate Mathematica. An example of such an interface can be found at http://integrals.wolfram.com/ .

A lot of further information − including a complete documentation of the system, and technical support − is available on the company’s website.

22.6 Example of a Theory Exploration Session

A Mathematica session is a sequence of input−output actions: the user types in a command, evaluates it (send it in to the Mathematica kernel) and the system returns an answer (marked with '>>'). Here are a few examples:

Some numerical calculations:

```
2 ^ 100
>> 1267650600228229401496703205376
```

```
2^100//N
>> 1.26765 × 10^30
```

```
1/3+2/7
>> 13/21
```

Some symbolic computations

```
3x−x+2
>> 2+2 x
```

```
(x+2y+1)(x−2)^2
>> (−2 + x)^2 (1 + x + 2 y)
```

```
Expand[%]
>> 4 − 3 x^2 + x^3 + 8 y − 8 x y + 2 x^2 y
```

Integration

```
Integrate[x^n,x]
>> x^(1+n)/(1+n)
```

```
Integrate[1/(x^4−a^4),x]
>> −ArcTan[x/a]/(2 a^3) + Log[a−x]/(4 a^3) − Log[a+x]/(4 a^3)
```

Sums

```
Sum[x^(i (i+1)),{i,1,Infinity}]
>> (−2 x^(1/4) + EllipticTheta[2,0,x])/(2 x^(1/4))
```

Example of a function definition:

```
f[n_] := n!
```

or

```
f[n_] := n  f[n-1] ;
f[1] = 1
```

or

```
f[n_] := Module[{t = 1, i},
          For[i =1, i <=n, i++, t *= i];
       t]
```

23 MathLang

23.1 Short Description

MathLang is a mathematical language and also the associated framework for writing mathematical texts. The mathematical language is developed from the mathematical vernacular [Bruijn87] and the weak type theory [KamareddineNederpelt04]. Mathlang allows computerisation (i.e. storing on the computer) of mathematical texts written in a Common Mathematical Language [KamareddineNederpelt04] and provides methods for adding, checking and displaying various information aspects [KamareddineWells07]. One aspect is a weak type system assigning categories (term, statement, adjective etc.) to parts of text, binds names to meanings, and checks that a kind of *grammatical sense* is maintained. Another aspect is combining mathematical information and visual presentation, associating natural language text to mathematical information. A third aspect allows identifying chunks of text and marking their roles (e.g. theorem, explanation, example, section) and indicating *relationships* between chunks.

The framework also helps the user to formalize the mathematical document, obtaining a document that can be checked by a proof checker (like Mizar and Isabelle).

23.2 Technical Information on the System

23.2.1 Name of the System and Website

MathLang, no website known. The webpage of the Ultra group involved in the development of MathLang is: http://www.macs.hw.ac.uk/ultra/.

23.2.2 Project Leaders and Group

Leader: Prof. Fairouz Kamareddine (ULTRA Group, Heriot-Watt University, Edinburgh, Scotland, UK).

Group: Manuel Maarek, Krzysztof Retel and J. B. Wells.

23.2.3 Main Publications

Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel and Joe Wells, Digitised Mathematics: Computerisation vs. Formalisation. *Review of the National Center for Digitization*, Volume 10, pages 1−8, Faculty of Mathematics, Belgrade, Serbia, 2007.

Fairouz Kamareddine and J.B. Wells, Computerising mathematical texts in MathLang, *Second Workshop on Logical and Semantic Frameworks, with Applications*, Ouro Preto, Minas Gerais, Brazil, 28 August 2007. ENTCS, Ayala−Rincon and Heusler (editors). ISSN: 1571−0661, January 2008. Elsevier.

23.2.4 Implementation Language

The type checker that analyzes the MathLang texts is implemented in Camlp4 (the parser) and OCAML (type inferences).

23.2.5 System Availability and Prerequisites

On request from the system's authors.

23.3 Algorithm Libraries

23.4 User Language

23.4.1 Programming Language

23.4.2 Logic Language for the Formulation of Mathematical Knowledge

Mathematics can be written in the logic chosen by the user. The MathLang system processes pieces of mathematics expressed in Common Mathematical Language. In certain cases, 'incorrect' content can also be processed (like formulae of the kind a = b = c).

23.4.3 Mathematical Syntax

MathLang is also the mathematical language of the system. It has three aspects:

- The Core Grammatical Aspect (CGa) which provides a kind of grammar for well−formed mathematics. It has grammatical categories and allows checking for some basic well-formedness conditions (e.g. origin of all names and symbols can be tracked).

- The Text and Symbol Aspect (TSa) which allows the integration of normal typesetting and authoring software (with representations like LaTeX, XML or TEXmacs) with the mathematical structure represented by CGa.

- The Document Rhetorical Aspect which deals with identifying and relating portions of text (e.g. a theorem is a chunk of text and so is its proof).

23.5 Mathematical Knowledge Bases

23.5.1 Available Theories and Knowledge Bases

Using Mathlang, parts of "A Compendium of Continous Lattices" [GierzEtAl80] (formalized by Mizar), Landau's "Foundations of Analysis" [Landau51] (formalized by Automath) and Euclid's "Elements" were computerized [Heath56].

23.5.2 Tools for Retrieval in Mathematical Knowledge Bases

23.5.3 Tools for Inventing Mathematical Knowledge

23.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

23.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

23.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

23.5.7 Standardization, Inter−Operability

Using the informations from CGa and TSa (see User Language/Mathematical Syntax), a MathLang text can be exported to Mizar, first as a proof sketch with holes and then as a fully completed proof [KamareddineEtAl07a].

23.5.8 Web Access

23.6 Example of a Theory Exploration Session

MathLang is used with TEXmacs, where some macros have been defined to be used by authors of MathLang documents. Using these macros, authors can select and annotate pieces of the mathematical text they are working on. Another way of storing mathematical documents is by using the XML format. To see how Mathlang can be used by an author we point the reader to [KamareddineEtAl07b] and [KamareddineEtAl07c].

24 MATHsAiD

24.1 Short Description

MATHsAiD [McCaslandBundy06] is a system for automated ***discovery of ''interesting'' mathematical theorems***. Its discovery mechanism comprises two stages: generation and filtering. It operates on equivalence classes of (for now first-order) formulae and terms (equivalence meaning respectively logical equivalence and equality). Interesting results were obtained in group theory, set theory, and arithmetic. An extension for higher-order formulae is planned.

24.2 Technical Information on the System

24.2.1 Name of the System and Website

Mechanically Ascertaining Theorems from Hypotheses, Axioms and Definitions (MATHsAiD).

http://dream.inf.ed.ac.uk/projects/mathsaid/

24.2.2 Project Leaders and Group

The following persons are associated with the project: Alan Bundy, Roy McCasland, Patrick Smith and Simon Colton.

24.2.3 Main Publications

Roy McCasland, Alan Bundy, and Patrick Smith. Ascertaining Mathematical Theorems, Electronic Notes in Theoretical Computer Science, Volume 151 (1), 2006, pp21 − 38. http://dx.doi.org/10.1016/j.entcs.2005.11.021

Roy McCasland and Alan Bundy. MATHsAiD : a Mathematical Theorem Discovery Tool, Proceedings of SYNASC' 06, pp17 − 22, IEEE Computer Society Press, 2006. http://dream.inf.ed.ac.uk/projects/mathsaid/MTDT.pdf

Roy McCasland, Alan Bundy, and Serge Autexier. Automated discovery of inductive theorems.In From Insight to Proof: Festschrift in Honor of A.Trybulec. R.Matuszewski and P.Rudnicki, editors, University of Bialystok, Bialystok, 2007. http://www.inf.ed.ac.uk/publications/online/1097.pdf

24.2.4 Implementation Language

The interface of MATHsAiD is implemented in Java and its logical engine in Prolog.

24.2.5 System Availability and Prerequisites

The system is available upon request from Roy McCasland, under a GPL−like licence. For running MATHsAiD an installation of Java 5.0 is required. For now, because of the Prolog library that comes with it, MATHsAID works only on Windows.

24.3 Algorithm Libraries

24.3.1 Numerical, Discrete, Algebraic, etc. Libraries

24.3.2 Reasoners

24.3.3 Graphical Tools and Interfaces

MATHsAID has a graphical user interface implemented in Java. The user can input new theories (given by axioms/definitions) or add some extra axioms/definitions to existing theories. Then based on a theory (or a set of axioms in a given theory), the system produces the resulting theorems. The user can manually insert extra hypotheses and "terms of interest" (the results will be then filtered, obtaining only theorems containing this terms).

24.4 User Language

24.4.1 Programming Language

24.4.2 Logic Language for the Formulation of Mathematical Knowledge

For now, the system uses first-order logic for the formulation of mathematical knowledge. An extension towards higher-order terms is planned.

24.4.3 Mathematical Syntax

MATHsAiD has its own custom-tailored syntax.

24.5 Mathematical Knowledge Bases

24.5.1 Available Theories and Knowledge Bases

The theories investigated up to now are: Sets, Groups and three different representations of the natural numbers: Positive Integers (with ≤), Nstar and Natural Numbers (focusing on the induction aspect).

24.5.2 Tools for Retrieval in Mathematical Knowledge Bases

24.5.3 Tools for Inventing Mathematical Knowledge

MATHsAiD automatically discovers and proves theorems (lemmas, corollaries, etc.) from a given set of axioms and definitions (supplied by the user).

Its main components are the automatic hypothesis generator, the theorem generator and the theorem filter[McCaslandBundy06].

Given the set of axioms and definitions, ***the hypothesis generator*** (HG) builds up:

- a finite sequence $\{H_i\}_{i=1}^{n}$, where H_i is a set of hypotheses and
- a selection of axioms and terms of interest corresponding to each H_i.

This allows the system to concentrate on local aspects of the theory, and thus (in the following components) to build theorems in layers, rather than build them all at once. The HG also looks at the existing axioms and theorems for finding converses. If such a converse can exist, it is passed over to the theorem generator that will attempt to prove it. The order of the outputs of HG matter: the simpler hypotheses will be fed to the theorem generator before more complicated ones will.

The ***theorem generator*** takes as input a set of hypotheses H_k, asserts them and applies a forward-chaining process, using the build-in first-order theorem prover, deriving all conclusions. Conclusions satisfying certain criteria (exemplified below) are then asserted, then the process starts over, until no more new conclusions can be found. The resulting conclusions are passed over to the theorem filter.

The most important criteria a conclusion has to satisfy is that it should not be a conclusion of an axiom, the "trivial" conclusion of a theorem or "trivially derived" from the existing knowledge (axioms, theorems). A more detailed description of triviality is found in [McCaslandBundy06].

The ***theorem filter*** takes these conclusions, and runs them through a number of tests, like irredundancy (checking whether all hypothesis from H_k are needed to prove the conclusion) or simplicity (based on a heuristic detailed in [McCaslandetal06]). All conclusions failing a test are eliminated.

All remaining conclusions are coupled back with H_k, and recorded as theorems.

The system was recently extended towards automated discovery of inductive theorems [McCaslandetal07].

24.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

MathsAid contains an inductive equation prover (used only in the background), as stated in [McCaslandetal07]. Further details are not known.

24.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

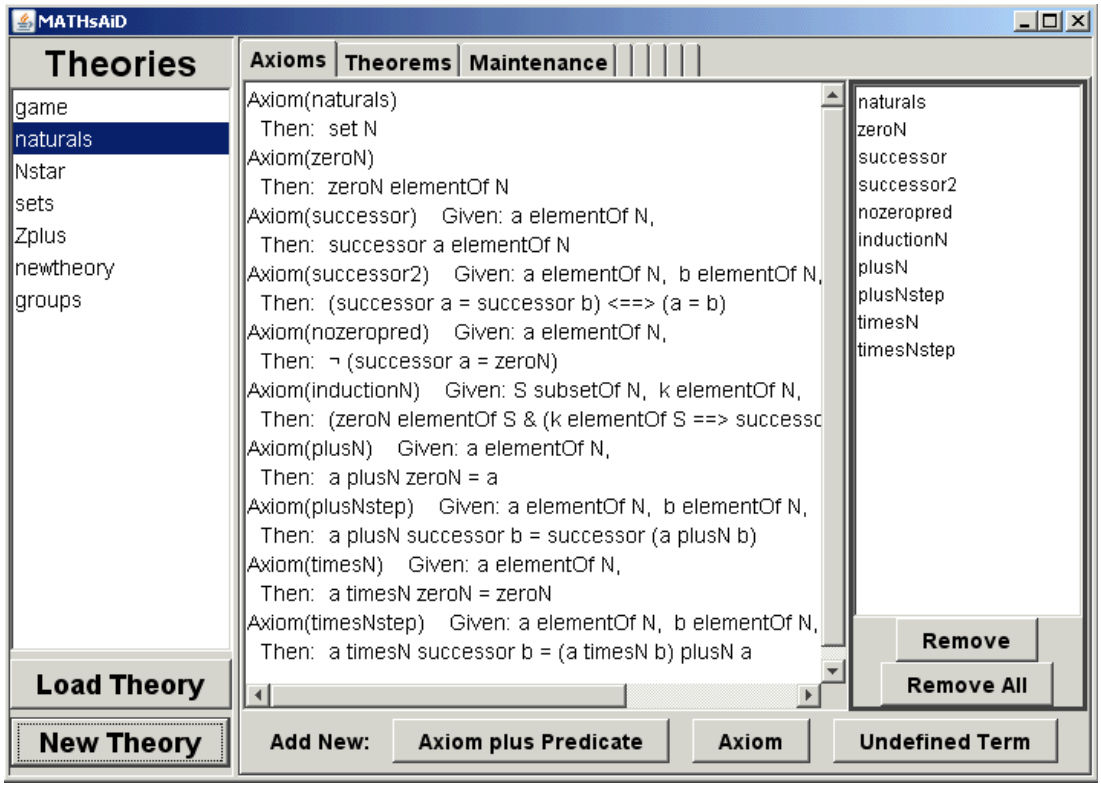
24.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

24.5.7 Standardization, Inter−Operability

24.5.8 Web Access

24.6 Example of a Theory Exploration Session

The user can select a theory, or add a new one:



Then, by clicking on the Theorems tab, he can start an automated discovery on, in this case, the theory of natural numbers. (or on some of the axioms from the theory of naturals).

The resulting theorems will be:

Theorem 1

```
Given:a elementOf N;
then:a plusN (successor zeroN)=successor a.
```

Theorem 2

```
Given:c elementOf N;b elementOf N;a elementOf N;
then:(a plusN b) plusN c=a plusN (b plusN c).
```

Theorem 3

```
Given:a elementOf N;
then:zeroN plusN a=a.
```

Theorem 4

```
Given:a elementOf N;b elementOf N;
then:(successor b) plusN a=successor b plusN a.
```

Theorem 5

```
Given:b elementOf N;a elementOf N;
then:a plusN b=b plusN a.
```

Theorem 6

```
Given:a elementOf N;
then:a timesN (successor zeroN)=a.
```

Theorem 7

```
Given:c elementOf N;b elementOf N;a elementOf N;
then:a timesN (b plusN c)=(a timesN b) plusN (a timesN c).
```

Theorem 8

```
Given:c elementOf N;b elementOf N;a elementOf N;
then:(b plusN c) timesN a=(b timesN a) plusN (c timesN a).
```

Theorem 9

```
Given:c elementOf N;b elementOf N;a elementOf N;
then:(a timesN b) timesN c=a timesN (b timesN c).
```

Theorem 10

```
Given:a elementOf N;
then:zeroN timesN a=zeroN.
```

Theorem 11

```
Given:a elementOf N;
then:(successor zeroN) timesN a=a.
```

Theorem 12

```
Given:b elementOf N;a elementOf N;
then:a timesN b=b timesN a.
```

25 Maxima

25.1 Short Description

Maxima is an open−source computer algebra system based on DOE−MACSYMA, the oldest computer algebra system. It is It provides both symbolic computation and high−precision integer and floating−point arithmetic.

25.2 Technical Information on the System

25.2.1 Name of the System and Website

Maxima. http://maxima.sourceforge.net/index.shtml

25.2.2 Project Leaders and Group

The project is voluntarily developed by a group of 26 people (status as of end of 2007).

25.2.3 Main Publications

The Maxima Reference Manual. Version 5.13.0. Available at http://maxima.sourceforge.net/docs.shtml

Paulo Ney de Souza, Richard J. Fateman, Joel Moses, Cliff Yapp. The Maxima Book. 19th September 2004. Available from http://maxima.sourceforge.net/docs.shtml

25.2.4 Implementation Language

It is written in Common Lisp. Additionally has over 26,000 lines of Fortran, and over 600 lines of C code.

25.2.5 System Availability and Prerequisites

The system is open−source, available freely under GPL. Binaries for Windows and versions of Linux are available. If the system is compiled from the sources, some version of Common Lisp is needed.

25.3 Algorithm Libraries

Numerical Libraries: Numerical constants, Manipulation of expressions involving loga−rithms, Numerical integration, Fourier transforms, etc., Number theory.

Calculus: Limits of expressions**,** Differential calculus, Integral calculus, Defining and solving (differential) equations, Taylor and power series.

Algebra: Standard forms for polynomials, and functions operating on them, Creating and working with arrays, Matrix operations, Indicial/Component/Algebraic Tensor Manipulations, Symmetries, Groebner bases.

Special functions: Bessel functions of the 1st and 2nd kind, Modified Bessel function of the 1st and 2nd kind, Hermite polynomial , Legendre function, Struve H function, Struve L func−tion, Generalized Hypergeometric function, Gamma function, Incomplete gamma function, Tail of incomplete gamma function, Whittaker functions of the 1st and 2nd kind,, Complement of the erf function, Complete elliptic integral of the first kind, Parabolic cylinder function, Elliptic Functions and Integrals.

There are also contributed packages, which treat various specific mathematical areas and utilities for manipulating mathematical expressions, e.g.: Additional routines for ODEs, Descrip−tive statistics, Jordan matrices, Probability distributions, Generating function of sequences, Functions for working with Groebner bases, Implicit derivatives, Interpolation package, L−BFGS unconstrained minimization package, Functions for linear algebra, Least squares, Newton’s method, Orthogonal polynomials, Romberg method for numerical integration, Simpli−fication rules and functions, Linear recurrences, Statistical inference package, Stirling formula, String processing, Units and dimensions package, Functions for hypergeometric summation.

The system has several plotting packages available: A Maxima−Gnuplot interface, Graphics for dynamical systems and fractals, Direction fields plots.

25.3.1 Reasoners

25.3.2 Graphical Tools and Interfaces

At its core, Maxima is a command line system. There are various graphical interfaces to Maxima some of them widely used, like for example, the Emacs mode for Maxima (which is − however − textual!). Another major mode for Emacs is Emaxima, which allows the user to insert Maxima sessions and code in a LaTeX document. Xmaxima is currently the default interface environment on Windows (it is available under Linux, too). It is based on Tcl/Tk, and acts largely as an enhanced command prompt.

The TeXmacs editor has a plugin for Maxima, which uses the TeX output Maxima can produce. Another known Maxima interface is wxMaxima based on wxWidgets, which can display mathematical formulae in a two−dimensional display format.

25.4 User Language

25.4.1 Programming Language

Maxima has a full programming language which is similar to Pascal and Algol. Semantically it is similar to Lisp

25.4.2 Logic Language for the Formulation of Mathematical Knowledge

None specified.

25.4.3 Mathematical Syntax

Maxima defines its own syntax mathematical syntax. Mathematical knowledge can be expressed by Maxima expressions which are atoms or lists consisting of an operator and its arguments.

25.5 Mathematical Knowledge Bases

25.5.1 Available Theories and Knowledge Bases

25.5.2 Tools for Retrieval in Mathematical Knowledge Bases

25.5.3 Tools for Inventing Mathematical Knowledge

25.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

25.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

25.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

25.5.7 Standardization, Inter−Operability

Maxima has a translator to Fortran and can also output TEX strings (facility used by the TEXmacs editor).

25.5.8 Web Access

25.6 Example of a Theory Exploration Session

We will present here a very short interaction session with Maxima, without showing all its capabilities.

Maxima can be used as a calculator (note that the user input must be ended with ';', the input and output are differently marked):

```
(%i1) 9+7;
(%o1) 16
(%i2) −17*19;
(%o2) −323
(%i3) 10/2;
(%o3) 5
```

Previous input can be accessed using the '%' symbol:

```
(%i4) % - 10;
(%o4) -5
(%i5) %o1 * 3;
(%o5) 48
```

Expanding a polynomial

```
(%i1) (x + 3*y + x^2*y)^3;
                           2             3
(%o1)                    (x  y + 3 y + x)
(%i2) expand (%);
       6  3      4  3       2  3       3      5  2       3  2
(%o2) x  y  + 9 x  y  + 27 x  y  + 27 y  + 3 x  y  + 18 x  y
                                          2      4        2
3
                                  + 27 x y  + 3 x  y + 9 x  y + x
```

If we want to substitute x with 5/z then:

```
(%i3) %o2, x=5/z;
            2        3                  2                3
      135 y    675 y    225 y   2250 y    125   5625 y    1875 y
(%o3) ------ + ------ + ----- + ------- + --- + ------- + ------
        z          2        2         3       3       4         4
                  z        z         z       z       z         z
                                               2           3
                                        9375 y    15625 y
3
                                     + ------- + -------- + 27 y
                                          5          6
                                         z          z
```

Solving equations:

```
(%i6) a + b*c = 1;
(%o6)                        b c + a = 1
(%i7) b - a*c = 0;
(%o7)                        b - a c = 0
(%i8) a + b = 5;
(%o8)                         b + a = 5
(%i9) solve ([%o6, %o7, %o8], [a, b, c]);
            25 sqrt(79) %i + 25      5 sqrt(79) %i + 5
(%o9) [[a = -------------------, b = -----------------,
            6 sqrt(79) %i - 34       sqrt(79) %i + 11
    sqrt(79) %i + 1        25 sqrt(79) %i - 25
c = ---------------], [a = -------------------,
          10               6 sqrt(79) %i + 34
    5 sqrt(79) %i - 5        sqrt(79) %i - 1
b = -----------------, c = - ---------------]]
    sqrt(79) %i - 11               10
```

26 MBase

26.1 Short Description

MBase is a structured and distributed repository of mathematical knowledge [Kohlhase−Franke01]. It is implemented as a mathematical service of the MathWeb mathematical software bus [FrankeKohlhase99], which is in turn built on top of OMDoc. MathWeb connects a wide range of mathematical services (including interfaces to Omega, Bliksem, Otter, SPASS, RDL, Mace, Maple, CoCoA).

26.2 Technical Information on the System

26.2.1 Name of the System and Website

MBase: A Mathematical Knowledge Base, http://www.mathweb.org/mbase/

26.2.2 Project Leaders and Group

The following persons are currently developing the system: Andreas Franke, Michael Kohlhase and Paul Libbrecht.

26.2.3 Main Publications

M. Kohlhase and H.M.Franke. MBase: Representing Knowledge and Content for the Integration of Mathematical Software Systems. *Journal of Symbolic Computation*, 2001, 32, 365−402.

Michael Kohlhase and Andreas Franke. System Description: MBase, an Open Mathematical Knowledge Base. *CADE−17 Proceedings*, LNAI 1831, 2000, pp.455−459.

26.2.4 Implementation Language

MBase is based on mOZart/Oz [mOZart], java (JDBC), a relational data base management system (the implicit one that comes with the system is MySQL), and on OMDoc.

26.2.5 System Availability and Prerequisites

MBase is distributed under the terms of the GNU General Public License. The whole MBase application can be used under Linux or (experimentally) under Solaris. The only other prerequisites are mOZart and MySQL, which come with the system. (At the time of writing this document, the system could not be found to be downloaded anymore).

26.3 Algorithm Libraries

MBase is linked over the MathWeb mathematical software bus to a variety of computer algebra systems, like Maple and CoCoA. As such, it has access to the algorithms implemented in those systems. Also, libraries of some theorem provers (e.g., TPS proving system (http://gtps.math.cmu.edu/tps.html, Ωmega) have been translated to OMDoc and loaded into MBase.

26.3.1 Numerical, Discrete, Algebraic, Etc. Libraries

26.3.2 Reasoners

26.3.3 Graphical Tools and Interfaces

26.4 User Language

26.4.1 Programming Language

26.4.2 Logic Language for the Formulation of Mathematical Knowledge

26.4.3 Mathematical Syntax

MBases uses the mathematical syntax of OMDoc.

26.5 Mathematical Knowledge Bases

26.5.1 Available Theories and Knowledge Bases

The theories covered in MBase are: Zermelo−Fraenkel Set Theory, Neumann−Bernays−Goedel Set Theory, Group Theory, Rings, Fields, Reals, Integers, Naturals, Polynomials, Calculus, Topology, etc. The online demo of MBase contains several preloaded theories. These are imported from the OMDoc libraries of Omega, but also from TPS, and from OpenMath content dictionaries in OMDoc form. (By the time of writing this document, the on−line demo was not accessible.)

26.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The online demo of MBase allows browsing through the theories and retrieval facilities like: (a) textual search on all mathematical elements available in the knowledge base (called Quick−Search); (b) textual search restricted to names of theories,axioms,symbols,definitions and theorems; (c) pattern search for terms and formulae. The pattern matching mechanism (applicative higher−order) is inherited from Oz, the underlying programming language.

As announced in [KohlhaseFranke01], there are plans to improve the system by adding inference procedures like higher−order matching and (logic−internal) structuring mechanisms like theory morphisms.

The queries made are based on the underlying programming language Oz. Oz has a notion of "tuple" which consists of a label and a number of values, e.g. *foo (a b c)* is a 3-tuple with label *foo*. Identifiers starting with lowercase letters (like *a, b, c, foo*) are constants. The variables will start with a capital letter (like *F, X, Y, Z*). The "_" is considered to be an anonymous vari−able.

Pattern matching will be performed on the internal representation of terms and formulae (i.e. on variable−free tuple trees). E.g. *foo (a b c)* will match *foo (X b c)* but not *foo(_ _)* or *foo (X b X)*.

26.5.3 Tools for Inventing Mathematical Knowledge

26.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

26.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

26.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

26.5.7 Standardization, Inter−Operability

MBase is connected via the MathWeb software bus to a variety of systems, translating back and forth mathematical libraries in various formats. MBase documents can be accessed by other systems using the same syntax, like ActiveMath does, or having export/import facilities for OMDoc, like PVS has.

26.5.8 Web Access

An online demo is available at http://mbase.mathweb.org:8080/mbase/ (currently down).

26.6 Example of a Theory Exploration Session

Following the pattern syntax described at http://mbase.mathweb.org:8080/Pattern−Syntax.html, (currently unavailable) we have the following search examples:

• Searching for a symbol with the exact name "pi" is done by: *s(name: exact("pi")).* (In MBase, symbols have the label s.)

• Searching for a symbol with the substring "cut" in the name is done by: *s(name:substr("cut")).*

• Regular expressions are allowed, e.g. *s (name:regexp(".group$"))* looks for symbols that end with "group" but have more than 5 characters.

• Searching for the commutativity law *F(x, y) = F(y, x)* is done by *eq(a (F X Y) a (F Y X))*, which is a shorthand for the expanded form *a(s(name:regexp('^=$)|(^[Ee][Qq])')) [a (F [X Y]) a (F [Y X])]*.

• Searching for $f(a, a) = a$ is done by *eq(a(F A A) A)*.

• Searching for the associativity law $f(x, f(y, z)) = f(f(x, y), z)$ is done by *eq(a (F X a(F Y Z)) a(F a(F X Y) Z))*.

27 MIZAR

27.1 Short Description

Mizar aims at transforming mathematical ideas into journal articles that are readable both by humans and machines, warranting consistency of the definitions and correctness of proofs [RudnickiTrybulec01].The mathematical objects in **Mizar** are theorems, definitions and schemes (i.e. theorems with second−order variables). The main components of the system are the **Mizar** Verifier and the **Mizar** Mathematical Library (**MML**). **MML** is the largest library of formalized mathematics existing today.

27.2 Technical Information on the System

27.2.1 Name of the System and Website

MIZAR, http://www.mizar.org/.

27.2.2 Project Leaders and Group

The **Mizar** group is led by Andrzej Trybulec and has the following members: Grzegorz Bancerek, Czeslaw Bylinski, Yasushi Fuwa, Mariusz Giero, Adam Grabowski, Pauline N.Kawa−moto, Artur Kornilowicz, Jaroslaw Kotowicz, Roman Matuszewski, Robert Milewski, Yatsuka Nakamura, Adam Naumowicz, Krzysztof Retel, Piotr Rudnicki, Christoph Schwarzweller, Yasunari Shidama, Bartlomiej Skorulski, Josef Urban, Katsumi Wasaki, Freek Wiedijk and Mariusz Zynel.

27.2.3 Main Publications

G.Bancerek,and P.Rudnicki, Information Retrieval in MML. In A. Asperti, B. Buchberger and J. Davenport (eds.),
Proceedings of the Second International Conference on Mathematical Knowledge Manage−ment, Springer, 2003, 2594, 119−132.

Y. Nakamura et al., Mizar Lecture Notes (4−th Edition, Mizar Version 6.1.12), Shinshu University, Nagano, 2002.

P. Rudnicki and A.Trybulec, Mathematical Knowledge Management in Mizar. In B. Buch−berger. and O. Caprotti (eds.), Proceedings of MKM2001, 2001.

27.2.4 Implementation Language

Mizar is coded in Pascal using the Free Pascal compiler (compatible with the GNU Pascal Compiler and Delphi/Kylix).

27.2.5 System Availability and Prerequisites

The **Mizar** system is available for most of the existing operating systems (Win32, Linux, Solaris, FreeBSD, Darwin/Mac OS). Free disk space of at least 75 MB is needed for the **Mizar** processor, the public data base of the **Mizar** Mathematical Library (MML) and abstracts of the **Mizar** articles in the **MML**.

27.3 Algorithm Libraries

27.3.1 Numerical, Discrete, Algebraic, etc. Libraries

27.3.2 Reasoners

The only reasoning tool [WiedijkChecker] involved in **Mizar** is the *by* construct (representing a subproof in a proof of a theorem). It does not cover full first order provability, but a weaker variant that can be decided fast and is "quite good at reasoning from type informa−tion, applying equalities, deducing existential statements" [WiedijkMizar].

27.3.3 Graphical Tools and Interfaces

MizarMode [Urban06] was developed as an Emacs authoring environment for Mizar. It provides proof assistance functions by integrating the functionalities of MMLQuery, MoMM and the Mizar Proof Advisor (see the Tools for Retrieval subsection of this chapter).

Alcor [Cairns04] is a graphical user interface usable for Mizar. It provides a textual editor together with a search tool for keywords. The search can be performed by highlighting words or by entering text.The list of search results will contain the location (i.e. Mizar reference) and the type of the items found.To see the actual definition, the user clicks on the location, and the required item is displayed, as given in an Mizar abstract (not a full Mizar article).

27.4 User Language

27.4.1 Programming Language

27.4.2 Logic Language for the Formulation of Mathematical Knowledge

The logic language underlying Mizar is classical first−order logic with the added capability of forming second−order schemes. The inference system is the Jakowski system of natural deduction. Its library of formalized mathematical data, MML, is based on the Tarski−Grothend−ieck set theory.

27.4.3 Mathematical Syntax

The Mizar Language is a formal language derived from the mathematical vernacular [Bruijn87, WiedijkVernacular]. A file written according to the Mizar Syntax is called a Mizar article (.miz), and has to be accompanied by a vocabulary file (.voc) specifying the lexical elements (functions, predicates, etc.).

A Mizar article [WiedijkMizar] contains an *environ* header and a sequence of theorems and definitions. The header names the other Mizar articles used in the current one as well as some parsing details (vocabularies needed, notations used and constructors involved). The *reserve* statement reservers variables for a certain type. (e.g. *reserve a for Real;*).

A definition is of the form:

definition
let arguments;
assume preconditions;
func: pattern −>type means label:statement;
correctness proof;
end;

The keyword *pattern* represents the way in which the operation is written (e.g. *log(a,b)* or *x−y*). In the *statement*, the defined object is referred as *it*.

The correctness proof has to guarantee, given the *preconditions*, the existence and unique−

ness of the defined object.

In a definition, *synonyms* and *antonyms* can be given, and properties like commutativity or symmetry can be attributed.

A theorem is of the form:

theorem label:statement
proof
proof steps
end;

Among the *proof steps*, we find the following rules*: let* (for universal introduction e.g. *let var be type*), *take* (for existential introduction), *consider* (for existential elimination), *per cases* (for disjunction elimination). Some natural deduction rules have no Mizar counterpart and are handled by so-called diffuse steps using *by;* see the subsection on **Reasoners** of this chapter for more details. For more on syntax see [WiedijkMizar].

27.5 Mathematical Knowledge Bases

27.5.1 Available Theories and Knowledge Bases

As mentioned before, the knowledge base of Mizar is called the Mizar Mathematical Library (MML). It is the largest library of formalized mathematics available in the world at this time. It aims at the reconstruction of the core of mathematics. Built on the axioms of Tarski−Grotend−ieck set theory, its current version (as of November 2007) includes 942 articles, written by 182 authors, comprising 42694 theorems, 7957 definitions, 728 schemes, 6942 registrations, 5879 symbols, and 1903 keywords.

Several well−known theorems that were formalized in Mizar are: Alexander's Lemma, the Banach Fixed Point Theorem for compact spaces, the Brouwer Fixed Point Theorem, the Birkhoff Variety Theorem for many-sorted algebras, Fermat's Little Theorem, the Fundamental Theorem of Algebra, the Fundamental Theorem of Arithmetic, the Goedel Completeness Theorem, the Hahn−Banach Theorem for complex and real spaces, the Jordan Curve Theorem for special polygons, the Reflection Theorem of set theory.

The biggest formalization project is the (still ongoing) formalization of the textbook "A Compendium of Continuous Lattices" [GierzEtAl80].

27.5.2 Tools for Retrieval in Mathematical Knowledge Bases

MMLQuery [BancerekRudnicki03] is a fast *query language* for MML. A set of indices (which is their version of metadata) is extracted from the library, and the user can retrieve a theorem by combining various filters on these indices. Some examples are:

- Getting the names and links to all articles containing NAT in their identifier:
  list of articles | grep−i NAT
- Obtaining all theorems referring to the mode *GROUP_2:mode 1*:
  list of th from (GROUP_2:mode 1 article);

- Items which are neither notations nor registrations:
  list of item | [not notat and not reg];
- Finding all items refering to all constructors from the theorem labelled *FUNCT_1:70*:
  at least(FUNCT_1:th 70 ref)*

The Mizar Proof Advisor [Urban05a, Urban06] accepts as input a complete formula and outputs a list of theorems that could help in proving the input formula, sorted by their expected relevance. It is based on the machine learning toolkit SNoW [CarlsonEtAl99], which imple−ments methods (like neural nets, Markov models, bayesian nets, etc) used for processing natural language.The proof advisor extracts features out of each Mizar formula (for now only the signature of the formula) and guesses the relation between these features and the theorems / definitions used in the proof of the formula. Based on these guesses, the system returns theo−rems and definitions that could be relevant for the user input. The best results were obtained using naive bayesian nets.

Most of Mizar Matches (MoMM) is another search tool for mathematical databases opti−mized for Mizar [Urban05]. Based on Stephan Schulz's prover E, its main goal is to eliminate redundant formulae. The tool translates parts of the MML into its own clausal−like format and eliminates the ones that can be subsumed by the others, using a kind of typed subsumption.

27.5.3 Tools for Inventing Mathematical Knowledge

27.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The Mizar verifier is a non−programmable and non−tactical verifier [Urban06]. It operates like a compiler, in the traditional "write−compile−correct the mistakes" style. It takes a Mizar article as input (see the subsection on mathematical syntax of the language, above) and checks the file for logical errors [BancerekRudnicki03], using a restricted set of natural deduction rules.

27.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

27.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

27.5.7 Standardization, Inter−Operability

MML articles can be exported to Prolog−based formats for theorem proving via the Mizar Problems for Theorem Proving (MPTP) [Urban2005a]. Importing facilities for parts of Mizar are provided by ISAR and HOL.

27.5.8 Web Access

MMLQuery can be used interactively on the web at http://mmlquery.mizar.org/mmlquery/three.html (as of 10.12.07). The Journal of Formalized Mathematics [JFM] gathers the articles accepted in MML. It has been published since 1989 and is now available on the web.

27.6 Example of a Theory Exploration Session

An example of formalizing a proof in Mizar is the proof that $\sqrt{2}$ is irrational.
The proof proceeds as follows [WiedijkMizar]:

The statement of the problem is:

```
sqrt 2 is irrational
```

The definitions assumed:

the definition of sqrt

```
reserve a for real number;
definition let a;
    assume 0 <= a;
    func sqrt a -> real number means
    :: SQUARE_1:def 4
    0 <= it & it^2 = a;
    end;
```

the definition of irrational

```
reserve x for set,
    m,n for Integer;
definition
    func RAT means
    :: RAT_1:def 1
    x in it iff ex m,n st x = m/n;
    end;
definition let r be number;
attr r is rational means
    :: RAT_1:def 2
    r in RAT;
    end;
reserve x for real number;
notation let x;
    antonym x is irrational for x is rational;
    end;
```

The proof proceeds as follows:

```
environ
    vocabulary SQUARE_1,IRRAT_1,ARYTM_3, RAT_1, INT_1;
    constructors NAT_1,PREPOWER, PEPIN, MEMBERD;
    notations XCLMPLX_0,
XREAL_0,INT_1,NAT_1,RAT_1,SQUARE_1,IRRAT_1;
    registrations XREAL_0,INT_1,MEMBERED;
    notations
XCMPLX_0,XREAL_0,INT_1,NAT_1,RAT_1,SQUARE_1,IRRAT_1;
    theorems
INT_1,SQUARE_1,REAL_2,INT_2,XCMPLX_1,NAT_1,RAT_1,NEWTON;
    requirements ARITHM,REAL,NUMERALS,SUBSET;
begin
    theorem
    sqrt 2 is irrational
proof
    assume sqrt 2 is rational;
    then consider i being Integer, n being Nat such that
    W1: n<>0 and
    W2: sqrt 2=i/n and
    W3: for i1 being Integer, n1 being Nat st n1<>0 & sqrt
2=i1/n1 holds n<=n1
        by RAT_1:25;
    A5: i=sqrt 2*n by W1,XCMPLX_1:88,W2;
    C: sqrt 2>=0 & n>0 by W1,NAT_1:19,SQUARE_1:93;
        then i>=0 by A5,REAL_2:121;
               then reconsider m = i as Nat by INT_1:16;
    A6: m*m = n*n*(sqrt 2*sqrt 2) by A5
        .= n*n*(sqrt 2)^2 by SQUARE_1:def 3
        .= 2*(n*n) by SQUARE_1:def 4;
        then 2 divides m*m by NAT_1:def 3;
        then 2 divides m by INT_2:44,NEWTON:98;
        then consider m1 being Nat such that
          W4: m=2*m1 by NAT_1:def 3;
       m1*m1*2*2 = m1*(m1*2)*2
       .= 2*(n*n) by W4,A6,XCMPLX_1:4;
       then 2*(m1*m1) = n*n by XCMPLX_1:5;
       then 2 divides n*n by NAT_1:def 3;
       then 2 divides n by INT_2:44,NEWTON:98;
       then consider n1 being Nat such that
    W5: n=2*n1 by NAT_1:def 3;
    A10: m1/n1 = sqrt 2 by W4,W5,XCMPLX_1:92,W2;
    A11: n1>0 by W5,C,REAL_2:123;
         then 2*n1>1*n1 by REAL_2:199;
    hence contradiction by A10,W5,A11,W3;
    end;
```

The Mizar Verifier takes this article, compiles it and returns the mistakes. The user improves the code, and runs the verifier again.

28 Nuprl

28.1 Short Description

The Nuprl system is designed to give assistance with creation of mathematical theories (proofs, formulae, definitions, ...). It underlines the computational aspects of terms, assertions and proofs. In a broader sense, it is a system for implementing mathematics [NuPRLBook]. The last version of the system (Nuprl 5) is subsumed by − and included into − the FDL project (see the respective chapter of this document). Most of the system information below is extracted from the version 4.xx documentation which is still valid for version 5.

28.2 Technical Information on the System

28.2.1 Name of the System and Website

NuPRL or Nuprl.

PRL is an acronym for "Proof/Program Refinement Logic". Several "PRL" systems, equivalent in some sense, were implemented, NuPRL (also read "new pearl") is the version "nu" of one series of PRL systems.

http://www.cs.cornell.edu/info/projects/nuprl/index.html

28.2.2 Project Leaders and Group

The leader of the project is Robert L.Constable (Dean of Faculty of Computing and Information Science, Computer Science Department, Cornell University).

The group of members working on NuPRL consists of (including former members) Eli Barzilay, David Basin, Ralph Benzinger, Jim Caldwell, Tat−Hung Chan, Wilfred Chen, W. Rance Cleaveland, Karl Crary, Rich Eaton, Timothy G. Griffin, Ozan Hafizogullari, Mark Hayden, Jason Hickey, Lori Lorigo, Amanda Holland−Minkley, Paul Jackson, Todd B. Knoblock, Alexey Kopylov, Nax P. Mendler, Wojciech Moczydlowski, Evan Moran, Rod Moten, Chetan Murthy, Pavel Naumov, Alexey Nogin, James T. Sasaki, Scott F. Smith, Melissa Totman, Judith Underwood.

28.2.3 Main Publications

The PRL Group. Implementing Mathematics with the Nuprl Proof Development System. Computer Science Department, Cornell University. October 1995. Available at: http://www.cs.cornell.edu/info/projects/nuprl/book/doc.html (last checked on 16.11.2007).

Jackson, Paul B. "Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra," Ph.D. Thesis, Cornell University, TR95−1905. 1995.

Christoph Kreitz. The Nuprl Proof Development System, Version 5. Reference Manual and User's Guide. (2002) Available on−line at:

http://www.cs.cornell.edu/info/projects/nuprl/html/nuprl5docs.html (last checked on 16.11.2007).

28.2.4 Implementation Language

"The Nuprl implementation consists of more than 100K lines of Lisp and tactic code implemented in ML. Parts of the system consist of legacy codes going as far back as the late 1970's (Edinburgh LCF)" [CaldwellCowles02].

28.2.5 System Availability and Prerequisites

The system can be downloaded freely from the project's website (via ftp). By compiling it, it can be used under most operating systems. For this, a licensed lisp compiler will be needed (CMUCL, Allegro, etc). To run the installation script, perl 5 is needed. Additionally, but optional, the Emacs editor is also needed.

28.3 Algorithm Libraries

28.3.1 Numerical/Algebraic/Symbolic/etc. Libraries

28.3.2 Reasoners

Proofs in Nuprl are done interactively. The user states a goal, and then by specifying refinement tactics − by name and possible parameters −, the goal is decomposed in a top−down manner. The number of rules is very large, so instead of remembering each rule's name, the system provides some generic names. The proof editor − the environment within the users develop their proofs − will infer the specifically needed rule from the local context. Some refinement tactics may be very complicated.

With version 5, users can invoke the MetaPRL refiner, and there's ongoing work to connect the system with HOL [GordonMelham93], Mathematica [Wolfram], Isabelle [Paulson07a].

Nuprl can extract programs from correct proofs.

28.3.3 Graphical Tools and Interfaces

The system comes with its own interface. For a description, see 'Example of a Theory Exploration Session' subsection below for a description.

28.4 User Language

28.4.1 Programming Language

The system has its own metalanguage, implemented in ML. User can write refinement rules, tactics, programs, etc. using this language. Additionally, to interact with the system, there is also a command language that initiates editing of proofs, tactics, etc.

28.4.2 Logic Language for the Formulation of Mathematical Knowledge

The logical language of Nuprl (also called the object language) is, in principle, based on Martin−Löf's type theory. The types available in the system can be grouped into five coarse−grained categories: familiar types and type constructors (like in the typed programming lan−guages ML or Pascal); dependent functions and dependent product constructors; quotient and set types; propositions as types; recursive types and partial functions.

Users can define various logics and then use them for stating theorems.

28.4.3 Mathematical Syntax

The syntax of NuPRL's ML language. See [Kreitz02, Appendix B] for details. The system also allows to define ones own syntax.

28.5 Mathematical Knowledge Bases

28.5.1 Available Theories and Knowledge Bases

The Nuprl knowledge base uses a transactional model for entering and modifying objects (definitions, theorems, etc.) following similar protocols as database systems do. Objects are not deleted/overwritten, rather a version control mechanism is used [NuPRLBook]. The order of the objects in the database is important. Referred objects must occur earlier in the library than the objects referring them.

The kinds of objects that can be stored in the library are four: DEF (define new notations), THM (proofs in tree form), EVAL (list of bindings − "let id = term ;;"), ML (ML programs, tactics).

Each object in the library has a status associated to it:

- raw: the object has been changed, but it is not yet checked;
- bad: the object has been checked and errors has been found;
- incomplete (only for proofs): proof contains no errors, but it is not finished;
- complete: object is correct and complete. [NuPRLBook].

The knowledge base supports dependency tracking in order to check theorem validity w.r.t. specific sets of rules, axioms, etc. [NuPRLBook].

The Nuprl library consists of a collection of independent theories (books). We list them below:

Nuprl Basics (By Stuart Allen): Explains the basic concepts and methods used for mathematical expression in Nuprl.

Standard Resources: A library of "standard" Nuprl objects (arrays, rationals, binary relations, booleans, integers, etc.)

Elementary Number Theory: Elementary divisibility theory over the integers, Gcd function and relation, Chinese remainder theorem.

Lists: Concepts and facts about lists.

Finite automata (Constable et. al): A constructive formalization of part of Aho, Hopcroft & Ulmann's book.

Chain Replication Protocol (Mark Bickford)

Event Systems (Mark Bickford)

Graph Theory (Mark Bickford)

Hybrid Protocols (Mark Bickford)

Discrete Mathematics (Stuart Allen)

Fundamental Theorem of Arithmetic (Stuart Allen)

Iterated Binary Operations (Stuart Allen)

Russel's Paradox (Stuart Allen)

Towers of Hanoi (Stuart Allen)

Classical Propositional Logic (J. Caldwell)

General Automata Theory (Mark Bickford)

The Zeno Paradox (Pavel Naumov)

Simple Imperative Programming (Paul Naumov)

Turing Machine Basics (P. Naumov)

Bar−Type Rules (Karl Cray)

Constructive General Algebra (P. Jackson)

Permutations (P. Jackson)

Finite Multi−sets (P. Jackson)

Constructive Factorization Theory (P. Jackson)

28.5.2 Tools for Retrieval in Mathematical Knowledge Bases

28.5.3 Tools for Inventing Mathematical Knowledge

28.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

28.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

The Nuprl version 5 is subsumed by the FDL project which has some tools for structuring and re−structuring of mathematical content. See the chapter on FDL, in this document.

28.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

Using the ML language, users of Nuprl can implement new proof tactics, which can be checked by the system. (In fact, when tactics are stored into the Nuprl library as ML objects − see 'Available Theories and Knowledge Bases' above − they must be checked correct by the system.) The system also supports reflection (see [Barzilay06]).

28.5.7 Standardization, Inter−Operability

The group has done some experiments with integrating PVS [PVSWeb] libraries into Nuprl [KleinbergEtAl03]. The Nuprl library is also included in the Helm (see the chapter on Helm of this document) repository of browsable mathematical content.

Various translator between NuPRL and other systems have been implemented. See, for example, [NaumovEtAl01]. Another such example is the interaction between NuPrl and the Weyl computer algebra system, described in [Jackson94a].

With version 5, a new design of the system, Nuprl LPE (logical programming environment) is to replace the former monolithic architecture of the system. The central component is a knowledge base to which various components (proof engines, editors, evaluators, translators) can connect.

28.5.8 Web Access

The system has a tool that automatically converts formal, computer−generated mathematical texts into a set of HTML files. The tool preserves the original, non−linear text structure [Naumov98]. The libraries listed in the 'Available Theories and Knowledge Bases' subsection above are generated with this tool, and can be inspected via a web browser.

The Nuprl library is also available for browsing from Helm's website.

28.6 Example of a Theory Exploration Session

28.6.1 General Description

The Nuprl system has six major components [NuPRLBook]: a window manager, a proof editor, a text editor, a library module, a command module, and a function evaluator. Switching between windows can be done with (combinations of) mouse clicks and keyboard. The window manager is responsible of providing an interface for creating, inspecting, modifying objects like definitions, theorems, proofs and libraries on a terminal screen. We will detail some of these components below.

The windows used in the Nuprl system are: The Command Window, The Library Window, The Refinement Editor (or the Proof Editor), The Text Editor.

The "P>" prompt in the Command Window tells the user that the command processor waits for a command. There are four groups of commands: commands that control the library win−dow, commands that manipulate objects, commands that save work between sessions, and miscellaneous commands.

Examples of library commands:

```
P>   jump object
```

causes the library window to redisplay, centering on the specified entry (*object* is the name of an object in the library).

```
P>   move  objects place
```

objects are moved after the specified *place.*

Examples of object manipulation commands:

```
P>   archive  objects
```

Copies of each of the *objects* are made and saved as the old version. (remember that there are no ways to delete an object.)

```
P>   check  objects
```

Each object is checked. If this operation completes without errors, the *objects* are from now on available for use in other objects.

Examples of storing result commands:

```
P>   dump objects {to   filename}
```

Write a representation of the object from the current library to the given file.

```
P>    load place from file
```

Read in a file created with the ’dump’ command. The library entries are added at the given *place*. Certain rules about how to read in the content of the file are taken into account. See [NuPRLBook].

Among the miscellaneous commands we mention ’exit’ which terminates the current NuPRL session, and ’shell’ which creates an interactive subshell (only on Linux/Unix versions of NuPRL)

As mentioned earlier, each object in the library has a status associated to it:

− raw: the object has been changed, but it is not yet checked; these entries are marked with ’?’;

− bad: the object has been checked and errors has been found; these entries are marked with ’−’;

− incomplete (only for proofs): proof contains no errors, but it is not finished; these entries are marked with ’#’;

− complete: object is correct and complete; these entries are marked with ’*’. [NuPRLBook].

Nuprl texts are sequences of characters stored internally as recursive trees. They are also know as text trees or T−trees. The Nuprl text editor is a structure editor for these trees. The modifications to the objects edited with this editor are reflected immediately

For a tutorial on how to use the Nuprl text editor and the Nuprl proof editor see [Jackson94]. We show here only an example of how to create a library (theory) and a proving example from the predicate logic, both examples taken from [NuPRLBook].

28.6.2 A Proof Example

Nuprl proofs are stored in a Lisp−readable form consisting of: the name of the theorem, its status (complete, partial, or in error), the sequent term which is the goal of the proof, the tactic script used to generate the proof, the extract term of the proof, and a list of lemma references made in the proof [CaldwellCowles02].

We show the proof of the following statement as a sequence of windows seen by the user of the system.

$$\forall_{A:U_1}\ \forall_{P:A\to A\to U_1}\ \exists_{y:A}\ \forall_{x:A}\ P(x)(y) \Longrightarrow \forall_{x:A}\ \exists_{y:A}\ P(x)(y)$$

Proofs in Nuprl are represented as trees. Each node contains a sequent (composed of a numbered list of hypotheses and a goal − preceded by ’>>’) and a rule (if the node is not a leaf).

Some steps in the proof of the statement above have been skipped in this presentation. Note the occurrence of the symbols used for raw and complete objects, ’#’ and ’*’, mentioned

before. The ’top’ keyword shows which position in the proof tree is displayed in the proof editor.

Entering the goal:

```
,----------------------,-------------------------------------,
|EDIT THM t3           |EDIT main goal of t3                 |
|----------------------|-------------------------------------|
|?top                  |>>all A:U1.all P:A->A->U1.           |
|<main proof goal>     |       some y:A.all x:A.P(x)(y)=>    |
|                      |       all x:A.some y:A.P(x)(y)      |
|                      '-------------------------------------'
|                                                            |
'------------------------------------------------------------'
```

After applying the ’intro’ refinement rule, we have two goals:

```
,----------------------------------------------------------,
|EDIT THM t3                                               |
|----------------------------------------------------------|
|# top 1 1                                                 |
|1. A:(U1)                                                 |
|2. P:(A->A->U1)                                           |
|>>some y:A.all x:A.P(x)(y)=>all x:A.some y:A.P(x)(y)      |
|                                                          |
|BY intro at U1                                            |
|                                                          |
|1# 1. A:(U1)                                              |
|2. P:(A->A->U1)                                           |
|3. some y:A.all x:A.P(x)(y)                               |
|   >>all x:A.some y:A.P(x)(y)                             |
|                                                          |
|2# 1. A:(U1)                                              |
|2. P:(A->A->U1)                                           |
|   >>some y:A.all x:A.P(x)(y) in U1                       |
'----------------------------------------------------------'
```

Here, an elimination step on the third hypothesis is tried:

```
,-------------------------------------,-----------------,
|EDIT THM t3                          |EDIT rule of t3  |
|-------------------------------------|-----------------|
|# top 1 1 1 1                        |elim 3           |
|1. A:(U1)                            |                 |
|2. P:(A->A->U1)                      |                 |
|3. some y:A.all x:A.P(x)(y)          |                 |
|4. x:(A)                             |                 |
|>>(some y:A.P(x)(y))                 |                 |
|                                     |                 |
|BY<refinement rule>                  '-----------------|
|                                                       |
'-------------------------------------------------------'
```

The elimination step requires two new variables (y_0 and h):

```
,-----------------------------------------------,
|EDIT THM t3                                    |
|-----------------------------------------------|
|# top 1 1 1 1                                  |
|1. A:(U1)                                      |
|2. P:(A->A->U1)                                |
|3. some y:A.all x:A.P(x)(y)                    |
|4. x:(A)                                       |
|>>(some y:A.P(x)(y))                           |
|                                               |
|BY elim 3 new y0,h                             |
|                                               |
|1# 1. A:(U1)                                   |
|2. P:(A->A->U1)                                |
|3. some y:A.all x:A.P(x)(y)                    |
|4. x:(A)                                       |
|5. y0:(A)                                      |
|6. h:(all x:A.P(x)(y0))                        |
|   >>(some y:A.P(x)(y))                        |
'-----------------------------------------------'

,-----------------------------------------------,
|EDIT THM t3                                    |
|-----------------------------------------------|
|# top 1 1 1 1 1                                |
|1. A:(U1)                                      |
|2. P:(A->A->U1)                                |
|3. some y:A.all x:A.P(x)(y)                    |
|4. x:(A)                                       |
|5. y0:(A)                                      |
|6. h:(all x:A.P(x)(y0))                        |
|>>(some y:A.P(x)(y))                           |
|                                               |
|BY intro y0                                    |
|                                               |
|1*1. A:(U1)                                    |
|2. P:(A->A->U1)                                |
|3. some y:A.all x:A.P(x)(y)                    |
|4. x:(A)                                       |
|5. y0:(A)                                      |
|6. h:(all x:A.P(x)(y0))                        |
|   >>y0 in (A)                                 |
|                                               |
|2# 1. A:(U1)                                   |
|2. P:(A->A->U1)                                |
|3. some y:A.all x:A.P(x)(y)                    |
|4. x:(A)                                       |
|5. y0:(A)                                      |
|6. h:(all x:A.P(x)(y0))                        |
|   >>(P(x)(y0))                                |
'-----------------------------------------------'
```

The first generated subgoal, above, is proved immediately (identical with the assumption 5). We continue with the second subgoal:

```
,-------------------------------------------------,
|EDIT THM t3                                      |
|-------------------------------------------------|
|*top 1 1 1 1 1 2                                 |
|1. A:(U1)                                        |
|2. P:(A->A->U1)                                  |
|3. some y:A.all x:A.P(x)(y)                      |
|4. x:(A)                                         |
|5. y0:(A)                                        |
|6. h:(all x:A.P(x)(y0))                          |
|>>(P(x)(y0))                                     |
|                                                 |
|BY elim h on x                                   |
|                                                 |
|1*1. A:(U1)                                      |
|2. P:(A->A->U1)                                  |
|3. some y:A.all x:A.P(x)(y)                      |
|4. x:(A)                                         |
|5. y0:(A)                                        |
|6. h:(all x:A.P(x)(y0))                          |
|   >>x in (A)                                    |
|                                                 |
|2*1. A:(U1)                                      |
|2. P:(A->A->U1)                                  |
|3. some y:A.all x:A.P(x)(y)                      |
|4. x:(A)                                         |
|5. y0:(A)                                        |
|6. h:(all x:A.P(x)(y0))                          |
|7. (P(x)(y0))                                    |
|   >>(P(x)(y0))                                  |
'-------------------------------------------------'
```

We have, again, obtained two subgoals which are identical with hypothesis in the local context (in each case), and the proof is done.

28.6.3 A Library Example

Here is a library with some propositional logic rules. The '*' character means that the library entries are completed. What we see below is the content of the Library Window.

```
,-------------------------------------------------------------
-,
| Library
|
|-------------------------------------------------------------
-|
|*C class_prop_begin    *********** CLASS PROP ***************
|
|*t dneg_elim           ∀ A:ℙ{i}. ¬¬ A ⟹ A
|
```

```
|*t imp_elim            ∀ A:ℙ{i}. ∀ B:ℙ{i}. (A ⟹ B) ⟹ ¬ A ∨ B
|
|*t neg_imp_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ⟹ B) ⟹ A ∧ ¬
B |
|*t neg_or_elim         ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∨ B) ) ⟹ A ∧ ¬
B|
|*t neg_and_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∧ B) ) ⟹ A ∨ ¬
B|
|*t pierce             ∀ A:ℙ{i}. ∀ B:ℙ{i}. ((P ⟹ Q) ⟹ P) ⟹
P |
|*C class_prop_end     ************************************
|
'-------------------------------------------------------------
-'
```

$\mathbb{P}_i$ reads as 'propositional universe *i*' and is a type for propositions. The subscript *i* corresponds to the type theoretic tradition established by Bertrand Russell. There is an infinite, cumulative series of types of propositions: $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{P}_3$, ...; $\mathbb{P}_i$ contains all the levels from 1 up to *i−1*, the subscripts are called *levels.*

To create an own theory, we have to scroll at the bottom of the library and add delimiters for the new theory:

```
bottom ()
add_theory _delimiters "user"
```

Now the Library Window looks like:

```
,-------------------------------------------------------------
-,
| Library
|
|-------------------------------------------------------------
-|
|*C class_prop_begin    *********** CLASS PROP ***************
|
|*t dneg_elim           ∀ A:ℙ{i}. ¬¬ A ⟹ A
|
|*t imp_elim            ∀ A:ℙ{i}. ∀ B:ℙ{i}. (A ⟹ B) ⟹ ¬ A ∨ B
|
|*t neg_imp_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ⟹ B) ⟹ A ∧ ¬
B |
|*t neg_or_elim         ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∨ B) ) ⟹ A ∧ ¬
B|
|*t neg_and_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∧ B) ) ⟹ A ∨ ¬
B|
|*t pierce             ∀ A:ℙ{i}. ∀ B:ℙ{i}. ((P ⟹ Q) ⟹ P) ⟹
P |
|*C class_prop_end     ************************************
|
|*C user begin          ************** USER ******************
|
|*C user end            ************************************
|
```

```
,------------------------------------------------------------
-'
```

Creating definitions and/or theorems is done in two stages. First, we must create a slot in the library for the new definition/theorem. The command for this is 'create' with some parameters: the type of entity we want to add, the place in the library, the name of the entity.

```
create "not_over_and" thm "user_end"
```

The command is to be entered into the command window. The Library Window is modified as follows

```
,------------------------------------------------------------
-,
| Library
|
|------------------------------------------------------------
-|
|*C class_prop_begin    ********** CLASS PROP ***************
|
|*t dneg_elim           ∀ A:ℙ{i}. ¬¬ A ⟹ A
|
|*t imp_elim            ∀ A:ℙ{i}. ∀ B:ℙ{i}. (A ⟹ B) ⟹ ¬ A ∨ B
|
|*t neg_imp_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ⟹ B) ⟹ A ∧ ¬
B |
|*t neg_or_elim         ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∨ B) ⟹ ¬ A ∧ ¬
B|
|*t neg_and_elim       ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∧ B) ⟹ ¬ A ∨ ¬
B|
|*t pierce             ∀ A:ℙ{i}. ∀ B:ℙ{i}. ((P ⟹ Q) ⟹ P) ⟹
P |
|*C class_prop_end     ************************************
|
|*C user begin         ************** USER ******************
|
|?t not_over_and       [term]
|
|*C user end           ************************************
|
'------------------------------------------------------------
-'
```

After having created the slot for the new theorem, we have to invoke the proof and text editors for it. This is done with the 'view' command:

```
view "not_over_and"
```

In addition to the Library (and the Command) window(s), the user sees now a new window, the proof editor window:

```
,-------------------------------------,
| EDIT THM not_over_and               |
|-------------------------------------|
```

```
|?  top                                |
|<main proof goal>                     |
|                                      |
’______________________________________’
```

A click on the ’<main proof goal>’ will open the Nuprl text editor, where the theorem can be typed in. We don’t show this window, here. The theorem we type in is: ∀A:ℙ{i}.∀B:ℙ{i}. ¬(A∧B) ⟺ ¬A∨¬B , and the Library window looks like:

```
,-------------------------------------------------------------------
-,
| Library
|
|-------------------------------------------------------------------
-|
|*C class_prop_begin    ********** CLASS PROP ***************
|
|*t dneg_elim           ∀ A:ℙ{i}. ¬¬ A ⇒ A
|
|*t imp_elim            ∀ A:ℙ{i}. ∀ B:ℙ{i}. (A ⇒ B) ⇒ ¬ A ∨ B
|
|*t neg_imp_elim        ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ⇒ B) ⇒ A ∧ ¬
B |
|*t neg_or_elim         ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∨ B) ⇒ ¬ A ∧ ¬
B|
|*t neg_and_elim       ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∧ B) ⇒ ¬ A ∨ ¬
B|
|*t pierce             ∀ A:ℙ{i}. ∀ B:ℙ{i}. ((P ⇒ Q) ⇒ P) ⇒
P |
|*C class_prop_end     ************************************
|
|*C user begin         ************** USER ******************
|
|#t not_over_and       ∀ A:ℙ{i}. ∀ B:ℙ{i}. ¬(A ∧ B) ⇔ ¬ A ∨ ¬
B|
|*C user end           ************************************
|
’-------------------------------------------------------------------
-’
```

Notice that the mark before the newly introduced theorem is now ’#’ which means that the object is syntactically correct, but needs checking (i.e. a proof). This is done as shown in the subsection ’A Proof Example’ above.

Saving and loading the theory is done with the commands in the Command Window described above. (’dump’ , ’load’).

29 OMDoc

29.1 Short Description

The Open Mathematical Documents (OMDOC) format "is a content markup scheme for (collections of) mathematical documents including articles, textbooks, interactive books, and courses. OMDoc also serves as the content language for agent communication of mathematical services on a mathematical software bus" [Kohlhase06]. It is build as an extension of OpenMath [OpenMath] that treats whole mathematical theories. It is based on a three−level hierarchy consisting of:

- mathematical formulae (coded in content MathML [MathML] or OpenMath),
- various statements (like definitions, assertions and examples) and
- theories (related by morphisms for expressing e.g. theory inclusion/interpretation).

29.2 Technical Information on the System

29.2.1 Name of the System and Website

Open Mathematical Documents (OMDoc), http://www.omdoc.org/ .

29.2.2 Project Leaders and Group

OMDoc is led by Michael Kohlhase (Jacobs University Bremen), and has the following members: Paul Libbrecht, Andreas Franke, George Goguadze, Olga Caprotti, Alberto Gonzalez Palomo, Christoph Lange, and Florian Rabe.

29.2.3 Main Publications

M. Kohlhase, OMDoc. An Open Markup Format for Mathematical Documents (version 1.2), LNAI 4180, August 2006, Springer−Verlag GmbH. http://www.omdoc.org/pubs/omdoc1.2.pdf

M. Kohlhase, OMDoc:Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge Proceedings of "Artificial Intelligence and Symbolic Computation", Springer LNAI,2000. http://www.omdoc.org/pubs/aisc00.pdf

M. Kohlhase, OMDoc:An Infrastructure for OpenMath Content Dictionary Information Bulletin of the ACM Special Interest Group for Algorithmic Mathematics SIGSAM, 2000. http://www.omdoc.org/pubs/sigsam.pdf

29.2.4 System Availability and Prerequisites

OMDoc is a content markup language, and as such, is available free of charge.

29.3 Algorithm Libraries

29.4 User Language

29.4.1 Programming Language

29.4.2 Logic Language for the Formulation of Mathematical Knowledge

29.4.3 Mathematical Syntax

OMDoc is a markup format and data model for Open Mathematical Documents. It serves as semantics-oriented representation format and ontology language for mathematical knowledge.

The mathematical formulae are represented in OMDoc by OpenMath objects (the element *OMOBJ*). These have a tree-like representation containing applications (*OMA*), binding structures (*OMBIND* using *OMBVAR* to specify the bound variables), variables (*OMV*), and symbols (*OMS*).

The structure of the mathematical statements (*definition, axiom, example, theory, lemma)* is made explicit by a set of formal mathematical properties (*FMP)* and commented mathematical properties (*CMP).* E.g.

```
<definition for="#plus" type="recursive">
        <CMP>Addition is defined by recursion on the second argument </CMP>
        <FMP> X + 0 = 0</FMP>
        <FMP>X + s(Y ) = s(X + Y )</FMP>
</definition>
```

Also authorship, title can be specified in a <*metadata*> tag. For more details about the OMDoc syntax see [Kohlhase06].

29.5 Mathematical Knowledge Bases

29.5.1 Available Theories

Since OMDoc is a markup format it does not have its own libraries of mathematical theories. However, ActiveMath**,** MBase**,** Omega**,** SWiM and other systems (see chapter 26 in [Kohlhase06]) use OMDoc for the representation of structured mathematical knowledge.

29.5.2 Tools for Retrieval in Mathematical Knowledge Bases

MathWebSearch is an online search engine for mathematical expressions developed by the KWARK team at Bremen [KohlhaseSucan06]. It contains

• knowledge bases obtained by harvesting the internet for content representations of mathe−matical expressions (currently the OpenMath and the content MathML formats are supported),

• an indexing mechanism using substitution trees [Graf96] that creates index terms for all mathematical expressions (i.e.for each math element in MathML and more or less for each *apply* element in OpenMath) and

• a query language for mathematical expressions, realized as a generic extension mechanism for XML−based representation formats.

The repositories of mathematical knowledge that are harvested include the ActiveMath [MelisSiekmann04] repository, the CONNEXIONS corpus [CONNEXIONS] and the MBase system [KohlhaseFranke01].The mathematical expressions are interpreted by the indexing mechanism as first−order terms. The query language is obtained by adding new attributes and tags to the XML−based languages. For example, the attribute *mq:generic* is used for matching any subterm in the index, the attribute *mq:anyorder* for specifying an arbitrary order of the parameters of the current node

The user can input the desired search query in OpenMath or MathML format.In the former case, one may also take advantage of the WIRIS [WIRIS] OpenMath Editor (a user−friendly Java plug−in for editing mathematical expressions). The standard term retrieval algorithm for substitution trees [Graf94] is used as a search algorithm. The system can find an expression (occurring as a subexpression in any statement) up to α-conversion by adding the *mq:generic* attribute to every bound variable in the search query. Searches in informal text are not possible.

A related OMDoc tool also developed by the KWARK team is a semantic wiki (a wiki with Semantic Web Capabilities) allowing in the wiki pages also OMDoc expressions texts called SWiM [Lange07].

29.5.3 Tools for Inventing Mathematical Knowledge

29.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

29.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

29.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

29.5.7 Standardization, Inter−Operability

OMDoc is a XML−based markup format, and as such, it is platform-independent, and can be - and is - used successfully for communication between applications.

29.5.8 Web Access

OMDoc itself can not be accessed on the web. The OMDoc repositories from ActiveMath and MBase can be visualized on the web.

29.6 Example of a Theory Exploration Session

An example of an OMDoc document is the following fragment, taken from a file dealing with the natural numbers,

https://svn.omdoc.org/repos/omdoc/branches/omdoc−1.2/examples/deussen/nat.numbers.omdoc.

```
<?xml version="1.0" encoding="utf−8"?>
<!DOCTYPE omdoc PUBLIC "−//OMDoc//DTD OMDoc V1.2//EN"
"../../dtd/omdoc.dtd"[]>
 <omdoc xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns="http://www.mathweb.org/omdoc" xml:id="nat.numbers.omdoc">
      <metadata>
       <dc:title xml:lang="en">Natural Numbers</dc:title>
              <dc:creator role="aut">Barbara Schuett
Kerber</dc:creator>
               <dc:creator role="aut">Michael
Kohlhase</dc:creator>
       </metadata>
       <theory xml:id="nat.numbers">
               <symbol name="succ" role="object" scope="global">
                           <metadata>
                               <dc:subject
xml:lang="de">Nachfolgerfunktion</dc:subject>
                               <dc:subject>successor
function</dc:subject>
                           </metadata>
                   </symbol>
                   <symbol name="plus" role="object"
scope="global">
                              <metadata>
                               <dc:subject
xml:lang="de">Addition</dc:subject>

<dc:subject>addition</dc:subject>
                               </metadata>
                     </symbol>
                     …
                      <symbol name="nat" role="sort"
scope="global">
                                        <metadata>
                                            <dc:subject
xml:lang="de">Natuerliche Zahlen</dc:subject>
                                            <dc:subject>natural
numbers</dc:subject></metadata>
                       </symbol>
                        …
                       <symbol name="comp" role="object"
```

```
scope="global">
                        <metadata>
                            <dc:subject
xml:lang="de">Komposition von Funktionen</dc:subject>

<dc:subject>Composition of functions</dc:subject>
                                    </metadata>
                      </symbol>
                      <definition xml:id="comp.def" for="#comp"
type="implicit">
                        <FMP>
                                              <OMOBJ
xmlns="http://www.openmath.org/OpenMath">
                                                 <OMBIND>
                                                    <OMS cd="quant1"
name="forall"/>
                                                       <OMBVAR>
                                                         <OMATTR>
                                                           <OMATP>
                                                            <OMS
cd="simpletypes" name="type"/>
                                                                 <OMA>
                                                               <OMS
cd="simpletypes" name="funtype"/>
                                                               <OMV
name="B"/>

<OMV name="A"/>

</OMA>
                                                              </OMATP>
                                                             <OMV
name="G"/>
                                                          </OMATTR>
                                                             <OMATTR>
                                                                <OMATP>
                                                                 <OMS
cd="simpletypes" name="type"/>

<OMA>

<OMS cd="simpletypes" name="funtype"/>

<OMV name="C"/>

<OMV name="B"/>

</OMA>

</OMATP>
                                                                    <OMV
name="F"/>
                                                                </OMATTR>
                                                                  <OMATTR>
                                                                  <OMATP>

<OMS cd="simpletypes" name="type"/>
```

```
<OMV name="A"/>
                                                                </OMATP>
                                                                    <OMV
name="X"/>

</OMATTR>
                                                        </OMBVAR>
                                                           <OMA>
                                                                <OMS
cd="relation1" name="eq"/>
                                                                <OMA>
                                                                    <OMA>
                                                                        <OMS
cd="basic.algebra" name="comp"/>
                                                                        <OMV
name="F"/>
                                                                        <OMV
name="G"/>
                                                                    </OMA>
                                                                    <OMV
name="X"/>
                                                           </OMA>
                                                                <OMA>
                                                                    <OMV
name="F"/>
                                                                <OMA>
                                                                    <OMV
name="G"/>
                                                                       <OMV
name="X"/>
                                                                </OMA>
                                                        </OMA>
                                    </OMA>
                            </OMBIND>
                        </OMOBJ>
        </FMP>
</definition>
<symbol name="set-difference" role="object" scope="global">
    <metadata>
        <dc:subject xml:lang="de">einelementige
Menge</dc:subject>
                        <dc:subject>comp</dc:subject>
           </metadata>
 </symbol>
 <definition xml:id="set-difference.def" for="#set-difference"
type="implicit">
    <FMP>
        <OMOBJ xmlns="http://www.openmath.org/OpenMath">
                            <OMBIND>
                                                        <OMS cd="quant1"
name="forall"/>
                                                        <OMBVAR>
                                                         <OMATTR>
                                                                <OMATP>
                                                               <OMS
cd="simpletypes" name="type"/>
```

```
                                                        <OMA>
                                                              <OMS
cd="simpletypes" name="funtype"/>
                                                              <OMV
name="A"/>

<OMS cd="truthval" name="bool"/>
                                                         </OMA>
                                                                </OMATP>
                                                               <OMV
name="G"/>
                                                        </OMATTR>
                                                        <OMATTR>
                                                            <OMATP>
                                                                <OMS
cd="simpletypes" name="type"/>
                                                                <OMV
name="A"/>
                                                            </OMATP>
                                                            <OMV name="X"/>
                                                        </OMATTR>
                                                </OMBVAR>
                                                <OMA>
                                                    <OMS cd="logic1"
name="equivalent"/>
                                                    <OMA>
                                                        <OMS
cd="basic.algebra" name="set-difference"/>
                                                        <OMV name="G"/>
                                                        <OMV name="H"/>
                                                        <OMV name="X"/>
                                                    </OMA>
                                                    <OMA>
                                                       <OMS cd="logic1"
name="and"/>
                                                    <OMA>
                                                      <OMV name="G"/>
                                                     <OMV name="X"/>
                                                    </OMA>
                                                    <OMA>
                                                        <OMS cd="logic1"
name="not"/>
                                                    <OMA>
                                                        <OMV name="H"/>
                                                        <OMV name="X"/>
                                                    </OMA>
                                                </OMA>
                                            </OMA>
                                        </OMA>
                                    </OMBIND>
          </OMOBJ></FMP>
         </definition>
   </theory>
</omdoc>
```

Its corresponding fragment of XHTML document (taken from https://svn.omdoc.org/repos/omdoc/branches/omdoc−1.2/examples/deussen/nat.numbers.xhtml):

```
Natural Numbers

Barbara Schuett Kerber, Michael Kohlhase

Concept succ
Concept plus
…
Concept nat
…
Concept comp
Definition
forall G,F,X eq(comp(F,G)(X), F(G(X)))
Concept set−difference
Definition
forall G,X equivalent(set−difference (G,H,X), and (G(X),
not(H(X))))
```

30 Omega

30.1 Short Description

Ωmega is an interactive theorem prover and a mathematical assistant tool. It focuses on proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system containing a proof data structure (PDS) and several subsystems, e.g. a distrib−uted knowledge base (MBase), a user interface (LΩUI), an explanation module (P.rex), and a mathematical web service that connects Ωmega with external systems (MathWeb/MathServe). It is based on knowledge-based proof planning [Bundy91]: the proof of a theorem is planned at an abstract level and an outline of the proof is found. The outline (the abstract proof plan) is then expanded recursively and a proof within a logical calculus is constructed.

30.2 Technical Information on the System

30.2.1 Name of the System and Website

The Ωmega System, http://www.ags.uni−sb.de/~omega/software/omega/index.html

30.2.2 Project Leaders and Group

Leader: Prof. Jörg Siekmann. Group members: Serge Autexier, Christoph Benzmüller, Dominik Dietrich, Armin Fiedler, Andreas Franke, Helmut Horacek, Henri Lesourd, Marvin Schiller, Ewaryst Schulz, Frank Theiss and Marc Wagner.

30.2.3 Main Publications

Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. Computer Supported Mathematics with OMEGA, Journal of Applied Logic, special issue on Mathematics Assistance Systems 4(4), December, 2006

M. Kohlhase and H.M.Franke, MBase: Representing Knowledge and Content for the Integration of Mathematical Software Systems, Journal of Symbolic Computation, 2001, 32, 365−402.

A. Fiedler, P.Rex: An interactive proof explainer, In Automated Reasoning—1st International Joint Conference, Rajeev Goré, Alexander Leitsch, and Tobias Nipkow (eds.), IJCAR 2001, LNAI 2083, pp. 416−420, Siena, Italy, Springer Verlag, 2001.

30.2.4 Implementation Language

The user interface of Ωmega (LOUI) is implemented in the concurrent constraint programming language mOZart/Oz [mOZart]. P.rex is implemented in a goal-directed production system, Lisp based, ACT−R [AndersonLebiere98]. MBase is based on mOZart/Oz, java (JDBC), a relational data base management system (like Oracle), and on OMDoc.

The proof data structure (PDS), the interactive prover and the knowledge bases in Ωmega are implemented in Allegro Lisp.

30.2.5 System Availability and Prerequisites

ΩMEGA is distributed under the GNU General Public License. For running LΩUI mOZart is needed, and for running Ωmega, Allegro Common Lisp is needed. For using MBase, Java is needed.

30.3 Algorithm Libraries

Ωmega is linked over the MathWeb [FrankeKohlhase99] mathematical software bus to a variety of computer algebra systems, like Maple and CoCoA. As such, it has access to the algorithms implemented in those systems (see the respective chapters in this document).

30.3.1 Numerical, Discrete, Algebraic, Etc. Libraries

30.3.2 Reasoners

The basis for interactive theorem proving in Ωmega is made by the inference system together with tactics (sequences of rule applications). Ωmega's inference mechanism is an interactive theorem prover based on a higher−order natural deduction sorted version of Church's simply typed λ-calculus. To prove a conjecture, a proof plan for a proof is con−structed and then the proof planner uses it to guide the construction of the proof itself. (A proof plan is an outline or plan of a proof, containing methods and tactics.)

The tactics usually depend on a certain theory (e.g. tactics for polynomial multiplication), and, for avoiding an unsound logic calculus, every tactic has to be expandable to rules. Thus, a fully expanded proof (without open nodes) that is verified by the Ωmega proof checker will be correct, since the natural deduction calculus is correct.

The **P**roof Plan **D**ata **S**tructure (PDS) represents proofs and proof plans at various levels of granularity and abstraction. It is a directed acyclic graph, with open nodes (unjustifid proposi−tions that need to be proved) and closed nodes (propositions that are already proved).

During the proof development the PDS is modified either by the user of Ωmega, the proof planner MULTI or the suggestion mechanism Ω-ANTS (which can also call external systems) until a proof plan is found. After a proof plan is found, it is expanded by sub-methods and sub-tactics into lower levels of abstraction until a proof at the logical calculus level is found. Now, this proof (actually the PDS) can be checked by the proof-checker [Cheikhrouhou−Sorge00].

The proof plans are classified with respect to a taxonomy of mathematical theories in the mathematical knowledge base MBASE [KohlhaseFranke01]. There are different styles of proof development in Ωmega: tactical theorem proving, interactive island proof planning or fully automated proving (using the proof planner MULTI).

(1) The tactical theorem prover is a general purpose interactive theorem prover at the calcu−lus level. (It is called tactical since it builds on tactics—macro−steps embedding a sequence of inference steps). (See an example of a proof of "$\sqrt{2}$ is irrational" done using tacticals in the Example section of this chapter).

(2) Interactive island planning takes as input the proof plan. The user also provides the main subgoals, called islands, together with their assumptions. The details of the proof (down to the logic level) are filled in (ideally) automatically, by using the external systems interfaced to Ωmega. If this is not possible, additional user input is required, and the island approach will be applied once again. The proofs have now a level of abstraction similar to the one found in mathematical textbooks. (See an example of a proof of "$\sqrt{2}$ is irrational" done using interactive island planning in the Example section of this chapter).

(3) The proof planner MULTI [MeierMelis05, MelisEtAl08] uses a so-called blackboard architecture [EngelmoreMorgan88]. The modification and refinement of the proof plan is made there automatically. The knowledge sources (i.e. components) of MULTI solve a problem by putting the current solution state on the blackboard, where all other knowledge sources of MULTI can access it and develop it further. Using MULTI, an automatically planned and expanded proof of "$\sqrt{2}$ is irrational" was obtained [SiekmannEtAl03].

30.3.3 Graphical Tools and Interfaces

LΩUI [SiekmannEtAl99] is the graphical user interface of Ωmega. It combines a graphical display of information in a proof graph, a selective term browser with hypertext facilities, proof and proof plan presentation in natural language, and an editor for adding and maintaining the knowledge base. It is realized in an agent−based client-server architecture.

P.rex [Fiedler01] is an interactive, user-adaptive proof explanation system connected to Ωmega. It has its own specification language (TWEGA) for proofs and mathematical theories (containing e.g. axioms, definitions, and theorems along with proofs). A **dialog planner** (goal-directed production system) chooses a degree of abstraction for each proof step to be explained, based on assumptions about the users knowledge. It can also adapt by accepting at any time user interactions, entering clarification dialogs for updating its user model and for modifying the explanation correspondingly. The system chooses an explanation adapted to the knowledge of the user and reacts flexibly to the users interactions. The dialog plan is then passed through the **sentence planner** (that plans the internal structure of the sentences) to the **surface realizer**. The latter displays the sentences on the user interface, also allowing the user to enter remarks, requests and questions.

The development graph manager Maya [AutexierEtAl02] and the generic mediator between text editors and proof assistants PLATO [WagnerEtAl06] present also graphical interfaces in the Ωmega system.

30.4 User Language

30.4.1 Programming Language

The search for proofs is done at a level of abstractions defined by tactics and methods.

30.4.2 Logic Language for the Formulation of Mathematical Knowledge

The language of Ωmega, POST (**P**artial Functions **O**rder **S**orted **T**ype Theory), is a variant of simply-typed higher order λ-calculus with sorts.

30.4.3 Mathematical Syntax

Ωmega has its own custom−tailored syntax (Lisp based) that is exported to OMDoc docu−ments (which is the mathematical syntax of MBase).

30.5 Mathematical Knowledge Bases

30.5.1 Available Theories and Knowledge Bases

MBase (see the chapter on MBase in this document) is the main knowledge base available to Ωmega. It is a structured and distributed repository of mathematical knowledge. It is implemented as a mathematical service of the MathWeb mathematical software bus, which is in turn built on top of OMDoc. MathWeb connects a wide range of mathematical services (including interfaces to Ωmega, Bliksem, Otter, Mace, Maple, CoCoA). The theories covered in MBase are: Zermelo−Fraenkel Set Theory, Neumann−Bernays−Goedel Set Theory, Group Theory, Rings, Fields, Reals, Integers, Naturals, Polynomials, Calculus, Topology, etc.

30.5.2 Tools for Retrieval in Mathematical Knowledge Bases

Ωmega uses the retrieval facilities of MBase (for the knowledge imported from MBase).

The MathWeb [FrankeKohlhase99] software bus, searches—on the internet—a web service capable of solving a given mathematical problem. It does this, by classifying the mathematical services to which it has access by the description of the problem they solve (i.e. inputs, output, pre/postconditions). Since 2003 MathWeb was no longer maintained, and MathServe [Zimmer−Autexier06] emerged as its successor.

30.5.3 Tools for Inventing Mathematical Knowledge

30.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

There are three different styles of proof development in Ωmega: tactical theorem proving, interactive island proof planning or fully automated planning (using the proof planner MULTI). For more details please see the section on **Reasoners** of this chapter, as well as the referenced papers.

30.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

30.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

30.5.7 Standardization, Inter−Operability

Ωmega is connected via the MathWeb software bus to a variety of systems, translating back and forth mathematical libraries in various formats. Its syntax is translated into OMDoc, the syntax of MBase. OMDoc documents can be accessed by other systems using the same syntax, like ActiveMath, or having export/import facilities for OMDoc like PVS.

30.5.8 Web Access

After the user installs locally mOZart, the graphical interface LΩUI can be launched locally, and it will connect over the internet to an Ωmega server, enabling a restricted usage of Ωmega.

30.6 Example of a Theory Exploration Session

Following [Wiedijk06], we consider the formalization of the "$\sqrt{2}$ is irrational" problem. The statement of the problem in Ωmega is:

```
(th~defproblem sqrt2-not-rat
    (in real)
    (conclusion (not (rat (sqrt 2))))
    (help "√2 is not a rational number"))
```

The definitions of rational (rat) and sqrt are:

```
(th~defdef rat
    (in rational)
    (definition
        (lam (x num)
        (exists-sort (lam (y num)
        (exists-sort (lam (z num)
            (= x (frac y z))) int\0)) int)))
(sort)
(help "The set of rationals, constructed as fractions a/b of
integers."))

(th~defdef sqrt
    (in real)
    (definition
        (lam (x num) (that (lam (y num) (= (power y 2) x))))))
(help "Definition of square root."))
```

For proving the theorem, some definitions and theorems are needed, like the definition of even, the theorem that 2 is a common divisor of even numbers, etc. :

```
(th~deftheorem even-on-integers
    (in real)
    (conclusion
        (forall-sort (lam (x num)
            (equiv (evenp x)
                (exists-sort (lam (y num)
                      (= x (times 2 y))) int))) int))
    (help "An integer x is even, iff an integer y exists so
that x=2y."))

(th~deftheorem square-even
    (in real)
    (conclusion
        (forall-sort (lam (x num)
            (equiv (even (power x 2)) (evenp x)))
            int))
    (help "x is even, if x^2 is even."))

(th~deftheorem even-common-divisor
    (in real)
    (conclusion
        (forall-sort (lam (x num)
            (forall-sort (lam (y num)
                (implies (and (evenp x) (evenp y))
                    (common-divisor x y 2)))
            int))
        int))
    (help "If x and y are even, then they have '2' as a common
divisor."))
```

The theorem can be proved in two styles: by a proof script [Wiedijk06] using tacticals, or, interactively by using an Emacs session. We present here the proof script proof style.

```
DECLARATION DECLARE ((CONSTANTS (M NUM) (N NUM) (K NUM)))
RULES NOTI default default
MBASE IMPORT-ASS (RAT-CRITERION)
TACTICS FORALLE-SORT default default ((SQRT 2)) default
TACTICS EXISTSE-SORT default default (N) default
TACTICS ANDE default default default
TACTICS EXISTSE-SORT (L7) default (M) default
TACTICS ANDE* (L8) (NIL)
OMEGA-BASIC LEMMA default ((= (POWER M 2) (TIMES 2 (POWER N
2))))
TACTICS BY-COMPUTATION (L13) ((L11))
OMEGA-BASIC LEMMA (L9) ((EVENP (POWER M 2)))
RULES DEFN-CONTRACT default default default
OMEGA-BASIC LEMMA (L9) ((INT (POWER N 2)))
TACTICS WELLSORTED default default
TACTICS EXISTSI-SORT (L15) ((POWER N 2)) (L13) (L16) default
MBASE IMPORT-ASS (SQUARE-EVEN)
TACTICS ASSERT ((EVENP M)) ((SQUARE-EVEN L10 L14)) (NIL)
```

```
RULES DEFN-EXPAND (L17) default default
TACTICS EXISTSE-SORT default default (K) default
TACTICS ANDE (L19) default default
OMEGA-BASIC LEMMA default ((= (POWER N 2) (TIMES 2 (POWER K
2))))
TACTICS BY-COMPUTATION (L23) ((L13 L22))
OMEGA-BASIC LEMMA default ((EVENP (POWER N 2)))
RULES DEFN-CONTRACT default default default
OMEGA-BASIC LEMMA (L20) ((INT (POWER K 2)))
TACTICS WELLSORTED (L26) ((L21))
TACTICS EXISTSI-SORT default ((POWER K 2)) (L23) default default
TACTICS ASSERT ((EVENP N)) ((SQUARE-EVEN L6 L24)) (NIL)
MBASE IMPORT-ASS (EVEN-COMMON-DIVISOR)
OMEGA-BASIC LEMMA (L20) ((INT 2))
TACTICS WELLSORTED (L28) (NIL)
TACTICS ASSERT (FALSE) ((EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27
L28)) (NIL)
RULES WEAKEN default default
```

31 OpenMath

31.1 Short Description

OpenMath is an extensible standard for representing mathematical knowledge together with their semantics. There can be different encodings of objects following the OpenMath standard. The most used one is the XML encoding. It needs to be said, that OpenMath largely overlaps with MathML recommendation of the W3C, however, as MathML is concerned with the *presentation* of mathematical objects, OpenMath is concerned with their semantics (*content*). OpenMath objects can be embedded into MathML objects. One of the intentions of OpenMath is that mathematics encoded using this standard to be communicated between mathematical systems. A number of systems (some of them described in this document, like ActiveMath and GAP) can translate their mathematical objects into OpenMath.

31.2 Technical Information on the System

31.2.1 Name of the System and Website

OpenMath. http://www.openmath.org/index.html.

31.2.2 Project Leaders and Group

The OpenMath Society, based in Helsinki, Finnland, coordinates the OpenMath activites. The Society is coordinated by an elected executive committee. The current OpenMath member group consists of the following persons: Heikki Apiola, Ernesto Barreiro, Stephen Braham, Stephen Buswell, Antonio Capani, Arjeh M. Cohen, Olga Caprotti, David Carlisle, Stephane Dalmas, James Davenport, Stan Devitt, Mike Dewar, Angel Diaz, Andreas Franke, Marc Gaëtano, George Goguazde, Gaston Gonnet, Vilya Harvey, Tanemember Huuskonen, Michael Kohlhase, Stephane Lavirotte, Paul Libbrecht, Bruce Miller, William Naylor, Yves Papegay, Manfred Riem, Mika Seppälä, Elena Smirnova, Clare So, Andrew Solomon, Andreas Strot−mann, Bob Sutor, Richard Timoney, Carlo Traverso, Sarah Turner, Stephen Watt, Renaud Rioboo, Sebastian Xambo, Hans Cuipers, Christine Müller, Normen Müller, Florian Rabe, Christoph Lange, Patrick Ion, Sam Dooley, Robert Miner, Margret Hitchcliffe, Odd Bringslid, Lionel Mamane, Masakazu Suzuki, Matti Pauna.

31.2.3 Main Publications

The OpenMath Society, The OpenMath Standard. Version: 2.0. Editors: S.Buswell, O.Ca−protti, D.P.Carlisle, M.C.Dewar, M.Gaëtano and M.Kohlhase. June 2004. (Available from the society’s webpage.)

31.2.4 Implementation Language

There are libraries which help develop OpenMath documents. Most of them are written in Java, C and C++.

31.2.5 System Availability and Prerequisites

31.3 Algorithm Libraries

31.3.1 Numerical, Discrete, Algebraic, etc. Libraries

31.3.2 Reasoners

31.3.3 Graphical Tools and Interfaces

There are a number of programs designed to help a user edit OpenMath content. We shortly describe some of them, and point the reader to the OpenMath website for further information.

CD Editor is a program with a graphical user interface for editing OpenMath Content Dictionaries (described in the subsections below).

The JOME OpenMath Editor, written in Java, helps users create and manipulate OpenMath (and MathML) objects. It has its own project page on sourceforge.net. Another Java written component which translates linear input into OpenMath (or MathML) is the proprietary STARS/-MathWriter (uses XML encodings).

In addition to these, there is an OpenMath/OMDoc Emacs Mode for creating OpenMath objects.

31.4 User Language

31.4.1 Programming Language

31.4.2 Logic Language for the Formulation of Mathematical Knowledge

The OpenMath standard allows formulating mathematics in any logic the user chooses.

31.4.3 Mathematical Syntax

Mathematical notions are represented as labelled trees which are called OpenMath objects or OpenMath expressions. There are three kinds of OpenMath (OM) expressions: basic objects, derived objects, and compound objects (recursively defined from basic objects and derived objects).

Basic OM objects are either integer, IEEE floating point number, character string, bytearray, variable, or symbol objects. From all these, symbols are of more interest. They contain of a *name*, a *Content Dictionary name*, and (optionally) a *Content Dictionary base URI*. Content Dictionaries (CDs for short) contain the definitions of symbols, and, optionally, additional properties about the symbols, in formal and non−formal description (see examples in the Exam−ple subsection).

Derived OM objects are used for embedding non−OM objects into an OM object.

Compound OM objects are recursively created from basic and derived OM objects by *application*, *attribution*, *binding*, *error* (i.e. creating objects that signal errors).

Content Dictionaries are collections of related symbols together with their meanings. To define a symbol into a CD the following fields are used: the symbol name, a description in plain text, a set of the defined symbol's properties − in plain text (a.k.a. *Commented Mathematical Properties*, CMP), a set of the defined symbol's properties − encoded in OM (a.k.a. *Formal Mathematical Properties*, FMP), one or more usage examples − encoded in OM. Of all these fields, the first two are required. CPM and FMP often come in pairs and can be seen also as usage examples (which, in practise, is done). Related CDs can be grouped together to form *CD Groups*. Extra information can be associated with CDs, like type information or style sheets for rendering OM encodings.

Different encodings of OM objects are possible. The most used one is the XML encoding. In their document describing the OM standard, the OM Society also describes a binary encoding.

For more on the OpenMath syntax we direct the reader to [OpenMath].

31.5 Mathematical Knowledge Bases

31.5.1 Available Theories and Knowledge Bases

The mathematical content available in OpenMath is stored in Content Dictionaries. Cur−rently they contain encodings for linear algebra, polynomials, group theory, transcedental functions, functional operators, units and dimensions, list functions, sets, combinatorics. There are a number of contributed CDs, and authors are encouraged to encode mathematics with OpenMath and submit their encodings to the society.

31.5.2 Tools for Retrieval in Mathematical Knowledge Bases

31.5.3 Tools for Inventing Mathematical Knowledge

31.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

31.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

31.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

31.5.7 Standardization, Inter−Operability

There are various stylesheets to convert OpenMath to MathML and OMDoc (and back), as well as stylesheets for translating MathML to TEX or RTF.

OpenMath is supported by several frameworks and systems as a data exchange format. The IAMC framework (Internet Accessible Mathematical Computation − http://icm.mcs.kent.edu/research/iamc.html) supports OpenMath as a data exchange format between the systems linked to it. The Logic Broker Architecture [ArmandoZini00] provides the infrastructure for reasoning systems to cooperate. Another system which connects mathematical reasoners is MathWeb. Here, the systems communicate via a common software bus and use OMDoc to exchange information. OpenXM (http://www.openxm.org) is a debian package which offers "infrastructure for mathematical communication". They propose their own stan−dard for mathematical computations (OpenXM−RFC) and promise they will support the Open−Math standard soon. Other methods to link computational engines are based on Java, for exam−ple JavaMath, which links computation systems, in particular Maple and GAP, via an API and using OpenMath.

In addition to all these, there are small pieces of software, called "phrasebooks", which do symbol translations between OpenMath and the following systems: Axiom, Gap, Mathematica, CoCoA. Possibly other systems have phrasebooks defined for them, too.

31.5.8 Web Access

31.6 Example of a Theory Exploration Session

The OpenMath Standard [OpenMath] describes two encodings: the XML encoding and the binary encoding. We give here only some examples of the XML encoding.

A variable *x* is encoded as: `<OMV name="x"/>`, where OMV means OpenMath variable.

The *sin* function, defined in one of OpenMath's CDs − "transc1" is encoded as: `<OMS name="sin" cd="transc1"/>.`

To encode the *sin* symbol as an OpenMath object we have to write it as:

```
<OMOBJ>
    <OMS name="sin" cd="transc1"/>
</OMOBJ>
```

If we want to encode the function application *sin(x)* we will use the OMA tag (OpenMath Application) in the following way:

```
<OMOBJ>
    <OMA>
      <OMS name="sin" cd="transc1"/>
      <OMV name="x"/>
    </OMA>
</OMOBJ>
```

Here is how the *sin* function is defined in the *transc1* CD. The first part of the CD *sin* definition contains the name of the symbol (*sin*) it's role (it is an application, i.e. function), and a textual description of it.

```
<CDDefinition>
<Name> sin </Name>
<Role>application</Role>
<Description>
    This symbol represents the sin function as described in
Abramowitz and Stegun, section 4.3. It takes one argument.
</Description>
```

A commented mathematical property (CMP) is encoded as:

```
<CMP>
    sin A = − sin(−A)
</CMP>
```

And the formal mathematical property (FMP) which encodes the commented property above is as follows. (Note the use of the *unary−minus* symbol which is defined in the arith1 CD and of the *eq* symbol which is defined in the relation1 CD.)

```
<FMP>
  <OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0"
cdbase="http://www.openmath.org/cd">
    <OMA>
      <OMS cd="relation1" name="eq"/>
      <OMA>
        <OMS cd="transc1" name="sin"/>
    <OMV name="A"/>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="unary_minus"/>
    <OMA>
      <OMS cd="transc1" name="sin"/>
      <OMA>
        <OMS cd="arith1" name="unary_minus"/>
        <OMV name="A"/>
```

```
        </OMA>
      </OMA>
        </OMA>
      </OMA>
    </OMOBJ>
  </FMP>
  </CDDefinition>
```

32 Plural

Note: It is recommended that the chapter on the Singular system is read before this one.

32.1 Short Description

Plural is a kernel extension of Singular, which is designed for considerably fast computations within the class of non−commutative polynomial algebras [Plural].

32.2 Technical Information on the System

32.2.1 Name of the System and Website

Plural. http://www.singular.uni−kl.de/plural/

32.2.2 Project Leaders and Group

The development of the system is coordinated by Gert−Martin Greuel, Viktor Levandovskyy, and Hans Schönemann (all from Department of Mathematics, University of Kaiserslautern). Contributions to the system were made by F.J. Lobillo, C. Rabelo, Yuriy Drozd, Oleksander Khomenko, Oleksandr Motsak, and Lesya Bodnarchuk.

32.2.3 Main Publications

32.2.4 Implementation Language

C/C++

32.2.5 System Availability and Prerequisites

Free software, distributed under the terms of the GNU General Public License (version 2 of the License). Downloadable from the project’s website, it’s available as a binary program for a variety of Unix platforms, Windows, MacOs X, and others.

32.3 Algorithm Libraries

The system is built as an offspring of Singular, by modifying the Singular kernel to deal with non−commutative algebra. I addition to the Singular libraries (see the respective section of this document) Plural has some specific libraries that come with the system. We mention these libraries below.

32.3.1 Algebraic Library

In addition to the Singular libraries, Plural comes with the following libraries:

- Definitions of important G−algebras;
- Central character decomposition of a module;
- General tools for noncommutative algebras;
- Procedures for calculating the Gelfand−Kirillov dimension;
- Quantum matrices, quantum minors and symmetric groups.

32.3.2 Reasoners

32.3.3 Graphical Tools and Interfaces

32.4 User Language

See the section about the Singular section. Plural uses the same language as Singular.

32.5 Mathematical Knowledge Bases

32.5.1 Available Theories

32.5.2 Tools for Retrieval in Mathematical Knowledge Bases

32.5.3 Tools for Inventing Mathematical Knowledge

32.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

32.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

32.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

32.5.7 Standardization, Inter−Operability

32.5.8 Web Access

32.6 Example of a Theory Exploration Session

See the section about the Singular system to get the feeling how a session within Plural looks like.

33 PVS

33.1 Short Description

PVS is an environment for formal specification and verification. The system consists of a specification language, a number of predefined theories, a type checker, an interactive theorem prover and a symbolic model checker, various utilities including a code generator and a random tester, formalized libraries. The PVS system is successfully used in areas like requirement checking, hardware verification, verifications of practical fault−tolerant systems, compiler correctness [OwreEtAl98].

33.2 Technical Information on the System

33.2.1 Name of the System and Website

PVS Prototype Verification System. http://pvs.csl.sri.com/index.shtml

33.2.2 Project Leaders and Group

The system is developed at SRI International, California, Computer Science Laboratory.
Principal investigator: John Rushby. Group members: Patrick Lincoln, Sam Owre, Natarajan Shankar, Ashish Tiwari.

33.2.3 Main Publications

Owre, S. and Rushby, J.M. and Shankar, and N., "PVS: A Prototype Verification System", in 11th *International Conference on Automated Deduction* (CADE), vol. 607, 1992, pp. 748−752.

33.2.4 Implementation Language

Common Lisp. (Currently they have ported to CMU Common Lisp, which is freely avail−able. However, they also provide binaries compiled with Allegro Common Lisp).

33.2.5 System Availability and Prerequisites

The system is available under the GPL licence. Pre−built binaries are available for various Linux/Unix systems, however Allegro Lisp or CMU Lisp should be installed on the system. Users can build (compile) the system themselves for other versions of Lisp (e.g. Common Lisp), too.

33.3 Algorithm Libraries

33.3.1 Numerical, Discrete, Algebraic, Etc. Libraries

The system does not have a library of algorithms as CASs have. The solution used to evalu−ate expressions like sin(13/25) is semantic attachments to refer to the (lisp) programs attached to predicate or function symbols. Only ground expressions (i.e. do not have higher−order or uninterpreted terms, and are formed from boolean and/or integer types) are evaluated. For details on how programming attachments in PVS works, see [CrowEtAl01].

33.3.2 Reasoners

The PVS system has a proof checker that is meant to support the efficient development of readable proofs [OwreEtAl02]. It implements a set of powerful inference rules, a mechanism to compose them into proof strategies, a mechanism to rerun proofs. The inference rules include propositional and quantifier steps, beta−reduction, equality replacement, use of lemmata, etc.

33.3.3 Graphical Tools and Interfaces

PVS uses Gnu or XEmacs to provide an integrated interface to its specification language and prover. The Emacs front−end uses various buffers for the proof interaction, theories, help, etc. It also have an optional graphical proof tree window implemented in Tcl/Tk. There's also the possibility to view theory hierarchies graphically, again, using Tcl/Tk.

33.4 User Language

33.4.1 Programming Language

To write algorithms as semantical attachments, the Lisp language is used.

33.4.2 Logic Language for the Formulation of Mathematical Knowledge

The logic language of PVS (also called the 'specification language') is a strongly typed higher−order with a rich type system [OwreEtAl02]. The terms of the language are constructed using function application, lambda abstraction, record and tuple construction. For details about PVS's language see [OwreEtAl99].

33.4.3 Mathematical Syntax

The syntax of the specification language.

33.5 Mathematical Knowledge Bases

33.5.1 Available Theories and Knowledge Bases

The PVS system as is provides a rather small library of formal content, most of it being needed for doing prototype specification checks. The 'prelude' library, distributed with the system, contains formalizations of the basic mathematical concepts, such as sets, bags, func−tions, relations, equality, ordering, numbers, etc. This library is built in to PVS. External librar−ies contain formalizations of finite sets, floor, div/mod, bitvectors, coalgebras, real analysis, graphs, temporal logics, quaternions, μ−calculus.

The many users of the system, in course of their work, do create a large body of knowledge. One example of a large body of formalization in PVS is the Nasa Langley PVS libraries [Nasa−Langley] which contains theories from algebra, complex numbers, logarithm, exponential and hyperbolic functions, topology, trigonometry − just to name a few. However, the many user−created libraries are not centralized or coordinated in anyway.

33.5.2 Tools for Retrieval in Mathematical Knowledge Bases

The system has a simple search mechanism accessible via the library browsing commands (search for declarations of a given symbol, declarations which refer an identifier/declaration).

33.5.3 Tools for Inventing Mathematical Knowledge

33.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The PVS proof checker.

33.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

33.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

33.5.7 Standardization, Inter−Operability

The system supports LaTeX output for the libraries content. The libraries have been included into FDL.

33.5.8 Web Access

33.6 Example of a Theory Exploration Session

The examples in this section are taken from [OwreEtAl99].

The PVS system consists of a specification language, a parser, a type−checker, a prover, specification libraries, and various browsing tools.

33.6.1 Specification Definitions Example

The specification language permits modularity and allows reuse by theory parametrization. A specification theory is a sequence of declarations which give names for types, constants, variables, axioms and formulae.

We give here an example of how to naively encode various set−theoretic operations.

```
sets_ [T: TYPE] : THEORY
    BEGIN
        set: TYPE = SETOF[T]
        member(x:T,a:set): bool = a(x)
        union(a,b:set): set = { x:T | member(x,a) OR
member(x,b) }
        intersection(a,b:set): set = { x:T | member(x,a) AND
member(x,b) }
        difference(a,b:set) : set = { x:T | member(x,a) AND NOT
member(x,b) }
        add(x:T,a:set) : set = { y:T | x = y OR member(y,a) }
        singleton(x:T) : set = { y:T | y = x }
        subset?(a,b:set) : bool = (FORALL (z:T) : member(z,a)
=> member(z,b))
        strict_subset?(a,b:set) : bool = subset?(a,b) AND a /=
b
        empty?(a:set) : bool = (FORALL (x:T) : NOT member(x,a))
        emptyset: set = { x:T | FALSE }
        fullset: set = { x:T | TRUE }
        extensionality: LEMMA
            FORALL (a,b: set):
                (FORALL (x:T): member(x,a) = member(x,b)) => (a
= b)
END sets_
```

The theory above is parametrized by T, so all the predicates are over the type T. The extensionality for sets can be proved from the extensionality for functions, so the extensionality axiom for sets is stated as a lemma.

Infinite sequences of elements of some type T are formalized as functions from natural numbers into T

```
sequences_[T: TYPE] : THEORY
    BEGIN
        sequence : TYPE = [nat−>T]
        nth(seq: sequence, n: nat): T = seq(n)
        suffix(seq:sequence, n:nat): sequence =
                    (LAMBDA (i:nat): seq(i+n))
        first(seq: sequence): T = nth(seq, 0)
        rest(seq: sequence): sequence = suffix(seq, 1)
END sequences_


groups [G : TYPE,
    e : G,
    o : [G,G−>G],
    inv : [G−>G] ] : THEORY
BEGIN
  ASSUMING
    a, b, c : VAR G
    associativity :  ASSUMPTION  a o (b o c) = (a o b) o c
    unit : ASSUMPTION  e o a = a AND a o e = a
    inverse : ASSUMPTION inv(a) o a = e AND a o inv(a) = e
```

```
  ENDASSUMING

  left_cancellation: THEOREM a o b = a o c IMPLIES b = c
  right_cancellation: THEOREM b o a = c o a IMPLIES b = c

END groups


sum2: THEORY
BEGIN
  n : VAR nat
  f,g : VAR [nat -> nat]

  sum(f,n) : RECURSIVE nat =
    IF n = 0 THEN
      0
    ELSE
      f(n-1) + sum(f, n - 1)
    ENDIF
  MEASURE n

  sum_plus : LEMMA
    sum((lambda n : f(n) + g(n)), n)
   = sum(f,n) + sum(g,n)

  square(n) : nat = n * n

  sum_of_squares : LEMMA
    6 * sum(square, n+1) = n * (n+1) * (2*n + 1)

  cube(n) : nat = n * n * n

  sum_of_cubes : LEMMA
    4 * sum(cube, n+1) = n*n*(n+1)*(n+1)

END sum2
```

33.6.2 Proof Checker Example

Goals and subgoals in PVS are represented as sequents of the form $\Gamma \vdash \Delta$. During proving, PVS builds up a tree of sequents. We present here an example using quantifiers. We define a theory which contains one theorem we want to prove.

```
predicate: THEORY
    BEGIN
        T : TYPE
        x, y, z: VAR T
        P, Q : [T -> bool]

        pred_calc: THEOREM
            (FORALL x: P(x) AND Q(x))
            IMPLIES (FORALL x: P(x)) AND (FORALL x: Q(x))
END predicate
```

Here is the proof script used to prove pred_calc (the script is edited with Emacs − the stan−dard editor for PVS).

```
pred_calc :
 |----------
{1} (FORALL x: P(x) AND Q(x)) IMPLIES (FORALL x: P(x)) AND
(FORALL x: Q(x))

Rule?
```

the "Rule?" prompt above points out where the system waits for user input. We choose to apply the (flatten) rule.

```
Rule? (flatten)
Applying disjunctive simplification to flatten sequent,
this simplifies to:
pred_calc :

 {-1} (FORALL x: P(x) AND Q(x))
    |----------
 {1} (FORALL x: P(x)) AND (FORALL x: Q(x))

Rule?
```

Notice that PVS marks the formulae in the sequent's antecedent with negative numbers.

Here we choose now to split the goal, then − on the first subgoal − we will skolemize and instantiate universally.

```
Rule? (split)
Splitting conjunctions,
this yields 2 subgoals:
pred_calc.1 :

 [-1] (FORALL x: P(x) AND Q(x))
    |----------
 {1} (FORALL x: P(x))

Rule? (skolem 1 "X")
For the top quantifier in 1, we introduce Skolem constants: X
this simplifies to:
pred_calc.1 :

 [-1] (FORALL x: P(x) AND Q(x))
    |----------
 {1} P(X)

Rule? (inst -1 "X")
Instantiating the top quantifier in -1 with the terms: X
this simplifies to:
pred_calc.1 :

 {-1} P(X) AND Q(X)
    |----------
 [1] P(X)
```

```
 Rule? (prop)
By propositional simplification,

This completes the proof of pred_calc.1.
```

In both the skolemization and instantiation steps above, we have specified the formula and the variable to skolemize/instantiate. For the second branch of the proof we let the system choose suitable formulae, variables and substitution terms by using the (skolem!) and (inst?) rules. Notice how the newly created terms are named:

```
pred_calc.2 :

 [-1] (FORALL x: P(x) AND Q(x))
    |----------
 {1} (FORALL x: Q(x))

Rule? (skolem!)
Skolemizing,
this simplifies to:
pred_calc.2 :

 [-1] (FORALL x: P(x) AND Q(x))
    |----------
 {1} Q(x!1)

Rule? (inst?)
Found substitution:
x gets x!1,
Instantiating quantified variables,
this simplifies to:
pred_calc.2 :

 {-1} P(x!1) AND Q(x!1)
    |----------
 [1] Q(x!1)

 Rule? (prop)
By propositional simplification,

This completes the proof of pred_calc.2.
Q.E.D.
```

Finally, we point out that the reason for which the formulae numbers occur sometimes in square sometimes in curly brackets is the following: the PVS checker marks the formulae that were affected/newly created by a proof step with curly brackets. The square brackets mark the formulae that were left unchanged by the proof step.

34 QED

34.1 Short Description

The QED project aims at building a computer system which was to represent mathematical knowledge and techniques. It was recognized in the QED Manifesto (written by anonymous authors) that it was to be a cooperation of many mathematicians, computer scientists and research agencies. Among the list of reasons for undertaking such an effort we mention: rigor−ous, incremental representation of mathematical knowledge which would make it usable to scan and search for specific or relevant results; key component for systems that do verification and synthesis of both software and hardware parts; framework for carrying out proofs in academic education; preserve mathematics from corruption; noise level reduction by avoiding redundan−cies, duplications, etc.; improve the coherence of mathematical knowledge.

The system would build up on a ’root logic’, but would also be accepting other logics provided there are methatheorems available to share proofs between logics. It was expected that the full foundation of the system could be expressed in a few pages of mathematics. The knowl−edge to be stored in the system are to be checked, different, independent basic checkers being supported by QED. The system would only be focused on formalizing and checking mathemat−ics, and not discovering new theorems.

Though the project has been dormant since some time, several projects and systems (many of them described also in this report) can be seen as trying to achieve some of the goals expressed in the QED Manifesto [QEDMan].

34.2 Technical Information on the System

34.2.1 Name of the System and Website

The QED Project. http://www−unix.mcs.anl.gov/qed/

34.2.2 Main Publications

The QED Manifesto. In Automated Deduction − CADE 12, Springer−Verlag, Lecture Notes in Artificial Intelligence, Vol. 814, pp. 238−251, 1994.

35 Singular

35.1 Short Description

Singular is a Computer Algebra System for polynomial computations with special emphasis on the needs of commutative algebra, algebraic geometry, and singularity theory [Singular]. Its main computational objects are polynomials, ideals and modules over a large variety of rings [GreuelPfister06]. It implements one of the fastest algorithm for computing Gröbner bases. The problems in non−commutative algebra can be tackled with Singular's offspring, Plural (see the corresponding section in this document).

35.2 Technical Information on the System

35.2.1 Name of the System and Website

Singular. http://www.singular.uni−kl.de/

35.2.2 Project Leaders and Group

The project is coordinated by Gert−Martin Greuel, Gerhard Pfister, and Hans Schönemann (all from Department of Mathematics, University of Kaiserslautern).

The project group currently consist of: Michael Brickenstein, Wolfram Decker, Alexander Dreyer, Anne Frühbis−Krüger, Kai Krüger, Viktor Levandovskyy, Oleksandr Motsak, Oliver Wienand

For former member of the team and other contributors to the system see the webpage.

35.2.3 Main Publications

G.−M. Greuel, G. Pfister. SINGULAR and Applications. Jahresbericht der DMV 108 (2006), 167−196.

C. Lossen, H. Schönemann. 21 Years of Singular Experiments in Mathematics. In: C. Lossen and G. Pfister (eds.), Singularities and Computer Algebra. Lecture Notes of LMS, Cambridge University Press, to appear (2005/06).

35.2.4 Implementation Language

Initially, the system was programmed in Modula−2, later it was switched to C. Several libraries of the system are implemented in C++.

35.2.5 System Availability and Prerequisites

Free software, distributed under the terms of the GNU General Public License (version 2 of the License). Downloadable from the project's website, it's available as a binary program for a variety of Unix platforms, Windows, MacOs X, and others.

As a prerequisite, we mention (X)Emacs, if it is not wished to interact with the system in terminal window.

35.3 Algorithm Libraries

35.3.1 Algebraic Library

There is one basic algorithm which is central in Singular. It is a general standard basis algorithm for any monomial ordering (including well−orderings − Buchberger's algorithm − and tangent cone orderings − Mora's algorithm).

The algorithms implemented in Singular can be grouped as follows (see the manual on the system's website [Singular]):

* Algorithms to compute the standard operations on ideals and modules: intersection, ideal quotient, elimination, etc.
* Different Syzygy algorithms and algorithms to compute free resolutions of modules.
* Combinatorial algorithms to compute dimensions, Hilbert series, multiplicities, etc.
* Algorithms for univariate and multivariate polynomial factorization, resultant and gcd computations.

35.3.2 Reasoners

35.3.3 Graphical Tools and Interfaces

35.4 User Language

35.4.1 Programming Language

The system provides the Singular language, very similar to the C programming language. The building blocks of the Singular language are expressions, commands and control structures.

35.4.2 Logic Language for the Formulation of Mathematical Knowledge

The system has no means to formulate theorems, definitions, etc. It is meant to program algorithms, in the Singular language, and then execute them.

35.4.3 Mathematical Syntax

The Singular language.

35.5 Mathematical Knowledge Bases

35.5.1 Available Theories and Knowledge Bases

35.5.2 Tools for Retrieval in Mathematical Knowledge Bases

35.5.3 Tools for Inventing Mathematical Knowledge

35.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

35.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

35.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

35.5.7 Standardization, Inter−Operability

Various packages from other system to Singular are available (e.g. from Gap to Singular − by M. Costantini, from Mathematica to Singular − by M. Kauers and V. Levandosky).

35.5.8 Web Access

35.6 Example of a Theory Exploration Session

The examples below are taken from the on−line manual that can be found on the Singular's website.

35.6.1 Introduction

When Singular is started, it prompts the '>' character. Every statement has to be terminated by ';' .

```
>37+5;
42
```

All objects have a type, e.g., integer variables are defined by the word int. Assignments are done by the symbol '=' . Note that the output is suppressed. This is the case when assignments are made.

```
>int k=2;
```

Testing equality / inequality is done using '==' / != (or <>) − as in C and C++, where 0 represents the boolean value FALSE, any other value represents TRUE.

```
> k==2;
1

>k≠2;
0
```

The value of an object is displayed by simply typing its name.

```
> k;
2
```

The last displayed (!) result is always available with the special symbol _ .

```
> 2*_;//the value from k displayed above
4
```

35.6.2 Declaring Rings

All the computations in Singular are done within some ring. The default one is ℤ/32003[x,y,z]with degree reverse lexicographical ordering.

```
> ring r=32003,(x,y,z), dp;
```

The ring ℚ[a,b,c,d] with lexicographical ordering:

```
> ring r=0,(a,b,c,d), lp;
```

The ring $\mathbb{Z}/7[x_1, ..., x_6]$ with lexicographical ordering for x_1, x_2, x_3 and degree reverse lexicographical ordering for x_4, x_5, x_6:

```
> ring r=7, (x(1..6)), (lp(3),dp);
```

35.6.3 Solving Systems of Polynomial Equations

The Singular libraries solve.lib and triang.lib provide several commands for solving systems of polynomial equations (based on a symbolic−numerical approach via Groebner bases, resp. resultants). In the example below, we show some of these commands at work.

```
>LIB "solve.lib";
>ring r=0,x(1..5),dp;
>poly f0=x(1)^3+x(2)^2+x(3)^2+x(4)^2−x(5)^2;
>poly f1=x(2)^3+x(1)^2+x(3)^2+x(4)^2−x(5)^2;
>poly f2=x(3)^3+x(1)^2+x(2)^2+x(4)^2−x(5)^2;
>poly f3=x(4)^2+x(1)^2+x(2)^2+x(3)^2−x(5)^2;
>poly f4=x(5)^2+x(1)^2+x(2)^2+x(3)^2;
>ideal i=f0,f1,f2,f3,f4;
>ideal si=std(i);
```

The dimension of the solution set (0 in this case) can be read from a Gröbner basis, w.r.t. any global monomial ordering

```
>dim(si);
0
```

The number of complex solutions (counted with multiplicities) is:

```
>vdim(si);
108
```

The given system has a multiple solution at the origin. We use *facstd* to compute equations for the non−zero solutions:

```
>option(redSB);
>ideal maxI=maxideal(1);
>ideal j=sat(si,maxI)[1];//output is Groebner basis
>vdim(j);//number of non−zero solutions (with mult's)
76
```

We compute a triangular decomposition for the ideal I. This requires first the computation of a lexicographic Gröbner basis (we use the FGLM conversion algorithm):

```
>ring R=0,x(1..5),lp;
>ideal j=fglm(r,j);
>list L=triangMH(j);
>size(L);//number of triangular components
7


> L[1];                    // the first component
_[1]=x(5)^2+1
_[2]=x(4)^2+2
_[3]=x(3)−1
_[4]=x(2)^2
_[5]=x(1)^2
```

We compute floating point approximations for the solutions (with 30 digits)

```
>def S=triang_solve(L,30);
//'triang_solve' created a ring,in which a list rlist of
numbers (the
```

```
//complex solutions) is stored.
//To access the list of complex solutions,type (if the name R
was assigned
//to the return value):
setring R;rlist;


>setring S;
>size(rlist);//number of different non−zero solutions
28


>rlist[1];//the first solution
[1]:
      0
[2]:
      0
[3]:
      1
[4]:
      (−I*1.41421356237309504880168872421)
[5]:
      −I
```

Alternatively, we could have applied directly the solve command:

```
>setring r;
>def T=solve(i,30,1,"nodisplay");//compute all solutions with
mult’s
//’solve’ created a ring,in which a list SOL of numbers (the
complex solutions)
//is stored.
//To access the list of complex solutions,type (if the name R
was assigned
//to the return value):
setring R;SOL;


>setring T;
>size(SOL);//number of different solutions
4


>SOL[1][1];SOL[1][2];//first solution and its multiplicity
[1]:
      [1]:
            1
      [2]:
            1
      [3]:
            1
      [4]:
            (i*2.449489742783178098197284074706)
      [5]:
            (i*1.732050807568877293527446341506)
[2]:
      [1]:
            1
```

```
      [2]:
            1
      [3]:
            1
      [4]:
            (-i*2.44948974278317809819728407 4706)
      [5]:
            (i*1.73205080756887729352744634 1506)
[3]:
      [1]:
            1
      [2]:
            1
      [3]:
            1
      [4]:
            (i*2.44948974278317809819728407 4706)
      [5]:
            (-i*1.73205080756887729352744634 1506)
[4]:
      [1]:
            1
      [2]:
            1
      [3]:
            1
      [4]:
            (-i*2.44948974278317809819728407 4706)
      [5]:
            (-i*1.73205080756887729352744634 1506)
1

>SOL[size(SOL)];//solutions of highest multiplicity
[1]:
   [1]:
      [1]:
            0
      [2]:
            0
      [3]:
            0
      [4]:
            0
      [5]:
            0
[2]:
     32
```

Or, we could remove the multiplicities first, by computing the radical:

```
>setring r;
>ideal k=std(radical(i));
>vdim(k);//number of different complex solutions
29
```

```
>def T1=solve(k,30,"nodisplay");//compute all solutions with
mult’s
//’solve’ created a ring,in which a list SOL of numbers (the
complex solutions)
//is stored.
//To access the list of complex solutions,type (if the name R
was assigned//to the return value):setring R;SOL;
>setring T1;
>size(SOL);//number of different solutions
29

>SOL[1];
[1]:
    1
[2]:
    1
[3]:
    1
[4]:
    (-i*2.44948974278317809819728407 4706)
[5]:
    (-i*1.73205080756887729352744634 1506)
```

36 Theorema

36.1 Short Description

The *Theorema* is a project and a software system that aims at supporting the entire process of mathematical theory exploration. It is built on top of the computer algebra system Mathematica.

36.2 Technical Information on the System

36.2.1 Name of the System and Website

Theorema. A System for Computer Supported Mathematical Theorem Proving and Theory Exploration.

http://www.theorema.org.

36.2.2 Project Leaders and Group

Bruno Buchberger and Tudor Jebelean (Risc−Linz).

Current members of the group are: Adrian Craciun, Besik Dundua, Madalina Erascu, Teimu−raz Kutsia, Florina Piroi, Nikolaj Popov, Camelia Rosenkranz, Markus Rosenkranz, Robert Vajda, Alexander Zapletal, Wolfgang Windsteiger.

36.2.3 Main Publications

B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, W. Windsteiger. Theorema: Towards Computer−Aided Mathematical Theory Exploration. *Journal of Applied Logic* 4(4), pp. 470−504. 2006. ISSN 1570−8683.

B. Buchberger, T. Jebelean, W. Windsteiger, T. Kutsia, K. Nakagawa, J. Robu, F. Piroi, A. Craciun, N. Popov, G. Kusper, M. Rosenkranz, L. Kovacs, C. Kocsis. F 1302: THEOREMA: Proving, Solving and Computing in General Domains. In: Special Research Program (SFB) F 013, Numerical and Symbolic Scientific Computing, Proposal for Continuation, Part I: Progress Report, April 2001−September 2003, P. Paule, U. Langer (ed.), pp. 148−170. October 2003. Johannes Kepler University Linz, Austria

36.2.4 Implementation Language

The programming language of Mathematica. *Theorema* does not call any Mathematica algorithms, unless the user explicitly, and in a controlled way, does this from within the system.

36.2.5 System Availability and Prerequisites

The system is free of charge. It is distributed as a collection of Mathematica packages. I.e. Mathematica is needed for running the system. Optionally, external proof systems which *Theorema* can call should be locally installed.

36.3 Algorithm Libraries

36.3.1 Numerical, Discrete, Algebraic, etc. Libraries

36.3.2 Reasoners

The system has a general higher−order predicate logic and a collection of special provers, like set theory prover, tuple induction prover, polynomial simplifier, equational provers, etc. See the various publications of the *Theorema* group for detailed descriptions on each of the reasoners.

36.3.3 Graphical Tools and Interfaces

The *Theorema* system uses Mathematica's front−end to interact with the user, and, therefore, takes advantage of its graphical capabilities (hyperlinks, colors, collapsing cells, etc). In addition to this, several graphical tools were implemented:

Logicographic symbols [NakagawaBuchberger01] provide means for introducing new intuitive notation. Logicographic symbols are graphical symbols that carry logical meaning, and can be used in the proof search mechanism.

Focus Windows [PiroiBuchberger02] is a tool for inspecting completed proofs. Proof steps are shown in a two−phase style, before and after the inference rule application on some given proof step in the proof tree.

Theorema's interactive proving environment is described in [PiroiKutsia05], and includes a visualization of the proof tree.

36.4 User Language

36.4.1 Programming Language

The *Theorema* language for implementing provers and algorithms is available, for the moment, only to the project's members.

36.4.2 Logic Language for the Formulation of Mathematical Knowledge

Untyped higher order logic extended with sequence variables.

36.4.3 Mathematical Syntax

The *Theorema* language syntax.

36.5 Mathematical Knowledge Bases

36.5.1 Available Theories and Knowledge Bases

There exist several theory formalizations done with *Theorema*, but these are not centralized. The SysteMaThEx project (Systematic Mathematical Theory Exploration with the Theorema System : Case Studies) aims at providing major case studies of computer−supported systematic theory exploration (see [HodorogCraciun07] and [CraciunHodorog07].

An environment for editing and building mathematical knowledge libraries is described in [PiroiBuchberger04].

36.5.2 Tools for Retrieval in Mathematical Knowledge Bases

A formula search mechanism that involves alpha conversion, and search by formulae containing certain symbols or patterns has already been implemented (see [PiroiEtAl07]). There is ongoing work to implement more sophisticated tools for mathematical knowledge retrieval, like, for example, retrieval by simple proving (see [Rosenkranz07]).

36.5.3 Tools for Inventing Mathematical Knowledge

There are two existing tools for inventing mathematical lemmata in *Theorema*. The first one, the cascade tool, infers new lemmata from the information generated by failing proofs, and recursively calls the original prover on the lemmata inferred. The 'lazy thinking' method is a systematic method for top−down synthesis and verification of lemmata and algorithms. See [BuchbergerCraciun03], for more details and further references to this paradigm.

Another tool for inventing knowledge is described in [Kovacs07] and treats the generation of invariants for while−loops, which is a helpful tool for (imperative) program verification

36.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

The *Theorema* reasoners.

36.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

36.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

There's ongoing work to add facilities for implementing reasoners within *Theorema*. The reasoners are to be first proved by the system, before including them in it. See [Giese−Buchberger07] for more details.

36.5.7 Standardization, Inter−Operability

Besides its 'internal' reasoners, *Theorema* can use 'external' automated reasoning systems via a special interface. The interface links *Theorema* with the external provers Bliksem [de−Nivelle07], EQP [McCune−EQP], E [Schulz02], Gandalf [Tammet97], Otter [McCune94], Scott [HodgsonSlaney00], Setheo [SchumannEtAl90], Spass [WeidenbachEtAl99], Vampire [RiazanovVoronkov99], Waldmeister [HillenbrandEtAl96], and with the finite model and counterexample searcher Mace [McCune03]. For more details on the interface see [Kutsia−Nakagawa01].

36.5.8 Web Access

36.6 Example of a Theory Exploration Session

We show an example of using the set theory prover of the *Theorema* system. First we load the *Theorema* packages, then we give some definitions of what a relation is, its equivalence class, factor set, etc. (see below). The definitions are all given in *Theorema*'s user language.

```
<< Theorema‘
```

```
Definition["is relation", any[R],
  is-relation[R] : ⇔ (R ⊆ X×X)]
```

```
Definition["class", any[x, R], class[R, x] := {a | a∈X ⟨a, x⟩ ∈ R}]
```

```
Definition["factor set", any[R], factor-set[R] := {class[R, x] | x ∈ X}];
                                                  x
```

```
Definition["is subset set", any[P], is-subset-set[P] :⇔ ∀ p ⊆ X]
                                                        p∈P
```

```
Definition["is reflexive", any[R],
  is-reflexive[R] :⇔ ∀ ⟨x, x⟩ ∈ R]
                     x∈X
```

```
Definition["is all non empty", any[P],
  is-all-nonempty[P] :⇔ (∀ p ≠ {})]
                       p∈P
```

We want to prove the following lemma, which states that if a relation is reflexive, then it's factor set is not empty. The 'Prove' command contains the knowledge that can be used by the set theory prover (aside from its implemented inference rules), and some parameters which guide the proof searching process.

```
Lemma["one", any[R],
  is-reflexive[R] ⇒ is-all-nonempty[factor-set[R]]]
```

```
Prove[Lemma["one"],
  using → ⟨Definition["factor set"], Definition["class"], Definition["is relation"], Definition["is reflexive"],
      Definition["is subset set"], Definition["is all non empty"]⟩, by → SetTheoryPCSProver,
  ProverOptions → {GRWTarget → {"goal", "kb"}, UseCyclicRules → True, RWExistentialGoal → True,
      DisableProver → {STC, PND}}, SearchDepth → 50, transformBy → ProofSimplifier,
  TransformerOptions → {branches → Proved, steps → Useful}];
```

And the content of the proof notebook that shows the proof found by the system is:

Prove:

(Lemma (one)) $\forall_{R}$ (is-reflexive[R] ⇒ is-all-nonempty[factor-set[R]]) ,

under the assumptions:

(Definition (factor set)) $\forall_{R}\left(\text{factor-set}[R] := \{\text{class}[R, x] \mid x \in X\}\right)$,

(Definition (class)) $\forall_{R,x}\left(\text{class}[R, x] := \{a \mid a \in X \wedge \langle a, x\rangle \in R\}\right)$,

(Definition (is relation)) $\forall_{R}$ (is-relation[R] :⇔ $R \subseteq X\times X$) ,

(Definition (is reflexive)) $\forall_{R}\left(\text{is-reflexive}[R] :\Leftrightarrow \forall_{x} (x \in X \Rightarrow \langle x, x\rangle \in R)\right)$,

(Definition (is subset set)) $\forall_{P}\left(\text{is-subset-set}[P] :\Leftrightarrow \forall_{p} (p \in P \Rightarrow p \subseteq X)\right)$,

(Definition (is all non empty)) $\forall_{P}\left(\text{is-all-nonempty}[P] :\Leftrightarrow \forall_{p} (p \in P \Rightarrow p \neq \{\})\right)$.

Using built-in simplification rules we can simplify the knowledge base:

Formula (Definition (is all non empty)) simplifies to

(1) $\forall_{P}\left(\text{is-all-nonempty}[P] :\Leftrightarrow \forall_{p} (p \in P \Rightarrow \{\} \neq p)\right)$.

We assume

(2) is-reflexive[R_0] ,

and show

(3) is-all-nonempty[factor-set[R_0]] .

Formula (3), using (Definition (factor set)), is implied by :

$$\text{is-all-nonempty}\left[\left\{\text{class}[R_0, \boldsymbol{x}] \mid \underset{x}{} \boldsymbol{x} \in \mathrm{X}\right\}\right],$$

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

$$\text{is-all-nonempty}\left[\left\{\left\{\boldsymbol{a} \mid \underset{a}{} \boldsymbol{a} \in \mathrm{X} \wedge \langle \boldsymbol{a}, \boldsymbol{x}\rangle \in R_0\right\} \;\middle|\; \underset{x}{} \boldsymbol{x} \in \mathrm{X}\right\}\right],$$

which, using (1), is implied by :

(4) $$\underset{p}{\forall}\left(\boldsymbol{p} \in \left\{\left\{\boldsymbol{a} \mid \underset{a}{} \boldsymbol{a} \in \mathrm{X} \wedge \langle \boldsymbol{a}, \boldsymbol{x}\rangle \in R_0\right\} \;\middle|\; \underset{x}{} \boldsymbol{x} \in \mathrm{X}\right\} \Rightarrow \{\} \neq \boldsymbol{p}\right).$$

We assume

(5) $$p_0 \in \left\{\left\{\boldsymbol{a} \mid \underset{a}{} \boldsymbol{a} \in \mathrm{X} \wedge \langle \boldsymbol{a}, \boldsymbol{x}\rangle \in R_0\right\} \;\middle|\; \underset{x}{} \boldsymbol{x} \in \mathrm{X}\right\},$$

and show

(6) $\{\} \neq p_0$.

From what we already know follows:

From (5) we know by definition of $\left\{T_x \mid \underset{x}{} P\right\}$ that we can choose an appropriate value such that

(7) $a1_0 \in \mathrm{X}$,

(8) $$p_0 = \left\{\boldsymbol{a} \mid \underset{a}{} \boldsymbol{a} \in \mathrm{X} \wedge \langle \boldsymbol{a}, a1_0\rangle \in R_0\right\}.$$

Formula (6) means that we have to show that

(15) $$\underset{p1}{\exists}\,(\boldsymbol{p1} \in p_0).$$

Formula (15), using (8), is implied by :

(16) $$\underset{p1}{\exists}\left(\boldsymbol{p1} \in \left\{\boldsymbol{a} \mid \underset{a}{} \boldsymbol{a} \in \mathrm{X} \wedge \langle \boldsymbol{a}, a1_0\rangle \in R_0\right\}\right).$$

In order to prove (16) we have to show :

(17) $$\underset{p1}{\exists}\,(\boldsymbol{p1} \in \mathrm{X} \wedge \langle \boldsymbol{p1}, a1_0\rangle \in R_0).$$

Now, let $\boldsymbol{p1} := a1_0$. Thus, for proving (17) it is sufficient to prove :

(18) $a1_0 \in \mathrm{X} \wedge \langle a1_0, a1_0\rangle \in R_0$.

We prove the individual conjunctive parts of (18) :

Proof of (18.1) $a1_0 \in \mathrm{X}$:

Formula (18.1) is true because it is identical to (7).

Proof of (18.2) $\langle a1_0, a1_0\rangle \in R_0$:

Formula (2), by (Definition (is reflexive)), implies :

(25) $$\underset{x}{\forall}\,(\boldsymbol{x} \in \mathrm{X} \Rightarrow \langle \boldsymbol{x}, \boldsymbol{x}\rangle \in R_0).$$

Formula (18.2), using (25), is implied by :

(26) $a1_0 \in \mathrm{X}$.

Formula (26) is true because it is identical to (7).

□

37 TPTP

37.1 Short Description

The TPTP (Thousands of Problems for Theorem Provers) Problem Library is a library of test problems for automated theorem proving systems (ATP). The problems part of this library are expressed in the first order logic. The main sources of these problems are existing electronic collections of problems and the ATP literature (see [SutcliffeSuttner99] for bibliography informa tion on the sources). Additionally, several other people have contributed problems to the library.

37.2 Technical Information on the System

37.2.1 Name of the System and Website

Thousands of Problems for Theorem Provers. http://www.tptp.org/

37.2.2 Project Leaders

Geoff Sutcliffe (Department of Computer Science, University of Miami) and Christian Suttner (Institut für Informatik, TU München)

37.2.3 Main Publications

G. Sutcliffe, and C.B. Suttner: The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning* 21(2), 177−203, 1998.

Geoff Sutcliffe, and Christian Suttner: The TPTP Problem Library, Department of Computer Science, James Cook University, Australia No. 99, February 1999.

37.2.4 Implementation Language

The library is a collection of text files, the syntax of the stored knowledge is the one of Prolog.

37.2.5 System Availability and Prerequisites

The library is freely available, copyrighted (c) 1993−2007, by Geoff Sutcliffe & Christian Suttner. Its use and verbatim redistribution are permitted, while modified versions or modified parts of the library require permission.

To be installed and used, the library needs some version of Prolog available on the operating system.

37.3 Algorithm Libraries

37.4 User Language

Remark: the library cannot be modified (and distributed) by others than the copyright hold−ers (authors). Users can play around with it, but it will only be a local, modified copy.

37.4.1 Programming Language

37.4.2 Logic Language for the Formulation of Mathematical Knowledge

First order logic (with formulae written in first order form or conjunctive normal form).

37.4.3 Mathematical Syntax

First order logic (with formulae written in first order form or conjunctive normal form).

37.5 Mathematical Knowledge Bases

TPTP is a repository of problems for testing Automated Theorem Provers (ATP). The problems are expressed in First Order Logic, each problem is stored in one file, in first order form or in conjunctive normal form. The same problem can have different versions depending on the different formulations, or axiomatization used. Problem versions are alternative presenta−tions of an underlying abstract problem [SutcliffeSuttner98]. Theory axiomatizations are kept in separate axiom files, and included in the problem files as needed and appropriate. On mode details about the problem and axiom sets naming, versioning, etc. see the given bibliography.

37.5.1 Available Theories and Knowledge Bases

The current version of the system (as of 30.10.2007) is 3.0.0. The problems of the library are categorized in 6 groups: Logic, Mathematics, Computer Science, Science and Engineering, Social sciences, Other. We detail, below, for each of these groups the areas covered and the axiomatizations currently available.

- **Logic**

  * Combinatory logic. Axiomatizations: Type−respecting combinators, Combinators

  * Logic calculi. Axiomatizations: Wajsberg algebra axioms, Alternative Wajsberg algebra, Propositional logic deduction, Propositional logic, Propositional logic rules and axioms, Propositional modal logic rules and axioms.

* Henkin models. Axiomatizations: Henkin model, Henkin model, Henkin model (equality).

• Mathematics.

* Set theory. Axiomatizations: Set theory membership and subsets, Set theory, Set theory based on Godel set theory, Set theory based on NBG set theory, Set theory based on NBG set theory, Naive set theory based on Goedel's set theory.

* Graph theory. Axiomatizations: Directed graphs and paths.

* Algebra.

Axiomatizations in Boolean Algebra: Ternary Boolean algebra (equality), Boolean algebra, Boolean algebra (equality), Boolean algebra (equality).

Axiomatizations in Robbins algebra: Robbins algebra.

Axiomatizations in Left distributive algebras: Embedding algebra.

Axiomatizations in Lattices: Lattice theory (equality), Lattice theory, Ortholat−tice theory (equality), Quasilattice theory (equality), Weakly Associative Lattices theory (equality), Tarski's fixed point theorem (equality).

Axiomatizations for Groups: Monoids, Semigroups, Group theory, Group theory (equality), Group theory (Named groups), Group theory (Named Semigroups).

Axiomatizations for Rings:Ring theory, Ring theory (equality), Alternative ring theory (equality).

Axiomatizatoin for Homological algebra: Standard homological algebra axioms.

* General algebra. Axiomatizations: Abstract algebra axioms, based on Godel set theory, Median algebra axioms.

* Number theory. Axiomatizations: Number theory, Number theory (equality), Number theory axioms, based on Godel set theory, Number theory (ordinals) axioms, based on NBG set theory, Translating from nXXX to RDN notation.

* Topology. Axiomatizations: Point−set topology.

* Analysis. Axiomatizations: Analysis (limits) axioms for continuous functions, Analy−sis (limits) axioms for continuous functions, A theory of Big−O notation.

* Geometry. Axiomatizations: Tarski geometry, Hilbert geometry, Simple curve axioms, Hilbert geometry axioms, adapted to respect multi−sortedness, Apartness geometry axioms, Ordered affine geometry.

* Field theory. Axiomatizations: Ordered field axioms (axiom formulation glxx), Ordered field axioms (axiom formulation re)

* Category theory. Axiomatizations: Category theory axioms, Category theory (equality) axioms.

• Computer Science.

* Computing theory

* Knowledge representation

* Natural Language Processing

* Planning. Axiomatizations: Blocks world axioms.

* Agents. Axiomatizations: CPlanT system.

* Commonsense Reasoning. Axiomatizations: Standard discrete event calculus axioms.

* Software creation. Axiomatizations: List specification

* Software verification. Axiomatizations: Program verification axioms, NASA soft−ware certification axioms, Cryptographic protocol axioms for messages, Cryptographic protocol axioms for messages, Cryptographic protocol axioms for public, Priority queue checker: quasi−order set with bottom element.

• **Science and Engineering.**

* Hardware creation. Axiomatizations: Definitions of AND, OR and NOT

* Hardware verification. Axiomatizations: Connections, faults, and gates. Axioms from a VHDL design description.

* Medicine. Axiomatizations: Physiology Diabetes Mellitus type 2.

• **Social sciences.**

* Management. Axiomatizations: Inequalities.

• **Other.**

* Syntactic. Axiomatizations: Synthetic domain theory for EBL.

* Puzzles. Axiomatizations: Mars and Venus axioms, Truthtellers and Liars axioms for two types of people, Quo vadis axioms, Sudoku axioms.

* Miscellaneous. Axiomatizations: Sets, numbers, lists, etc, that make up the Isabelle/−HOL library.

37.5.2 Tools for Retrieval in Mathematical Knowledge Bases

There is a script, tptp1T, which help user select lines from either the FOF problem statistics file or the CNF problem statistics file. The statistic files are (text) files where, for each problem file, various problem characteristics are stored, like, for example, which connectives and how many times occur in the formulae of the problem, how many predicates, arities of predicate and functional symbols, etc. One line in the statistic files corresponds to one problem. Problems can be selected by domain, by number of function symbols, by average number of literals per clause, etc.

There are additional tools for manipulating and interogating the TPTP data structure. The tools can be downloaded from the TPTP website, but they are undocumented.

37.5.3 Tools for Inventing Mathematical Knowledge

The TPTP system contains some *generator* files which are used to generate instances of generic problems. An example of such a problem is the N−queens problem: place N queens on an NxN chess board such that no queen attacks another. (Note: the satisfiability of the generated problem may depend on N). The tool used is tptp2X, written in Prolog.

37.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

37.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

The tptp2X program is a multi−functional utility for reformatting, transforming, and generating TPTP problem files. The transformations are applied directly to formulae, and are [Sutcliffe−Suttner98]:

◦ remove the connectives <=, <~>, ~|, and ~& from FOF problems, by rewriting formulae to use the other connectives;
◦ transform FOF problems to CNF problems, using various algorithms
◦ simplify a set of clauses.
◦ clausify and then simplify.
◦ convert a set of first order clauses to a set of first order formulae, converting negated conjecture clauses into a corresponding conjecture.
◦ convert a set of first order clauses to a set of propositional clauses, preserving satisfiability.
◦ reverse the literal ordering in the clauses of CNF problems.
◦ reverse the clause ordering in CNF problems.
◦ reverse the formula ordering in FOF problems.
◦ randomly reorder the clauses and literals in CNF problems or formulae in FOF problems.
◦ reverse the arguments of unit equality clauses in CNF problems.
◦ reverse the arguments of randomly selected unit equality clauses in CNF problems.
◦ remove equality axioms. One can remove any of the reflexivity, symmetry, transitivity, function substitution or predicate substitution axioms.
◦ add missing equality axioms to problems that contain equality.
◦ set the equality axioms in problems that contain equality.
◦ convert CNF problems to a pure equality representation.
◦ Mark Stickel’s magic set transformation to CNF problems (see reference in the mentioned article).
◦ replace all the symbols in the problem by short, meaningless symbols. Useful when only interested in the syntax of the problem. The equality atoms are not affected.

37.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

37.5.7 Standardization, Inter−Operability

Using the tptp2X utility program, problems from the TPTP library can be translated into the following formats:

Bliksem [deNivelle07], CARINE [Carine], CLIN−S [Chu94], CoDe, Dedam [Nieuwenhuis−EtAl97], DFG [HähnleEtAl96], DIMACS [Dimacs], EQP [McCune−EQP], FINDER [Slaney92], GLiDeS [BrownSutcliffe00], ILF [DahnEtAl95], KIF [GeneserethFikes92], lean−TAP [BeckertPosegga95], 3TAP [HähnleEtAl94], Mace4 [McCune03], Mathematica [Wol−fram], METEOR [Astrachan94], MGTP [FujitaEtAl92], OmDoc [Kohlhase00], OSCAR [Pollock90], Otter [McCune94], PROTEIN [BaumgartnerFurbach94], PTTP [Stickel84], SATCHMO [GeislerEtAl97], SCOTT [HodgsonSlaney00], SEM [ZhangZhang96], SETHEO [SchumannEtAl90], SPRFN [Plaisted88], THINKER [Pelletier87], old TPTP format [Sutcliffe−Suttner98], old TPTP format prefix form, FOF and CNF problems to the TPTP format [Sutcliffe-EtAl03], Waldmeister [HillenbrandEtAl96].

37.5.8 Web Access

On the system’s website there’s an interface, System On TPTP, where users can try to solve problems using the reasoning systems that understand the library. The interface allows users to chose which systems to tackle the problem, give parameters to the chosen systems, chose the system’s output format (where possible), etc.

37.6 Example of a Theory Exploration Session

The file format for the problem and axiom files has three main sections. The first one is a header section with information for the user. This information is not used by the reasoners that may work on the problem. The second section is a list of instructions for axiom files, and the third is a list of formulae that are specific to the problem or axiomatization [SutcliffeSuttner98].

Here is an example of a problem file, as stored in the TPTP library:

```
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−
% File     : NUM291+1 : TPTP v3.3.0. Released v3.1.0.
% Domain   : Number Theory (RDN arithmetic)
% Problem  : 3 !< 2
% Version  : Especial.
% English  :

% Refs     :
% Source   : [TPTP]
% Names    :

% Status   : Theorem
```

```
% Rating   : 0.37 v3.3.0, 0.36 v3.2.0, 0.45 v3.1.0
% Syntax   : Number of formulae    :  402 ( 375 unit)
%            Number of atoms       :  473 (   5 equality)
%            Maximal formula depth :   19 (   1 average)
%            Number of connectives :   74 (   3 ~  ;   1  |;
43  &)
%                                       (   3 <=>;  24 =>;
0 <=)
%                                       (   0 <~>;   0 ~|;
0 ~&)
%            Number of predicates  :   11 (   0 propositional;
1-4 arity)
%            Number of functors    :  260 ( 256 constant; 0-2
arity)
%            Number of variables   :  121 (   0 singleton; 121
!;   0 ?)
%            Maximal term depth    :    5 (   2 average)

% Comments :
%------------------------------------------------------------------
--------------
%----Include axioms for RDN arithmetic
include('Axioms/NUM005+0.ax').
include('Axioms/NUM005+1.ax').
include('Axioms/NUM005+2.ax').
%------------------------------------------------------------------
--------------
fof(n3_not_less_n2,conjecture,
    ( ~ less(n3,n2) )).
%------------------------------------------------------------------
--------------
```

Among the axiom files included by this problem file is Axioms/NUM005+2.ax, we display here only a part of it:

```
%------------------------------------------------------------------
--------------
% File     : NUM005+2 : TPTP v3.3.0. Released v3.1.0.
% Domain   : Number Theory
% Axioms   : Sum in RDN format
% Version  : Especial.
% English  : Impements a "human style" addition using RDN
format.

% Refs     :
% Source   : [TPTP]
% Names    :

% Status   :
% Syntax   : Number of formulae    :  115 ( 100 unit)
%            Number of atoms       :  164 (   3 equality)
%            Maximal formula depth :   19 (   2 average)
%            Number of connectives :   49 (   0 ~  ;   0  |;
34  &)
%                                       (   1 <=>;  14 =>;
0 <=)
```

```
%                                          (   0 <~>;   0 ~|;
0 ~&)
%           Number of predicates  :    8 (   0 propositional;
1-4 arity)
%           Number of functors    :   14 (  10 constant; 0-2
arity)
%           Number of variables   :   86 (   0 singleton;  86
!;   0 ?)
%           Maximal term depth    :    3 (   2 average)

% Comments : Requires NUM005+0.ax NUM005+1.ax
%------------------------------------------------------------------
---------------
%----Addition entry points
%----pos(X) + pos(Y)
fof(sum_entry_point_pos_pos,axiom,(
    ! [X,Y,Z,RDN_X,RDN_Y,RDN_Z] :
      ( ( rdn_translate(X,rdn_pos(RDN_X))
        & rdn_translate(Y,rdn_pos(RDN_Y))
        & rdn_add_with_carry(rdnn(n0),RDN_X,RDN_Y,RDN_Z)
        & rdn_translate(Z,rdn_pos(RDN_Z)) )
     => sum(X,Y,Z) )   )).

%----neg(X) + neg(Y)
fof(sum_entry_point_neg_neg,axiom,(
    ! [X,Y,Z,RDN_X,RDN_Y,RDN_Z] :
      ( ( rdn_translate(X,rdn_neg(RDN_X))
        & rdn_translate(Y,rdn_neg(RDN_Y))
        & rdn_add_with_carry(rdnn(n0),RDN_X,RDN_Y,RDN_Z)
        & rdn_translate(Z,rdn_neg(RDN_Z)) )
     => sum(X,Y,Z) )   )).

%----pos(X) + neg(Y), X < Y
fof(sum_entry_point_pos_neg_1,axiom,(
    ! [X,Y,Z,RDN_X,RDN_Y,RDN_Z] :
      ( ( rdn_translate(X,rdn_pos(RDN_X))
        & rdn_translate(Y,rdn_neg(RDN_Y))
        & rdn_positive_less(RDN_X,RDN_Y)
        & rdn_add_with_carry(rdnn(n0),RDN_X,RDN_Z,RDN_Y)
        & rdn_translate(Z,rdn_neg(RDN_Z)) )
     => sum(X,Y,Z) )   )).

%----pos(X) + neg(Y), X > Y
fof(sum_entry_point_pos_neg_2,axiom,(
    ! [X,Y,Z,RDN_X,RDN_Y,RDN_Z] :
      ( ( rdn_translate(X,rdn_pos(RDN_X))
        & rdn_translate(Y,rdn_neg(RDN_Y))
        & rdn_positive_less(RDN_Y,RDN_X)
        & rdn_add_with_carry(rdnn(n0),RDN_Y,RDN_Z,RDN_X)
        & rdn_translate(Z,rdn_pos(RDN_Z)) )
     => sum(X,Y,Z) )   )).

%---- rest of the file omitted ----
```

Below we show a solution example for the problem given before. Solution presentation is system dependent, the one below is given by the E−Setheo system. Other systems may give more details of the found proof.

```
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−
% File       : E−SETHEO−−−csp04−SAT
% Problem    : NUM291+1 : TPTP v3.1.0
% Transform  : add_equality
% Format     : oldtptp
% Command    : bin/run−e−setheo−sat %s %d

% Computer   : art04.cs.miami.edu
% Model      : i686 unknown
% CPU        : Intel(R) Pentium(R) 4 CPU 2.80GHz @ 2793MHz
% Memory     : 1000MB
% OS         : Linux 2.4.22−21mdk−i686−up−4GB
% CPULimit   : 600s

% Result     : Theorem 0.4s
% Output     : Assurance 0.4s
% Verified   :
% Statistics : −

% Comments   :
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−
%−−−−NO SOLUTION OUTPUT BY SYSTEM
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−
%−−−−ORIGINAL SYSTEM OUTPUT
% E−SETHEO csp04 single processor running on host
art04.cs.miami.edu
% (c) 2004 Technische Universität München
%
% running in SAT mode.
%
/home/graph/tptp/TSTP/PreparedTPTP/oldtptp−−−add_equality/NUM/NU
M291+1+eq_rstfp.oldtptp
% copying problem
/home/graph/tptp/TSTP/PreparedTPTP/oldtptp−−−add_equality/NUM/NU
M291+1+eq_rstfp.oldtptp into workdir
% executing cp
/home/graph/tptp/TSTP/PreparedTPTP/oldtptp−−−add_equality/NUM/NU
M291+1+eq_rstfp.oldtptp /tmp/NUM291+1+eq_rstfp.ol_PID950dtptp
% returncode 0
% problem: NUM291+1+eq_rstfp.ol_PID950
% time limit information: 600 total (entering statistics
module).
% problem analysis ...
% testing if first−order ...
% first−order problem
% executing mv /tmp/NUM291+1+eq_rstfp.ol_PID950dtptp
/tmp/NUM291+1+eq_rstfp.ol_PID950.fof
% returncode 0
% executing
```

```
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/eprover
--tptp-format --cnf  --print-saturated=eigEIGa
/tmp/NUM291+1+eq_rstfp.ol_PID950.fof | sed 's/#/%/' >
/tmp/NUM291+1+eq_rstfp.ol_PID950dtptp
% returncode 0
% ...clausified.
% executing /home/graph/tptp/Systems/E-SETHEO---csp04/bin/dctp
-preprocessor -nopresubsume -nopreprocess
/tmp/NUM291+1+eq_rstfp.ol_PID950dtptp >
/tmp/NUM291+1+eq_rstfp.ol_PID950.p2
% returncode 0
% statistics: 407 2 377 481 5 8 11 0 1 4 260 256 0 2 130 9
% time limit information: 600 total (leaving statistics module).
% selecting schedule ...
% schedule selection: problem is non-horn with equality long
(class nel).
% schedule:601 150 300 150 702 150 9 150
% configuring prover ...
% performing schedule ...
% executing cp /tmp/NUM291+1+eq_rstfp.ol_PID950dtptp
/tmp/NUM291+1+eq_rstfp.ol_PID950.601dtptp
% returncode 0
% entering next strategy 601 with resource 150 seconds.
% time limit information: 600 total / 151 strategy (entering
wrapper).
% configuration line
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/dctp
-preprocessor -taut -nosubsume -noeager -resisol -isollimit 2.
% executing mv /tmp/NUM291+1+eq_rstfp.ol_PID950.601dtptp
/tmp/NUM291+1+eq_rstfp.ol_PID950.601.p2;
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/limit_cpu 151
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/dctp
-preprocessor -taut -nosubsume -noeager -resisol -isollimit 2
-fullrewrite -resisol /tmp/NUM291+1+eq_rstfp.ol_PID950.601.p2 >
/tmp/NUM291+1+eq_rstfp.ol_PID950.601dtptp
% returncode 0
% analyzing results ...
% time limit information: 599 total / 150 strategy (leaving
wrapper).
% time limit information: 599 total / 150 strategy (entering
wrapper).
% configuration line
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/scheme-setheo
-cons -mate -foldup -dynsgreord 0 -sgindex -indexfoldup .
% executing
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/limit_cpu 150
/home/graph/tptp/Systems/E-SETHEO---csp04/bin/scheme-setheo
-cons -mate -foldup -dynsgreord 0 -sgindex -indexfoldup
/tmp/NUM291+1+eq_rstfp.ol_PID950.601dtptp >
/tmp/NUM291+1+eq_rstfp.ol_PID950.601.log 2>>
/tmp/NUM291+1+eq_rstfp.ol_PID950.601.wlog
% returncode 0
% analyzing results ...
% proof found
% time limit information: 599 total / 150 strategy (leaving
wrapper).
% task NUM291+1+eq_rstfp.ol_PID950 on art04.cs.miami.edu has
```

```
status SUCCESS (proof found by strategy 601) consuming 1 seconds
% deleting temporary files.
% e-SETHEO done. exiting
%
%----------------------------------------------------------------
---------------
```

The generator files have three header sections, similar with the ones of problem and axiom files, where the information is replaced with place−holders. The place−holders are filled when the generator is executed. The generator programs are written in Prolog.

38 Yacas

38.1 Short Description

As described on the system's website: "Yacas (Yet Another Computer Algebra System) is a small and highly flexible general−purpose computer algebra system and programming lan−guage. The language has a familiar, C−like infix−operator syntax. The distribution contains a small library of mathematical functions, but its real strength is in the language in which you can easily write your own symbolic manipulation algorithms. The core engine supports arbitrary precision arithmetic, and is able to execute symbolic manipulations on various mathematical objects by following user−defined rules. "

38.2 Technical Information on the System

38.2.1 Name of the System and Website

Yacas: Yet Another Computer Algebra System. http://yacas.sourceforge.net/homepage.html

38.2.2 Project Leaders and Group

The project has been started by Ayal Pinkus, who is also the main author and primary maintainer of the system. Many contributions were made by various people. For a complete list see: http://yacas.sourceforge.net/homepage.html?recent.html&credits.html (last checked 6.12.2007).

38.2.3 Main Publications

Ayal Z. Pinkus, Serge Winitzki. YACAS: A Do−It−Yourself Symbolic Algebra Environ−ment. In *Proceedings of the Joint International Conferences on Artificial Intelligence, Auto−mated Reasoning, and Symbolic Computation*, LNCS, Vol. 2385. pp. 332 − 336, ISBN:3−540−43865−3, 2002, Springer−Verlag.

38.2.4 Implementation Language

C++ and an own scripting language, based on functional language ideas. The script library contains declarations of transformation rules and of function syntax.

38.2.5 System Availability and Prerequisites

The system is free, distributed under GNU GPL. Runs on many platforms and operating systems, including Unix flavors (GNU/Linux and derivatives), Mac OS X, and 32−bit Microsoft Windows (TM). If the sources are downloaded, a C++ compiler is needed.

38.3 Algorithm Libraries

38.3.1 Numerical Library

Euclidean GCD algorithms, Prime numbers: the Miller−Rabin test and its improvements, Factorization of integers, The Jacobi symbol, Integer partitions, Gaussian integers.

Modular arithmetic, Factoring using modular arithmetic, Preparing the polynomial for factorization, Definition of division of polynomials, Determining possible factors modulo 2, Determining factors modulo 2^n given a factorization modulo 2, Efficiently deciding if a polynomial divides another, Newton iteration.

Basic methods in Numerical algorithms: Adaptive function plotting, Surface plotting, Parametric plots, The cost of arbitrary−precision computations, Estimating convergence of a series, Estimating the round−off error, Basic arbitrary−precision arithmetic, Continued fractions, Estimating convergence of continued fractions, Newton's method and its improvements, Fast evaluation of Taylor series, Asymptotic series for calculations, The AGM sequence algorithms, The binary splitting method

Elementary functions in numerical algorithms: Powers, Roots, Logarithm, Exponential, Calculation of Pi, Trigonometric functions, Inverse trigonometric functions, Factorials and binomial coefficients, Classical orthogonal polynomials, Series of orthogonal polynomials.

Special functions in numerical algorithms: Euler's Gamma function, Euler's constant gamma, Catalan's constant G, Riemann's Zeta function, Lambert's W function, Bessel functions, Bernoulli numbers and polynomials, Error function Erf(x) and related functions.

38.3.2 Symbolic Algebra Library

Sparse representations, Implementation of multivariate polynomials, Integration, Transforms, Finding real roots of polynomials.

38.3.3 Reasoners

38.3.4 Graphical Tools and Interfaces

The system can be called from within TeXmacs, a WYSIWYG editor (see http://www.texmacs.org/). A simple Windows graphical interface for the system is also avail−able (GUIYacas).

38.4 User Language

38.4.1 Programming Language

To write new algorithms, they have to be implemented in C++ and the Yacas scripting language.

38.4.2 Logic Language for the Formulation of Mathematical Knowledge

No logic is specified.

38.4.3 Mathematical Syntax

The Yacas syntax: the expressions are built up of words, which are either sequences of alphabetic characters, numbers, brackets, or built up of mathematical symbols like +, −, *, etc. Mixing up these characters can be done using the double quotes.

38.5 Mathematical Knowledge Bases

38.5.1 Available Theories and Knowledge Bases

38.5.2 Tools for Retrieval in Mathematical Knowledge Bases

38.5.3 Tools for Inventing Mathematical Knowledge

38.5.4 Tools for Verifying Mathematical Knowledge (General Reasoners, Special Reasoners)

38.5.5 Tools for Completion, Reduction, Generalization, Structuring, Re−Structuring of Mathematical Knowledge

38.5.6 (Verified) Programming of Reasoners, Meta−Programming, Quotation

38.5.7 Standardization, Inter−Operability

Yacas can handle input and output in OpenMath and ASCII. It has been linked with R − a statistic package − through Ryacas.

38.5.8 Web Access

The system can be used on−line on the project's website, as a Java applet.

38.6 Example of a Theory Exploration Session

Normally, the Yacas system is used in command line modus. There is also a simple GUI available, which works under Windows (needs the .Net framework). We show an example of a Yacas computation session:

```
In> 0+x
Out> x

In> x+1*y
Out> x+y

In> Sin(ArcSin(alpha))+Tan(ArcTan(beta))
Out> alpha+beta


In> A := {{0,−1},{1,0}}
Out> {{0,−1},{1,0}};
In> PrettyForm(%)
```

```
/               \
| ( 0 ) ( −1 ) |
|               |
| ( 1 ) ( 0 )   |
\               /
Out> True;
In> IsSkewSymmetric(A);
Out> True;


In> Expand((1+x)^5);
  5         4          3          2
x   + 5 * x   + 10 * x   + 10 * x   + 5 * x + 1

In> Degree(x^5+x−1);
Out> 5;
In> Degree(a+b*x^3, a);
Out> 1;


In> e := Expand((a+x)^4,x)
Out> x^4+4*a*x^3+(a^2+(2*a)^2+a^2)*x^2+
(a^2*2*a+2*a^3)*x+a^4;
In> Coef(e,a,2)
Out> 6*x^2;
In> Coef(e,a,0 .. 4)
Out> {x^4,4*x^3,6*x^2,4*x,1};
```

39 Conclusions

We have presented in this report a number of mathematical systems which we have analysed from the MKM point of view, namely, we were interested in wich tools for MKM they provide. We tried to make this collection of mathematical software descriptions comprehensive, how−ever, since there's a lot of attention to this subject, there are more and more systems ascending and under development, which makes it impossible for such a report to be complete. We will continuously update this collection of software systems and invite the reader to look at future editions of this technical report for up−to−date system descriptions.

40 References

[ACL2Web] The ACL2 System. Available at http://www.cs.utexas.edu/users/moore/acl2/ (last checked on 12.11.2007).

[AldorWeb] The Aldor Programming Language. http://www.aldor.org/.

[AtlasWeb] Atlas of Finite Group Representations − Version 3. http://brauer.maths.qmul.ac.uk/Atlas/v3/ (last checked on 08.10.2007).

[CONNEXIONS] CONNEXIONS Webpage, http://rhaptos.org/, last checked on 07.11.07.

[C−CoRNWeb] The Constructive Coq Repository at Nijmegen. http://c−corn.cs.ru.nl/index.html (last checked on 16.10.2007)

[Dimacs] DIMACS. Satisfiability Suggested Format.
ftp://dimacs.rutgers.edu/ pub/challenge/satisfiability/doc/satformat.tex (last checked on 04.11.2007).

[FunctionsSite] The Functions Site. http://functions.wolfram.com.

[HolWeb] The HOL 4 system. http://hol.sourceforge.net/.

[HolLightWeb] The Hol Light System. Written by John Harrison. Available at http://www.cl.cam.ac.uk/~jrh13/hol−light/

[JFM] Journal of Formalized Mathematics, http://www.mizar.org/JFM/ (as of 12.12.07).

[LogiCalWeb] The Logical Project. http://logical.inria.fr/ (last checked on 10.10.2007)

[Lucene] Lucene. The Lucene search engine, 2007. http://jakarta.apache.org/lucene/.

[MagmaWeb] MAGMA Computational Algebra System. Available at http://magma.maths.usyd.edu.au/magma/ (last checked on 25.10.2007).

[MapleWeb] The Maple Computer Algebra System. Available from http://www.maplesoft.com/

[MathBus] The MathBus Term Structure. http://www.cs.cornell.edu/Info/Projects/NuPrl/mathbus/mathbusTOC.htm (last checked on 21.11.2007).

[Matita] The Matita Theorem Prover. http://matita.cs.unibo.it/index.shtml

[Maxima] Maxima − A GPL CAS based on doe−macsyma. http://maxima.sourceforge.net

[Metaprl] Metaprl home page. http://metaprl.org.

[MOWGLI] MOWGLI (Mathematics On the Web: Get it by Logic and Interfaces), http://mowgli.cs.unibo.it/

[mOZart] The Mozart Programming System. http://www.mozart−oz.org/

[NasaLangley] The NASA Langley PVS Lbraries. Available on−line at http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS−library/pvslib.html (last checked on 23.11.2007)

[OCaml] The Objective Caml Language. http://caml.inria.fr/ocaml/index.en.html (last checked on 09.10.2007)

[PVSWeb] The PVS home page. http://pvs.csl.sri.com.

[QEDMan] The QED Manifesto. In Automated Deduction − CADE 12, Springer−Verlag, Lecture Notes in Artificial Intelligence, Vol. 814, pp. 238−251, 1994.

[QMath] QMath, http://www.matracas.net/qmath/index.en.html (last checked on 26.11.07)

[Yacas] The Yacas computer algebra system. http://yacas.sourceforge.net/homepage.html

[WIRIS] WIRIS CAS. http://www.wiris.com/

[MathWorld] Wolfram's MathWorld. http://mathworld.wolfram.com/

[Abbott06] J. Abbott. Challenges in Computational Commutative Algebra. In W. Decker, M. Dewar, E. Kaltofen, and S. Watt, editors, *Challenges in Symbolic Computation Software*, Internationales Begegnungs− und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[AbramowitzStegun65]M. Abramowitz and I.A. Stegun, Handbook of mathematical functions (with formulas, graphs, and mathematical tables), Dover: NewYork, 1965.

[MaplePVS] Andrew Adams, Martin Dunstan, Hanne Gottliebsen, Tom Kelsey, Ursula Martin, Sam Owre. Computer Algebra Meets Automated Theorem Proving: Integrating Maple and PVS. In proceedings of Theorem Proving in Higher Order Logics 14th International Conference, TPHOLs 2001 Edinburgh, Scotland, UK, September 3−6, 200. LNCS 2152/2001, pp. 27−42, ISBN 978−3−540−42525−0, Springer Berlin−Heidelberg.

[Agerholm94] S. Agerholm. Formalising a model of the lambda−calculus in HOL−st. Technical Report 354, University of Cambridge Computer Laboratory, 1994.

[AllenEtAl02] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. *FDL: A Prototype Formal Digital Library − Description and Draft Reference Manual.* Technical report, 16 June 2004. Department of Computer Science, Cornell−University, Ithaca, NY, 14853−7501.

[AndersonLebiere98] J.R Anderson and C. Lebiere, The Atomic Components of Thought, Lawrence Erlbaum, 1998.

[Anghelache04] R.Anghelache, Hermes − A Reliable Conversion from TeX to MathML. In *Proceedings of New Developments in Electronic Publishing of Mathematics* (4−th European Congress of Mathematics), 2004.

[ArmandoZini00] A. Armando and D. Zini. Interfacing Computer Algebra and Deduction Systems via the Logic Broker Architecture. In *Proc. of the 8th Calculemus Symposium*, 2000. A.K.Peters.

[Asperti07] A. Asperti, Matita Tutorial, August 2007. http://matita.cs.unibo.it/FILES/matita−tut.pdf

[AspertiEtAl07] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, Stefano Zacchiroli. User Interaction with the Matita Proof Assistant, Journal of Automated Reasoning, Special Issue on User Interfaces for Theorem Proving, Volume 39 , Issue 2 , August 2007, pages: 109 − 139.

[AspertiEtAl07a] A.Asperti, C.Sacerdoti Coen, E.Tassi, S.Zacchiroli. Crafting a Proof Assistant. Proceedings of Types 2006: Types for Proofs and Programs. Nottingham, UK — April 18−21, 2006. LNCS 4502, Springer, ISBN 978−3−540−74463−4, pp. 18−32, 2007.

[AspertiEtAl06] A.Asperti, H.Geuvers, I.Loeb, L.E.Mamane, C.Sacerdoti Coen. An Interactive Algebra Course with Formalised Proofs and Definitions. In *Fourth International Conference on Mathematical Knowledge Management* (MKM2005), Lecture Notes in Artificial Intelligence (LNAI), Vol. 3863, Pages 315—329, 2006.

[AspertiEtAl04] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A Content Based Mathematical Search Engine: Whelp. *Proceedings of TYPES 2004*, Ed. C. Paulin−Mohring, and B. Werner, LNCS 3839, pages 17—32, Springer Verlag .

[AspertiEtAl00a] A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. Towards a Library of Formal Mathematics. Technical Report of TPHOLS2000 Conference, Portland, Oregon, USA, August 2000.

[AspertiEtAl00] Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen and Irene Schena. An Hypertextual Electronic Library of Mathematics. talk at INRIA, Rocquencourt, on 05/06/2000.

[AspertiSelmi04] Andrea Asperti, and Matteo Selmi, Efficient Retrieval of Mathematical Statements, *Proceedings of MKM04*, Bialowieza, Poland, Ed. A. Asperti, G. Bancerek and A. Trybulec, LNCS 3119, pages 17—32, Springer Verlag.

[AspertiTassi07] A. Asperti, E. Tassi. Higher Order Proof Reconstruction from Paramodulation−based Refutations: The Unit Equality Case. In *Towards Mechanized Mathematical Assistants* (Proceedings of MKM and Calculemus 2007), LNCS 4573, p. 146−160, Springer Verlag.

[Aspinall00] David Aspinall. Proof General: A generic tool for proof development. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1785 of Lecture Notes in Computer Science, pages 38−42. Springer−Verlag, 2000.

[Astrachan94] Astrachan, O.L.: METEOR: Exploring Model Elimination Theorem Proving, Journal of Automated Reasoning 13(3), 283−296, 1994.

[MathML] Ron Ausbrooks et. al., Mathematical Markup Language (MathML) Version 2.0 (Second Edition), W3C Recommendation 21 October 2003.

[AutexierEtAl02] S. Autexier and D. Hutter and T. Mossakowski and A. Schairer. The development graph manager MAYA. *Proceedings 9th International Conference on Algebraic Methodology And Software Technology* (AMAST'02), Vol.2422. Springer, 2002.

[BallarinEtAl95] Clemens Ballarin, Karsten Homann, Jacques Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In A.H.M. Levelt (editor) Proceedings of International Symposium on Symbolic and Algebraic Computation, ISSAC'95, Montreal, Canada, ACM Press, pp150−157, 1995.

[BancerekRudnicki03] G. Bancerek, and P. Rudnicki, Information Retrieval in MML, In Eds. A. Asperti, B. Buchberger and J. Davenport, Proceedings of the Second International Conference on Mathematical Knowledge Management, Springer, 2003, 2594, 119−132.

[Barzilay06] Eli Barzilay. Implementing Reflection in Nuprl. PhD Thesis, Cornell University, January 2006. Available at: http://www.cs.cornell.edu/info/projects/nuprl/documents/Barzilay/thesis05.html.

[BauerEtAl02] C. Bauer, A. Frink, and R. Kreckel. Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language. *J. Symbolic Computation* (2002) 33, 1−12.

[BaumgartnerFurbach94] Baumgartner, P., and Furbach, U.: PROTEIN: A PROver with a Theory Extension INterface, Proceedings of the 12th International Conference on Automated Deduction, Springer−Verlag, 769−773, Eds: Bundy, A., 1994

[BeckertPosegga95] Beckert, B., and Posegga, J.: leanTAP: Lean, Tableau−based Deduction, Journal of Automated Reasoning 15(3), 339−358, 1995.

[BertotCasteran04] Y. Bertot and P. Castèran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, ISBN 3−540−20854−2, Springer, 2004.

[BertotKahnThery94] Y. Bertot, G. Kahn, and L. Théry. Proof by Pointing, in *TACS '94: Proceedings of the International Conference on Theoretical Aspects of Computer Software*, Springer−Verlag, 141-160, 1994.

[BescheEickOBrien99] H. U. Besche and B. Eick and E.A. O'Brien. A Millennium Project: Constructing Small Groups. *Internat. J. Algebra Comput.* 12, 623 − 644 (2002)

[BlackWindley98] P. E. Black and P. J. Windley. Formal verification of secure programs in the presence of side effects. In Jr. R. H. Sprague, editor, Proceedings of the Thirty−first Hawai'i International Conference on System Sciences (HICSS−31), volume III, pages 327−334. IEEE Computer Science Press, 1998.

[BosmaEtAl97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language., J. Symbolic Comput. 24(3−4), 235?265, 1997.

[BowenGordon95] J. P. Bowen and M. J. C. Gordon. A shallow embedding of Z in HOL. Information and Software Technology, 37(5−6):269−276, May/June 1995.

[BrownSutcliffe00] M. Brown and G. Sutcliffe: PTTP+GLiDeS − Using Models to Guide Linear Deductions, Proceedings of the CADE−17 Workshop: Model Computation − Principles, Algorithms, Applications, , Eds: P. Baumgartner, C. Fermueller, N. Peltier, and H. Zhang, 2000

[Bruijn87] N.G. de Bruijn, The mathematical vernacular, a language for mathematics with typed sets. In P. Dybjer et al. (eds.), Proceedings of the workshop on Programming Languages, 1987, Marstrand, Sweden

[Buchberger07] Bruno Buchberger. Math−Agents: Mathematical Journals as Active Reasoning Agents. Proposal Draft. Research Institute for Symbolic Computation, Johannes Kepler University, Linz, A4232 Schloss Hagenberg, Austria, 2007.

[Buchberger06] B. Buchberger. Mathematical Theory Exploration. August 17−20, 2006. Invited talk at IJCAR, Seattle, USA.

[Buchberger03] B. Buchberger. Groebner Rings in Theorema: A Case Study in Functors and Categories SFB Technical Report 2003−49, SFB "Scientific Computing", Mathematical Institutes, Johannes Kepler University, Linz, Austria.

[Buchberger01] The PCS Prover in Theorema. B. Buchberger. In: Proceedings of EUROCAST 2001 (8th International Conference on Computer Aided Systems Theory − Formal Methods and Tools for Computer Science), R. Moreno−Diaz, B. Buchberger, J.L. Freire (ed.), Lecture Notes in Computer Science 2178 , pp. 469−478. 19−23 February 2001. Copyright: Springer − Verlag Berlin, Las Palmas de Gran Canaria, ISSN 0302−9743, ISBN 3−540−429.

[Buchberger99] B. Buchberger. Theorem Proving Versus Theory Exploration. Calculemus Workshop, IRST, Univ. of Trento, Italy, November 7, 1999.

[Buchberger98] B. Buchberger. Theorema: Theorem Proving for the Masses Using Mathematica. June 20, 1998. Invited talk at Worldwide Mathematica Conference, Chicago, USA, (Available as SFB Report No. 98−04, Johannes Kepler University Linz, Spezialforschungsbereich F013, June, 1998, 27 pages.).

[Buchberger96] B. Buchberger. Mathematica as a Rewrite Language. In: T. Ida, A. Ohori, M. Takeichi (eds.), Functional and Logic Programming (Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming), November 1−4, 1996, Shonan Village Center), World Scientific, Singapore − New Jersey − London − Hongkong, 1996, pp. 1−13.

[Buchberger85] B. Buchberger. Symbolic Computation. Editorial for the Journal of Symbolic Computation, Vol. 1/1, pp. 3−9. 1985.

[BuchbergerCaprotti01] Bruno Buchberger, Olga Caprotti. MKM'01. First International Workshop on Mathematical Knowledge Management. Technical report no. 01−31 in RISC Report Series, University of Linz, Austria. September 2001.

[BuchbergerCraciun03] B. Buchberger, A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. In: Electronic Notes in Theoretical Computer Science, Fairouz Kamareddine (ed.) 93, pp. 24−59. 18 February 2004. ISBN 044451290X. Proc. of the Mathematical Knowledge Management Workshop, Edinburgh, Nov. 25, 2003.

[BuchbergerEtAl06] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, W. Windsteiger. Theorema: Towards Computer−Aided Mathematical Theory Exploration. Journal of Applied Logic 4(4), pp. 470−504. 2006. ISSN 1570−8683.

[BuchbergerEtAl00] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning), M. Kerber, M. Kohlhase (ed.), pp. 98−113. 6−7 August 2000. Copyright: A.K. Peters, Natick, Massachusetts, St. Andrews, Scotland, ISBN 1−56881−145−4.

[BuchbergerEtAl97] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, D. Vasaru. A Survey of the Theorema project. In: Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21−23, 1997), W. Kuechlin (ed.), pp. 384−391. 1997. ACM Press, ISBN 0−89791−875−4.

[BundyEtAl98] A. Bundy, S. Colton and T. Walsh: HR − A System for Machine Discovery in Finite Algebras. Proceedings of the Machine Discovery Workshop, ECAI−98, Brighton, England, 1998.

[Bundy91] A. Bundy, A Science of Reasoning. In *Computational Logic: Essays in Honor of Alan Robinson*, J.−L. Lassez J and G.Plotkin (eds), MIT Press, pp 178−198, 1991.

[Cairns04] P. Cairns, Alcor: A user interface for Mizar. MKM UI Workshop at 3rd Int. Conf on Mathematical Knowledge Management, 2004.

[CaldwellCowles02] Caldwell, J. L. and Cowles, J. R. Representing Nuprl Proof Objects in ACL2: Toward a proof checker for Nuprl, in *Third International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2−2002)*, 2002.

[CannonPlayoust96] John Cannon and Catherine Playoust. First Steps in Magma. Booklet available at http://magma.maths.usyd.edu.au/magma/pdf/first.pdf (last checked on 29.10.2007).

[CantOsols99] Cant, A., Osols, M.A. XIsabelle: A graphical user interface to the Isabelle theorem prover. Research Report of the Information Technology Division, DSTO Formal Report DSTO−RR−0070.

[Capani00] Antonio Capani. The Design of the CoCoA 3 System. PhD Thesis, DISI−TH−2000−02. Universita degli Studi di Genova, Dipartamento di Informatica e Scienze dell'Informazione, February 2000.

[CarlsonEtAl99] A.J. Carlson, C.M. Cumby, J.L. Rosen and D. Roth, SNoW User's Guide. UIUC Tech Report UIUC−DCS−R−99−210.

[McCaslandetal06] Roy McCasland, Alan Bundy, and Patrick Smith, Ascertaining Mathematical Theorems, Electronic Notes in Theoretical Computer Science, Volume 151 (1), 2006, pp21 − 38. ©Elsevier.

[McCaslandBundy06] Roy McCasland and Alan Bundy, MATHsAiD : a Mathematical Theorem Discovery Tool, Proceedings of SYNASC' 06, pp17 − 22, IEEE Computer Society Press, 2006.

[McCaslandetal07] Roy McCasland, Alan Bundy, and Serge Autexier. Automated discovery of inductive theorems, In From Insight to Proof: Festschrift in Honor of A.Trybulec. R.Matuszewski and P.Rudnicki, editors, University of Bialystok, Bialystok, 2007.

[CheikhrouhouSorge00] L. Cheikhrouhou and V. Sorge. PDS −−− A Three−Dimensional Data Structure for Proof Plans. In Proc. of ACIDCA'2000, 2000.

[Chu94] H. Chu: CLIN−S User's Manual, Department of Computer Science, University of North Carolina at Chapel Hill, 1994.

[CoCoAMan] The CoCoA Team. CoCoA 4.7 Manual. Available at http://cocoa.dima.unige.it/download/doc/help.pdf (last checked

[CoCoAWeb] The CoCoATeam. CoCoA: A System for doing Computations in Commutative Algebra. Available at http://cocoa.dima.unige.it/ (last checked on 17.10.2007).

[Coen04] C. S. Coen, Mathematical Knowledge Management and Interactive Theorem Proving, PhD. thesis, 2004, Department of Computer Science, University of Bologna.

[CoQWeb] The Coq Development Team. *The Coq Proof Assistant*. http://coq.inria.fr/ (last checked on 09.10.2007)

[CoQMan] The Coq Development Team. *The Coq Reference Manual, Version 8.1.* 2006 http://coq.inria.fr/doc/toc.html (last checked on 10.10.2007)

[Colton02] Simon Colton. The HR Program for Theorem Generation. Automated Deduction − CADE−18 : 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27−30, 2002. Proceedings, 2392, pages 285−290.

[ColtonEtAl00]Simon Colton, Alan Bundy and Toby Walsh. On the Notion of Interestingness in Automated Mathematical Discovery IJHCS: International Journal of Human−Computer Studies, Academic Press, 2000.

[ColtonMaple02] Simon Colton, Making Conjectures about Maple Functions. In proceedings of AISC/Calculemus'02, LNAI 2385, Springer.

[CraciunHodorog07]A. Craciun, M. Hodorog. Decompositions of Natural Numbers: From A Case Study in Mathematical Theory Exploration. In: Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC07), D. Petcu, V. Negru, D. Zaharie and T. Jebelean (ed.), pp. 1−8. 26−29 September 2007. West University of Timisoara, Romania, −. to appear as IEEE volume.

[CrowEtAl01] Judy Crow, Sam Owre, John Rushby, N. Shankar, and Dave Stringer−Calvert. Evaluating, Testing, and Animating PVS Specifications. CSL Technical Report, March 30, 200.

[CruzFilipe04] L. Cruz−Filipe. Constructive Real Analysis: a Type−Theoretical Formalization and Applications, PhD Thesis. University of Nijmegen, April 2004.

[McCune−EQP] W.W. McCune. EQP: Equational Prover, http://www−unix.mcs.anl.gov/AR/eqp/ (last checked on 04.11.2007)

[McCune94] W.W. McCune. Otter 3.0 Reference Manual and Guide, Argonne National Laboratory No. ANL−94/6, 1994.

[McCune03] McCune, W.W.: Mace4 Reference Manual and Guide, Argonne National Laboratory No. ANL/MCS−TM−264, 2003.

[DahnEtAl95] I.B. Dahn, J. Gehne, T. Honigmann, L. Walther, and A. Wolf: Integrating Logical Functions with ILF, Institut für Mathematik, Humboldt Universität zu Berlin, 1995

[Delahaye00] D. Delahaye. A Tactic Language for the System Coq. In *Proceedings of Logic for Programming and Automated Reasoning (LPAR)*, Reunion Island, volume 1955 of Lecture Notes in Computer Science, pages 85−95. Springer−Verlag, November 2000.

[EisenbudEtAl00] D. Eisenbud, D.R. Grayson, M. Stillman, and B. Sturmfels, editors. *Computations in Algebraic Geometry with Macaulay 2*, Algorithms and Computation in Mathematics Series, Volume 8, ISBN 978−3−540−42230−3. Springer, 2000.

[EngelmoreMorgan88] R. Engelmore and T. Morgan (Eds.), Blackboard Systems, Addison Wesley, 1988.

[Fiedler01] A. Fiedler, P.Rex: An interactive proof explainer, In Automated Reasoning — 1st International Joint Conference, (eds.) Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, IJCAR 2001, LNAI 2083, pp. 416−420, Siena, Italy, Springer Verlag, 2001

[Fox05] Anthony Fox. An algebraic framework for verifying the correctness of hardware with input and output: A formalization in HOL. In J. L. Fiadeiro et al., editor. CALCO 2005, volume 3629 of Lecture Notes in Computer Science. Springer, 2005, pages 157−174.

[FrankeKohlhase99] A. Franke and M. Kohlhase, System Description: MathWeb, an Agent−Based Communication Layer for Distributed Automated Theorem Proving. In *Proceedings of CADE'99*, Harald Ganzinger (ed.), LNCS 1632, Springer, 1999.

[FujitaEtAl92] Fujita, M., Hasegawa, R., Koshimura, M., and Fujita, H.: Model Generation Theorem Provers on a Parallel Inference Machine, Proceedings of the International Conference on Fifth Generation Computer Systems, 357−375, 1992.

[GAP4Man] The GAP Manual, http://www.gap−system.org/~gap/Doc/manuals.html (last checked on 08.10.2007)

[GAPWeb] The GAP Website, http://www.gap−system.org (last checked on 08.10.2007)

[GeislerEtAl97] Geisler, T., Panne, S., and Schütz, H.: Satchmo: The Compiling and Functional Variants, Journal of Automated Reasoning 18(2), 227?236, 1997.

[GeneserethFikes92] M.R. Genesereth, and R.E. Fikes: Knowledge Interchange Format, Version 3.0 Reference Manual, Computer Science Department, Stanford University No. Logic−92−1, 1992.

[GierzEtAl80] G.Gierz, K.H.Hofmann, K.Keimel, J.D.Lawson, M.Mislove, and D.S.Scott, *A Compendium of Continuous Lattices*, Springer−Verlag,1980.

[GieseBuchberger07] Martin Giese, Bruno Buchberger. Towards Practical Reflection for Formal Mathematics. Technical report no. 07−05 in RISC Report Series, University of Linz, Austria. 2007.

[GoguadzeEtAl] G. Goguadze, E. Melis, A.Asperti, MOWGLI Report D1.b. Structure and Metastructure of Mathematical Documents.

http://mowgli.cs.unibo.it/html_yes_frames/deliverables/requirement−analysis/d1b.html

[GordonEtAl79] M.J.C. Gordon, R. Milner ad C.P. Wadsworth, Edinburgh LCF: A Mechanized Logic of Computation, LNCS 78, 1979.

[GordonEtAl06] Mike Gordon, Warren A. Hunt, Jr., Matt Kaufmann, James Reynolds. An Integration of HOL and ACL2. Formal Methods in Computer Aided Design (FMCAD'06) pp. 153−160.

[GordonMelham93] M. J. C. Gordon and T. F. Melham, editors. Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press, 1993.

[Graf94] Peter Graf, Substitution Tree Indexing, 1994MPI−I−94−251, Saarbruecken. http://citeseer.ist.psu.edu/graf94substitution.html (as of 12.12.07)

[Graf96] Peter Graf, Term indexing, LNCS 1053, Springer Verlag, 1996.

[GuidiCoen03] F. Guidi, and C. S. Coen, Querying Distributed Digital Libraries of Mathematics, 2003, *Proceedings of Calculemus 2003*, Ed. Therese Hardin e Renaud Rioboo, ISBN 88−7999−545−6, pages 17—30, Aracne Editrice S.R.L.

[Macaulay2] Daniel R. Grayson and Michael E. Stillman. Macaulay 2, A Software System for Research in Algebraic Geometry. Available at http://www.math.uiuc.edu/Macaulay2/

[Plural] G.−M. Greuel, V. Levandovskyy, and H. Schönemann. Singular::Plural 2.1. A Computer Algebra System for Noncommutative

Polynomial Algebras. Centre for Computer Algebra, University of Kaiserslautern (2003). Available at http://www.singular.uni−kl.de/plural (last checked on 23.10.2007)

[GreuelPfister06] G.−M. Greuel, G. Pfister. SINGULAR and Applications. Jahresbericht der DMV 108 (2006), 167−196.

[Singular] G.−M. Greuel, G. Pfister, and H. Schönemann. Singular 3.0, A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern (2006). Available at http://www.singular.uni−kl.de (last checked on 23.10.2007)

[Grundy91] J. Grundy. Window Inference In The HOL System. HOL Theorem Proving System and Its Applications, 1991., International Workshop on the Volume , Issue , 28−30 Aug 1991 Page(s):177 − 189.

[HähnleEtAl94] Hähnle, R., Beckert, B., and Gerberding, S.: The Many−Valued Tableau−Based Theorem Prover 3TAP, Fakultät für Informatik, Universät Karlsruhe No. TR 30/94, 1994

[HähnleEtAl96] R. Hähnle, M. Kerber, and C. Weidenbach: Common Syntax of the DFG−Schwerpunktprogramm Deduction, Fakultät für Informatik, Universät Karlsruhe No. TR 10/96, 1996

[Halvorsen07] Jonas Halvorsen. Web Based GUI for Natural Deduction Proofs in Isabelle. Master thesis, School of Informatics, University of Edinburgh, 2007.

[Carine] Paul Haroun. The Automated Theorem Proving System CARINE. Available at http://www.atpcarine.com/ (last checked on 30.10.2007.

[Harrison96] J. Harrison. Theorem Proving with the Real Numbers. PhD thesis, University of Cambridge. Published as Technical Report 408 by the University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, England.

[Harrison96a] J. Harrison. A Mizar Mode for HOL. Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics, TPHOLs'96, Springer LNCS 1125, pp. 203−220, 1996.

[HarrisonThery93] J. Harrison and L. Thery. Extending the HOL theorem prover with a computer algebra system to reason about the reals. In J.J. Joyce and C−J.H. Seger, editors, Proceedings of HOL'93, volume 780 of Lecture Notes in Computer Science, pages 174−−184. Springer−Verlag, 1993. http://citeseer.ist.psu.edu/harrison93extending.html

[Heath56] Thomas L. Heath. *The 13 Books of Euclid's Elements*. Dover, 1956. In 3 volumes. Originally published in 1908.

[HillenbrandEtAl96] T. Hillenbrand, A. Buch, and R. Fettig: On Gaining Efficiency in Completion−Based Theorem Proving, Proceedings of the 7th International Conference on Rewriting Techniques and Applications, Springer−Verlag, 432−435, Eds: H. Ganzinger, 1996.

[HodgsonSlaney00] Kahlil Hodgson and John Slaney. System Description: SCOTT−5. Proceedings of the First International Joint Conference on Automated Reasoning. pages 443−446, 2000.

[HodorogCraciun07] M. Hodorog, A. Craciun. A Case Study in Systematic Theory Exploration: Natural Numbers. October 2007. RISC, University of Linz, Austria. Technical Report.

[Huet92] G.Huet. The Gallina specification language: A case study, *Foundations of Software Technology and Theoretical Computer Science*, Springer Berlin / Heidelberg, 229−240, 1992

[CoQTut] Gérard Huet, Gilles Kahn, and Christine Paulin−Mohring. *The Coq Proof Assistant. A Tutorial*. Version 1. http://coq.inria.fr/V8.1/tutorial.html (last checked on 12.10.2007).

[Jackson95] Paul B. Jackson. "Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra," Ph.D. thesis, Cornell University, TR95−1905. 1995.

[Jackson94] Paul B. Jackson. The Nuprl Proof Development System, Version 4.1 Introductory Tutorial. Unpublished manuscript, Cornell University, 1994. Available at http://www.cs.cornell.edu/Info/Projects/NuPrl/tutorial/tutorial.ps (last checked on 20.11.2007).

[Jackson94a] Paul B. Jackson. Exploring Abstract Algebra in Constructive Type Theory. In A. Bundy, editor, 12th International Conference on Automated Deduction, Lecture Notes in Artifical Intelligence. Springer−Verlag, June 1994.

[Kaliszyk06] Cezary Kaliszyk. Web Interfaces for Proof Assistants, In S. Autexiere and C. Benzmüller, editors, Proceedings of UITP'06.

[KamareddineEtAl07a] F. Kamareddine, M. Maarek, K. Retel and J.B Wells, Gradual Computerisation/Formalisation of Mathematical Texts into Mizar, In *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, *Studies in Logic, Grammar and Rhetoric*, R. Matuszewski and A. Zalewska (ed.), 10(23), pages 95—120 University of Bialystok, 2007.

[KamareddineEtAl07b] Fairouz Kamareddine, Robert Lamar, Manuel Maarek and J. B. Wells. Restoring Natural Language as a Computerised Mathematics Input Method. MKM 2007, Sixth International Conference on Mathematical Knowledge Management. In the Calculemus 2007 / MKM 2007 joint proceedings, LNAI 4573, Manuel Kauers, Manfred Kerber, Robert Miner and Wolfgang Windsteiger (Eds), Springer−Verlag, 2007.

[KamareddineEtAl07c] Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel and J. B. Wells. Narrative Structure of Mathematical Texts. MKM 2007, Sixth International Conference on Mathematical Knowledge Management. In the Calculemus 2007 / MKM 2007 joint proceedings, LNAI 4573, Manuel Kauers, Manfred Kerber, Robert Miner and Wolfgang Windsteiger (Eds), Springer−Verlag, 2007.

[KamareddineNederpelt04] F. Kamareddine and R. Nederpelt, A refinement of de Bruijn's formal language of mathematics. *Journal of Logic, Language and Information* Volume 13, issue 3, pages 287—340, 2004, Kluwer Academic Publishers, ISSN 0925−8531.

[KamareddineWells07] Fairouz Kamareddine and J.B. Wells, Computerising mathematical texts in MathLang, *Second Workshop on Logical and Semantic Frameworks, with Applications*, Ouro Preto, Minas Gerais, Brazil, 28 August 2007. ENTCS, Ayala−Rincon and Heusler (editors). ISSN: 1571−0661, January 2008. Elsevier.

[KanahoriSuzuki06] T.Kanahori, M.Suzuki. Refinement of digitized documents through recognition of mathematical formulae, in Proceedings of the 2nd International Workshop on Document Image Analysis for Libraries, pp.27−28 April 2006, Lyon (France).

[KanahoriEtAl04] T.Kanahori, M.Fujimoto and M.Suzuki. Authoring Tool for Mathematical Documents − Infty −, 3rd International Conference MKM2004, Bialowieja, Poland, 2004, Sept. Online Proceeding.

[KaufmannMoore97] Kaufmann, M. and Moore, J., "An Industrial Strength Theorem Prover for a Logic Based on Common Lisp", IEEE Transactions on Software Engineering, vol. 23, no. 4, pp. 203−213, 1997.

[KaufmannMoore06] Kaufmann, M. and Moore, J. Strother, Maintaining the ACL2 Theorem Proving System, in *Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning*, vol. 192, 2006.

[McKiernan00] McKiernan, Gerry. "arXiv.org: The Los Alamos National Laboratory E−Print Server." The International Journal on Grey Literature 1 (3): 127−138. 2000. http://www.public.iastate.edu/~gerrymck/arXiv.org.pdf

[KleinEtAl07] G. Klein, T. Nipkow, and L. Paulson (editors), The Archive of Formal Proofs, http://afp.sf.net/.

[KleinbergEtAl03] Kleinberg, Jon, Mark Bickford, Robert L. Constable, Richard Eaton, and Christoph Kreitz. A Nuprl−PVS Connection: Integrating Libraries of Formal Mathematics Cornell University Technical Report 2003−1889, 2003.

[Kohlhase06] M. Kohlhase, OMDoc. An Open Markup Format for Mathematical Documents (version 1.2), LNAI 4180, August 2006, Springer−Verlag GmbH. Updated version available online at http://www.omdoc.org/pubs/omdoc1.2.pdf.

[KohlhaseFranke01] M.Kohlhase and H.Franke, MBase:Representing Knowledge and Content for the Integration of Mathematical Software Systems, Journal of Symbolic Computation,2001, 32/4, pages 365 −402.

[KohlhaseSucan06] M. Kohlhase and Sucan, I., A Search Engine for Mathematical Formulae, Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006, Eds.: T.Ida, J.Calmet and D. Wang, Springer Verlag, 2006, pp 241−253.

[Kovacs07] L. Kovacs. Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema. RISC, Johannes Kepler University Linz, Austria. PhD Thesis. October 2007. RISC Technical Report No. 07−16.

[Kreitz02] Christoph Kreitz. The Nuprl Proof Development System, Version 5. Reference Manual and User's Guide. (2002) Available on−line at: http://www.cs.cornell.edu/info/projects/nuprl/html/nuprl5docs.html (last checked on 16.11.2007).

[Kreitz03] Christoph Kreitz. The FDL Navigator: Browsing and Manipulating Formal Content. Unpublished manuscript, Cornell University, 2003. Available on−line at: http://www.cs.cornell.edu/info/projects/nuprl/documents/Kreitz/03fdl−navigator.html(last checked on 21.11.2007).

[KutsiaNakagawa01] Temur Kutsia, Koji Nakagawa. An Interface between Theorema and External Automated Deduction Systems. In: Proceedings of 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'01), Steve Linton and Roberto Sebastiani (ed.), pp. 178−182. Jun 21−23 2001. Siena, Italy.

[Landau51] Edmund Landau. *Foundations of Analysis*. Chelsea, 1951. Translation from the german edition by F. Steinhardt.

[LangbackaEtAl95] T. Langbacka, R. Ruksenas and J. von Wright. TkWinHOL: A tool for doing window inference in HOL. Series A−160, Reports on Computer Science and Mathematics, Abo Akademi University, Finland, 1995.

[Lange07] Christoph Lange: SWiM – A Semantic Wiki for Mathematical Knowledge Management, Proceedings of Mathematical Knowledge Management 2007, Eds. M. Kauers et al. , Workshop MathUI, June 2007.

[LibbrechtGross06] P. Libbrecht, C. Gross, Experience Report Writing LeActiveMath Calculus, Proceedings of Mathematical Knowledge Management 2006, Jon Borwein, William Farmer(eds.), pages 251—265, LNAI 4108, Springer Verlag, August 2006.

[LibbrechtMelis06] Paul Libbrecht, Erica Melis, Methods for Access and Retrieval of Mathematical Content in ActiveMath, Proceedings of ICMS-2006, Ed. Nobuki Takayama, Andres Iglesias, Jaime Gutierrez, LNCS 4151, Springer Verlag GmbH.

[BreuerLinton98] Steve Linton and Thomas Breuer. The GAP 4 Type System: Organising Algebraic Algorithms. In Oliver Gloor, editor, ISSAC98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, pages 38−45. ACM Press,1998.

[LossenSchönem06] C. Lossen, H. Schönemann. 21 Years of Singular Experiments in Mathematics. In: C. Lossen and G. Pfister (eds.), Singularities and Computer Algebra. Lecture Notes of LMS, Cambridge University Press.

[MaekawaEtAl01] M. Maekawa, M. Noro, N. Takayama Y. Tamura and K. Ohara, The Design and Implementation of OpenXM−RFC 100 and 101, In *Computer Mathematics (Proc. ASCM2001)*, World Scientific, 102−111, 2001.

[MeierMelis05] Andreas Meier and Erica Melis, System Description: Multi A Multi−strategy Proof Planner. *Proceedings of the 20th International Conference on Automated Deduction*, LNCS 3632,pages 250−254. Springer Verlag, Tallinn, Estonia, July 22−27, 2005.

[MelisEtAl08], Erica Melis, Andreas Meier and Jörg Siekmann. Proof planning with multiple Strategies. *Artificial Intelligence*, 172/6−7, pages 656−−684, 2008.

[MelisEtAl05] E.Melis, P.Kärger, M.Homik, Interactive Concept Mapping in ActiveMath (iCMap)}, Delfi 2005 : 3. Deutsche eLearning Fachtagung Informatik, LNI, volume 66, Jörg M.Haake, Ulrich Lucke, Djamshid Tavangarian (eds.), September 2005, ISBN 3-88579-395-4, pages 247—258}, Rostock Germany.

[MelisSiekmann04] E. Melis, J. Siekmann, ActiveMath: An Intelligent Tutoring System for Mathematics, Seventh International Conference Artificial Intelligence and Soft Computing (ICAISC), 2004.

[MillerYoussef03] Bruce Miller and Abdou Youssef,Technical Aspects of the Digital Library of Mathematical Functions, Annals of Mathematics and Artificial Intelligence, Volume 38, pp.121−136, 2003.

[NakagawaBuchberger01] K. Nakagawa, B. Buchberger. Presenting Proofs Using Logicographic Symbols. In: Proceedings of the Workshop on Proof Transformation and Presentation at the IJAR−2001, A. Fiedler, H. Horacek (ed.), pp. −. 18 June 2001. Siena.

[NaumovEtAl01] P. Naumov, M.−O. Stehr, and J. Meseguer. The HOL/NuPRL Proof Translator: A Practical Approach to Formal Interoperability. In R.J. Boulton and P.B. Jackson, editors, The 14th International Conference on Theorem Proving in Higher Order Logics, volume 2152 of LNCS, pages 329−345. Springer−Verlag, 2001.

[Naumov98] P. Naumov. Publishing Formal Mathematics on the Web, Technical Report TR98−1689, Cornell University, Computer Science Department, 1998.

[NieuwenhuisEtAl97] R. Nieuwenhuis, J.M. Rivero, and M.A. Vallejo: Dedam: A Kernel of Data Structures and Algorithms for Automated Deduction with Equality Clauses, Proceedings of the 14th International Conference on Automated Deduction, Springer−Verlag, , Eds: W.W. McCune, 1997.

[NipkowEtAl07] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. Isabelle/Hol. A Proof Assistant forHigher−Order Logic, November 2007. Available at http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf

[NieuwenhuisRubio01] R. Nieuwenhuis and A. Rubio, Paramodulation−Based Theorem Proving, Handbook of Automated Reasoning I(7), Elsevier Science and MIT Press, 2001. ISBN 0−444−82949−0.

[deNivelle07] H. de Nivelle. Bliksem resolution prover. Available to download from http://www.ii.uni.wroc.pl/~nivelle/software/bliksem/ (last checked on 30.10.2007).

[HolLogic] Michael Norrish and Konrad Slind. The HOL System. Logic. (For Kananaskis 4). January 2, 2007.

[HolTut] Michael Norrish and Konrad Slind. The HOL System. Tutorial. (For Kananaskis 4). January 2, 2007.

[OpenMath] The OpenMath Society, The OpenMath Standard Version 2.0., Editors: S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaëtano, M. Kohlhase.

[OwreEtAl96] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer−Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 411−414, 1996.

[OwreEtAl92] Owre, S. and Rushby, J.M. and Shankar, and N., *PVS: A Prototype Verification System*, in 11th *International Conference on Automated Deduction* (CADE), vol. 607, 1992, pp. 748−752.

[OwreEtAl98] Sam Owre, John Rushby, N. Shankar and David Stringer−Calvert. *PVS: An Experience Report.* In Applied Formal Methods −− FM−Trends 98, Boppard, Germany, October 1998.

[OwreEtAl99] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer−Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[Paulson07a] Lawrence C. Paulson. *The Isabelle Reference Manual*. 22 November 2007. Available at http://isabelle.in.tum.de/dist/Isabelle/doc/ref.pdf

[Paulson07b] Lawrence C. Paulson. *Isabelle's Logic*. 2007. Available at http://isabelle.in.tum.de/dist/Isabelle/doc/logics.pdf

[Pelletier87] F. J. Pelletier: Further Developments in THINKER, and Automated Theorem Prover, Automated Reasoning Project, Australian National University No. TR−ARP−16/87, 1987

[PiroiBuchberger02] Florina Piroi, Bruno Buchberger. Focus Windows: A New Technique for Proof Presentation. Technical report no. 02−25 in RISC Report Series, University of Linz, Austria. December 2002.

[PiroiBuchberger04] F. Piroi, B. Buchberger. An Environment for Building Mathematical Knowledge Libraries. In: Proceedings of the Workshop on Computer−Supported Mathematical Theory Development, Second International Joint Conference (IJCAR), Wolfgang Windsteiger and Christoph Benzmueller (ed.), pp. 19−29. 4−8 July 2004. Cork, Ireland, ISBN 3−902276−04−5.

[PiroiKutsia05] Florina Piroi, Temur Kutsia. The Theorema Environment for Interactive Proof Development. In: Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings of the 12th International Conference, LPAR'05, G. Sutcliffe, A. Voronkov (ed.), Lecture Notes in Artificial Intelligence 3835, pp. 261−275. 2005. Springer Verlag, ISSN 0302−9743, ISBN 3−540−30553−X.

[PiroiEtAl07] F. Piroi, B. Buchberger, C. Rosenkranz, T. Jebelean. Organisational Tools for MKM in Theorema. Technical report no. 07−11 in RISC Report Series, University of Linz, Austria. 2007.

[Plaisted88] D.A. Plaisted: Non−Horn Clause Logic Programming Without Contrapositives, Journal of Automated Reasoning 4(3), 287−325, 1988

[Pollock90] Pollock, J.L.: Interest Driven Suppositional Reasoning, Journal of Automated Reasoning 6(4), 419−461, 1990.

[NuPRLBook] The PRL Group. Implementing Mathematics with the Nuprl Proof Development System. Computer Science Department, Cornell University. October 1995. Available at: http://www.cs.cornell.edu/info/projects/nuprl/book/doc.html

[RathsEtAl07] Thomas Raths, Jens Otten, Christoph Kreitz. The ILTP Problem Library for Intuitionistic Logic. *Journal of Automated Reasoning*, 38:261−−271, 2007.

[RiazanovVoronkov99] A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger, editor. Proc. of the 16th Int. Conf. on Automated Deduction, CADE'99, volume 1632 of LNAI, Trento, Italy, 1999. Springer, pages 292−296, 1999.

[Rosenkranz07] Camelia Rosenkranz. Retrieval and Structuring of Large Mathematical Knowledge Bases: A Theorema Approach. RISC, University of Linz, Austria. PhD Thesis. 2007. ongoing.

[RudnickiTrybulec01] P. Rudnicki and A.Trybulec, Mathematical Knowledge Management in Mizar, In Proceedings of MKM2001, Eds. B.Buchberger. and O. Caprotti, 2001.

[SchmittEtAl01] Stephan Schmitt, Lori Lorigo, Christoph Kreitz, and Aleksey Nogin. Jprover: Integrating connection−based theorem proving into interactive proof assistants. In *International Joint Conference on Automated Reasoning*, volume 2083 of Lecture Notes in Artificial Intelligence, pages 421−426. Springer−Verlag, 2001.

[SchubertBiggs94] T. Schubert and J. Biggs. A tree−based, graphical interface for large proof development. In *HOL Theorem Proving and Its Applications*, 7th Int. Workshop, 1994.

[Schulz02] S. Schulz. E—a brainiac theorem prover. J. AI Communications, 15(2/3):111−126, 2002.

[SchumannEtAl90] Schumann, J., Trapp, N., and van der Koelen, M.: SETHEO/PARTHEO Users Manual, Institut für Informatik, Technische Universität München No. SFB Bericht 342/7/90 A, 1990

[SiekmannEtAl03] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann and M. Pollet, Proof Development with Ωmega: The irrationality of $\sqrt{2}$, In *Thirty Five Years of Automating Mathematics*, (Ed.) F. Kamareddine, pages 271−314, Kluwer Applied Logic Series, Kluwer Academic Publishers, 2003.

[SiekmannEtAl99] Jorg Siekmann, Stephan Hess, Christoph Benzmuller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet and Volker Sorge. LOUI: Lovely OMEGA User Interface, *Formal Aspects of Computing*, Volume 11, Number 3 / September, 1999, ISSN: 0934−5043

[Slaney92] J.K. Slaney: FINDER (Finite Domain Enumerator): Notes and Guide, Automated Reasoning Project, Australian National University No. TR−ARP−1/92, 1992.

[Stickel84] Stickel, M.E.: A Prolog Technology Theorem Prover, New Generation Computing 2(4), 371−383, 1984.

[SutcliffeSuttner98] Geoff Sutcliffe, and Christian Suttner: The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning* 21(2), 177−203, 1998.

[SutcliffeSuttner99] Geoff Sutcliffe, and Christian Suttner: The TPTP Problem Library, Department of Computer Science, James Cook University, Australia No. 99, February 1999.

[SutcliffeEtAl03] G. Sutcliffe, J. Zimmer, and S. Schulz: Communication Standards for Automated Theorem Proving Tools, Proceedings of the Workshop on Agents and Automated Reasoning, 18th International Joint Conference on Artificial Intelligence, 52?57, Eds: V. Sorge, S. Colton, M. Fisher, and J. Gow, 2003

[SuzukiEtAl05] M.Suzuki, S.Uchida, A.Nomura Ground−Truthed Mathematical Character and Symbol Image Database, Proceedings of 8th International Conference on Document Analysis and Recognition (ICDAR 2005), Seoul, Korea, Vol.2, IEEE Computer Society Press (2005) pp.675−679.

[SuzukiEtAl03] M.Suzuki, F.Tamari, R.Fukuda, S.Uchida, T.Kanahori. Infty− an integrated OCR system for mathematical documents, *Proceedings of ACM Symposium on Document Engineering 2003*, Grenoble, Ed. C.Vanoirbeek, C.Roisin, E. Munson, 2003, pp.95−104

[Syme95] Donald Syme. A New Interface for HOL − Ideas, Issues and Implementation. Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications. Lecture Notes In Computer Science; Vol. 971.Pages: 324 − 339. 1995, ISBN:3−540−60275−5. Springer−Verlag London.

[Tammet97] T. Tammet. Gandalf. J. Automated Reasoning, 18(2):199−204, 1997.

[TheryEtAl92] L. Thery, Y. Bertot, and G. Kahn. Real Theorem Provers Deserve Real User−Interfaces. In *Proceedings of Fifth Symposium on Software Development Environ ments*. ACM, 1992.

[MagmaOverview] University of Sydney Computational Algebra Group: Overview of Magma V2.13 Features, September 2006.

[Urban06] J. Urban, MizarMode − Integrated Proof Assistance Tools for the Mizar Way of Formalizing Mathematics, Journal of Applied Logic 4(4), (Ed.) C. Benzmueller, 2006. ISSN 1570−8683.

[Urban05] J. Urban, MOMM − Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics, In International Journal on Artificial Intelligence Tools, 2005.

[Urban05a] J. Urban, MPTP − Motivation, Implementation, First Experiments. Journal of Automated Reasoning, 2005

[Vollinga05] J. Vollinga. "GiNaC − Symbolic computation with C++". *Nucl.Instrum.Meth.* A559 (2006) 282−284. Also available at arXiv.org as arXiv:hep−ph/0510057 v1. 5 Oct. 2005.

[WagnerEtAl06] Marc Wagner, Serge Autexier, Christoph Benzmüller. PLATO: A Mediator between Text−Editors and Proof Assistance Systems. *7th Workshop on User Interfaces for Theorem Provers* (UITP'06). Elsevier,2006.

[Warner01] Simeon Warner. Open Archives Initiative protocol development and implementation at arXiv. Expanded version of talk presented at Open Archives Initiative Open Meeting in Washington, DC, USA on 23 January 2001. Available on arXiv.org as arXiv:cs/0101027v1.

[Watt03] S.M. Watt. Aldor. pp. 265−270, in *Handbook of Computer Algebra.* J. Grabmeier, E. Kaltofen, V. Weispfenning (editors) , Springer Verlag, Heidelberg 2003 ISBN 3−540−65466−6.

[WeidenbachEtAl99] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topic. System abstract: Spass version 1.0.0. In H. Ganzinger, editor. Proc. of the 16th Int. Conf. on Automated Deduction, CADE'99, volume 1632 of LNAI, Trento, Italy, 1999. Springer, pages 378−382, 1999.

[WenzelBerghofer07] Markus Wenzel and Stefan Berghofer. The Isabelle System Manual. TU München. 2007.

[Wiedijk06] Freek Wiedijk (ed.), foreword by Dana Scott, The Seventeen Provers of the World, Springer LNAI 3600, 2006.

[WiedijkChecker] F. Wiedijk. Checker − notes on the basic inference step in Mizar. Note, last accessed on 10.12.07. http://www.cs.ru.nl/~freek/mizar/by.dvi

[WiedijkMizar] F. Wiedijk. Mizar: an impression, Note, last accessed on 10.12.07. http://www.cs.ru.nl/~freek/mizar/mizarintro.ps.gz

[WiedijkVernacular] F.Wiedijk, The mathematical vernacular, unpublished note. http://www.cs.kun.nl/~freek/notes/mv.ps.gz (last accessed at 12.12.07)

[Wolfram] S. Wolfram. *The Mathematica Book.* Wolfram Media Inc. 5th edition. 2003.

[WongSaunders05] Qiming Wong and Bonita Saunders, Web−Based 3D Visualization in a Digital Library of Mathematical Functions, in Proceedings of Web3D 2005 (10th International Conference on 3D Web Technology), ACM SIGGRAPH, 2005.

[Youssef04] Abdou Youssef, Search Systems for Math Equations, Invited talk at the IMA Workshop on "Special Functions in the Digital Age", University of Minnesota, Minneapolis, MN, July, 2004.

[Youssef07] Abdou Youssef, Methods of Relevance Ranking and Hit−content Generation in Math Search, the 6th International Conference on Mathematical Knowledge Management, pp. 393−406, June 2007, RISC, Hagenberg, Austria.

[Youssef06] Abdou Youssef, Roles of Math Search in Mathematics, pp.2−16, the 5th International Conference on Mathematical Knowledge Management, August 2006, Reading, UK.

[YoussefShatnawi06] Abdou Youssef and Mohammed Shatnawi, Math Search with Equivalence Detection Using Parse−tree Normalization, The 4th International Conference on Computer Science and Information Technology, April 2006, Amman, Jordan.

[Youssef05] Abdou Youssef, Information Search And Retrieval of Mathematical Contents:Issues And Methods. The ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE−2005), July 20−22, 2005, Toronto, Canada.

[ZhangZhang96] Zhang, J., and Zhang, H.: System Description: Generating Models by SEM, Proceedings of the 16th International Conference on Automated Deduction, Springer−Verlag, 308−312, Eds: Ganzinger, H., 1996.

[ZimmerAutexier06] J. Zimmer and S. Autexier. The MathServe Framework for Semantic Reasoning Web Services. *In Proceedings of IJCAR'06*, U. Furbach and N. Shankar (eds.), LNCS 4130, pp. 140−145, Springer, 2006.

[ZimmerEtAl02] Jürgen Zimmer and Andreas Franke and Simon Colton and Geoff Sutcliffe. Integrating HR and tptp2X into MathWeb to Compare Automated Theorem Provers. Proceedings of the CADE'02 Workshop on Problems and Problem sets, Copenhagen, Denmark, 2002.