of Combining Systems (FroCoS'96), Munich, Kluwer Series on Applied Logic, pp. 133–146, 1996.

CLASSIFICATION OF COMMUNICATION AND COOPERATION MECHANISMS FOR LOGICAL AND SYMBOLIC COMPUTATION SYSTEMS

JACQUES CALMET AND KARSTEN HOMANN
Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5 · 76131 Karlsruhe · Germany
{calmet,homann}@ira.uka.de

Abstract. The combination of logical and symbolic computation systems has recently emerged from prototype extensions of stand-alone systems to the study of environments allowing interaction among several systems. Communication and cooperation mechanisms of systems performing any kind of mathematical service enable to study and solve new classes of problems and to perform efficient computation by distributed specialized packages.

The classification of communication and cooperation methods for logical and symbolic computation systems given in this paper provides and surveys different methodologies for combining mathematical services and their characteristics, capabilities, requirements, and differences. The methods are illustrated by recent well-known examples.

We separate the classification into communication and cooperation methods. The former includes all aspects of the physical connection, the flow of mathematical information, the communication language(s) and its encoding, encryption, and knowledge sharing. The latter concerns the semantic aspects of architectures for cooperative problem solving.

1. Introduction

The design of general techniques to combine and integrate several systems has been initiated in many areas. For instance, the integration of theorem proving and symbolic mathematical computing has recently emerged from prototype extensions of single systems to the study of environments with interaction among distributed systems. However, there are no common languages, protocols, or standards for such interfaces.

Communication and cooperation mechanisms for logical and symbolic computation systems enable to study and solve new classes of problems and to perform efficient computation through cooperating specialized packages. On the one hand, computer algebra systems (*CAS*) offer an extensive collection of efficient mathematical algorithms which could improve the efficiency of theorem proving systems (*TPS*). On the other hand, they ignore AI methods (e.g. theorem proving, planning of proofs and computations, machine learning) and their capabilities, e.g. verification of properties of mathematical objects using a TPS.

Basic architectures for performing communication among TPS and CAS are introduced in [14]. The classification given here is a result of generalizations and extensions of communication and cooperation mechanisms for software systems performing any kind of mathematical computation. We call such systems *mathematical services* ($\mathcal{MS}$) which cover CAS and other symbolic computation packages, TPS, proof checkers and verification tools, numerical computation systems, visualization and type-setting applications, and format converters. This classification is illustrated by well-known recent examples of communication and cooperation mechanisms for both logical and symbolic computation systems. It provides and surveys different methodologies for combining such systems and their characteristics, capabilities, requirements, differences, and may guide the developments and selection of methods in this ongoing research. However, it must be pointed out that some of the presented architectures and communication methods are not specific to mathematical information and could be applied to combine other systems as well.

We separate the mechanisms into *communication* and *cooperation* methods. The former include all aspects of the physical connection, the flow of mathematical information, the communication language(s) and its encoding, encryption, and knowledge sharing. Communicating $\mathcal{MS}$ send and retreive mathematical information and messages. The aspects of the "semantics" of these interactions are specified according to the level of cooperation among the distributed systems. Depending on their behaviour they can be classified into: master/slave, subpackage, black box, trust, extensible and exchangeable, consistency and closure.

As of today, there is no systematic investigation of the different possible methodologies to integrate heterogeneous mathematical systems. The goal of this paper is to fill this gap.

This paper is organized as follows. Section 2 gives an overview about architectures combining logical and symbolic computation systems. The advantages are illustrated by some recent well-known examples. The classification of such architectures based upon the features of the involved communication and cooperation methods is given in section 3 and section 4 respectively.

2. Combining Logical and Symbolic Computation Systems

The advantages of combining logical and symbolic computation systems are improved expressive power and more powerful inference capabilities. There are various applications for composing those systems, like multi-logic provers, hardware and software verification, proofs with arithmetics and constraints, program transformations.

There is a lack of languages and standards for interfaces between systems for mathematical computation. The reasons are manyfold: (i) CAS and TPS are designed, implemented and validated as stand-alone systems, (ii) many systems are copyrighted and allow neither communication nor external access to internal methods, (iii) they do not provide interfacing.

Several communication and cooperation methods have already been examined. The basic level of cooperation is just to exchange mathematical information. To enable mathematicians, TPS or CAS to pass proofs, theorems, functions, algorithms or any kind of mathematical objects offline by electronic mail, cut *&* paste or ftp requires communication in terms of a common language. Open Mechanized Reasoning Systems [11] and OpenMath [2] introduce general languages suitable for specifying and communicating mathematical objects in theorem proving and symbolic mathematical computing respectively.

Higher levels of online cooperation can be achieved by adding links to interactive tools. The interfaces between HOL and Maple [13] and Isabelle and Maple [4] introduce the powerful arithmetics of a computer algebra system into a tactical theorem prover to reason about numbers or polynomials much more efficient. Maple [6] acts as a slave to the prover which controls external calls by evaluation tactics. [15] presents an interaction to provide expressive algebra of constructive type theory in computer algebra. The theorem prover Nuprl is an algebraic oracle to the CAS Weyl. Analytica [7] is an example for cooperation within the language of another system. It is written in the Mathematica [17] environment and can solve sophisticated problems in elementary analysis.

CAS/π represents a sophisticated example of a powerful graphical system-independent common user interface [16]. It was designed so that expert users can set up connections to alternative CAS or visualization tools easily and at runtime. An architecture for proof planning in distributed theorem proving is given in [8]. TPS compete and then cooperate using completion in pure equational logic using team work. The advantage of distribution is to profit from heuristics of several systems to reduce the typically immense search spaces.

3. Communication Methods

Prerequisite for distributed mathematical problem solving is communication. This section examines the communication language, its encoding and encryption, the flow of mathematical information, and the exchange of mathematical objects by common knowledge representation.

3.1. COMMUNICATION LANGUAGE AND ENCODING

A communication language defines how mathematical information can be exchanged among services. Such a language must be recognized by each system in order to to translate the information into their internal representation. Appropriate languages are the input language or internal encoding of one of the involved systems or standardized communication languages.

To select the syntax of the input language of one of the systems is natural and allows the straightforward interaction with this system. Currently, most interfaces between $\mathcal{MS}$ are built as prototypes designed to demonstrate the advantages of combining heterogeneous systems. Thus, the communication language has not been chosen according to general protocols. The prototypes described in [13, 4] communicate in terms of Maple expressions. The theorem provers HOL and Isabelle are extended both by adding syntax translations and evaluation tactics. In case of common knowledge representation (3.4) or subpackages (4.2) it is a good option to use the internal encoding of one specific system as communication method. The interaction with the Analytica [7] package is implemented with a common representation of the objects and in expressions of Mathematica's language.

Communicating in terms of the input language of one system is generally not a good choice because

systems are tough to interchange. [4] selects the input and output object representation of Maple as its communication language. To replace Maple by any other CAS requires to define a new syntax (4.3).

the input language differs from the representation language. The input object representations must be encoded into the internal application specific representations. Some types of cooperation gain efficiency at run time by communicating these internals (4.1,4.2).

services are based upon different semantics. There is no standardized semantics to expressions of mathematical objects. Some systems request case sensitive input, *d, diff* or *differentiate* may represent different functions, and the mathematical notions differ, like `^` or `**`.

Several communication languages for interfaces between software systems exchanging mathematical information have been developed, Camino-Real [3], ASAP, CC and central control [10], Posso/XDR [1], MP [12], CAS/π [16], and MathLink [18]. OpenMath [2] classifies these projects according to the framework given in the basic OpenMath model as illustrated in figure 1.

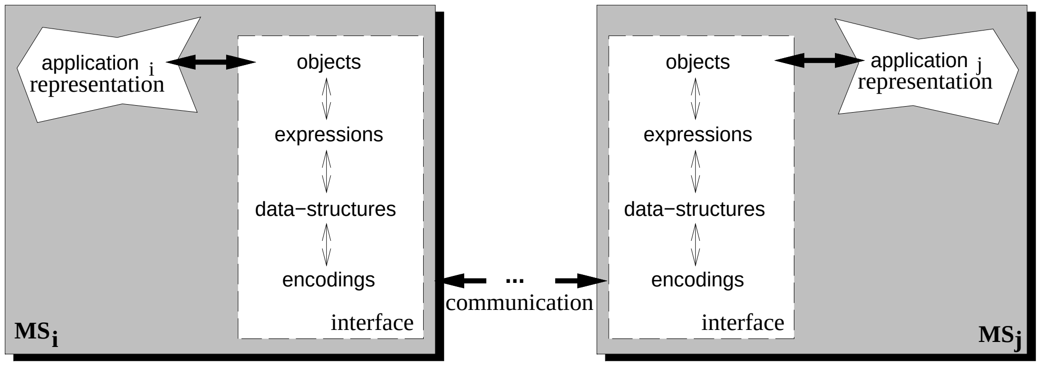

Figure 1. The communication model

The communication is not implemented as the input language to one of the involved systems but as an interface compiling the service specific representation into a standardized encoding. This encoding is either a stream of bytes or an extended Lisp-like representation suitable for transmission via files, cut *&* paste, email, ftp, and broadcasting like Unix sockets. Thus, the communication language can be described by specifying the different levels in the model: objects, expressions, data structures, and encodings.

3.2. ENCRYPTION

Current interfaces between $\mathcal{MS}$ do not consider system security aspects since the interaction often only involves packages running in the same local network. Because of the wish to transmit mathematical information via files, cut *&* paste, email or ftp (see above) future encodings must be designed to provide connections with identification and encryption.

3.3. BIDIRECTIONAL COMMUNICATION

Cooperation among several software systems can be achieved with indirect, unidirectional and bidirectional communication. According to the flow of mathematical information several architectures are illustrated in figure 2.

Although there are no links between the services with indirect communication, interaction is possible if both systems can communicate with a com-

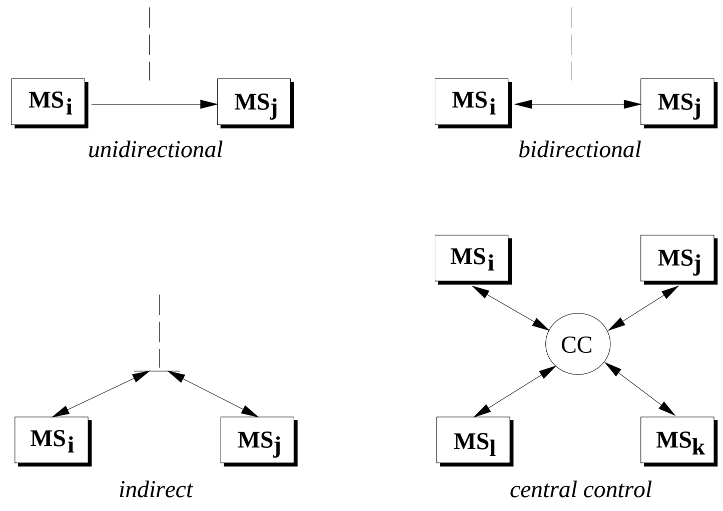

Figure 2. Flow of mathematical information

mon user interface, central unit, mediator or evaluator. Such an interface provides links to some $\mathcal{MS}$. A user can access the systems and can apply (symbolic or numerical) algorithms or theorems to solve a given problem, depending on the class of the problem. Such a simple type of interaction allows already the use of arbitrary CAS and ATP. However, the systems do not interact directly and a user must be familiar with both systems. Such an architecture combines the advantages, but also the drawbacks. CAS/π represents a sophisticated example for such an architecture [16].

To manage the communication and to hide the control from the user interface leads to an architecture with common evaluator or central control. The evaluator controls the selection of the modules by meta-knowledge on all functions and predicates. It also controls the application of algebraic algorithms and exchange of data and theorems in the $\mathcal{MS}$. The mathematical knowledge is represented separately in each module. The Central Control project [10] is a typical representative for this architecture. The tools are mainly independent: they can perform their tasks without the help of other tools.

Unidirectional links can most often be found when communicating with input or output devices like math editors, visualization tools, graphical interfaces, SGML, in case of master/slave cooperation (4.1), or subpackages (4.2). As mentioned previoulsy, typically such interfaces do not support general encodings as communication language but the input or output language of one system [4, 15, 13, 7].

The first environments providing bidirectional links have been studied recently. Such a communication requires to exchange common mathematical objects or relies on a common knowledge representation. At any step, arbitrary combinations of algorithms and theorems can be applied to solve a given problem. This combines the advantages of all involved mathematical services. The uni- and bidirectional communication is generalized to a software bus of mathematical services in [5] as illustrated in figure 3. The highlighted connection between Maple and Isabelle is described in [4].

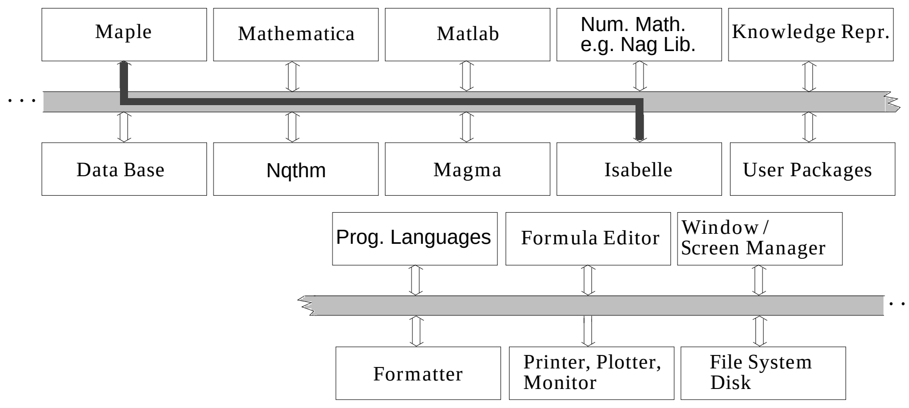

Figure 3. Software bus of mathematical services

3.4. COMMON KNOWLEDGE REPRESENTATION

Many applications require several $\mathcal{MS}$ to share their knowledge about mathematical objects. In many cases, communicating this information is neither efficient nor practical, because it may not be explicitly known which knowledge is required.

Some cooperation mechanisms obviously benefit from sharing their knowledge, i.e. communication with subpackages or direct function calls in foreign packages (4.2, Analytica [7]). A software bus (figure 3) may include a knowledge representation system suitable for representing the common knowledge. Both architectures are illustrated in figure 4.

Recent communication methods are not restricted only to exchange of function calls, theorems, numerical data, polynomials or basic mathematical information. For example, OpenMath [2] provides the exchange of mathematical objects with a defined semantics derived from its associated lexicons. However, there are no protocols to provide meta-knowledge about the systems algorithms or type information about their arguments.

To represent explicitly the mathematical information embedded in CAS requires to introduce the representation of meta-information, e.g. in

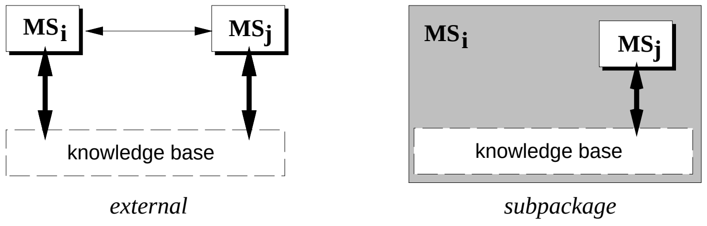

Figure 4. Common knowledge representation

terms of schemata [14]. Different schemata contain this knowledge as type schemata and algorithm schemata.

The explicit representation of the mathematical information of systems performing symbolic computations is an ongoing research project. The corresponding work has been initiated for theorem provers, e.g. by the open mechanized reasoning group [11].

4. Cooperation Methods

Communicating services exchange mathematical information and messages. This section introduces features and architectures according to the level of cooperation of these distributed systems. We illustrate interaction among systems playing different roles in the cooperation. Additionaly, some semantic issues and their resulting limitations are discussed.

4.1. MASTER/SLAVE

Mathematical problems are typically solved by dividing the problem into subproblems, solving subtasks by suitable CAS or TPS, and combining solutions to get the final result. Usually, one system is not sufficient for computing the solution. There are often more efficient special packages, some algorithms are not implemented, or the subproblem does not fit the scope of one $\mathcal{MS}$. However, it is often sufficient to solve the problem in the environment of one single system with the aid of other $\mathcal{MS}$. This is one reason why nowadays interfaces among CAS and TPS are typically restricted to master/slave cooperation [7, 13, 15, 11, 4]. The use of a $\mathcal{MS}$ is limited to some specialized tasks (algebraic simplifications, numerical computations) within the overall control of another $\mathcal{MS}$ (proofs, algebraic algorithms). The master acts as server to some client service.

Master/slave interfaces are easier to design. The master can act as a common control, the user interface of the master can act as GUI, the communication language can be chosen as the input and output language of the master, and the internal object representation of the master is the common knowledge representation. Master/slave communication typically occurs as unidirectional links (3.3) or with an intermediate bridge [13]. Some mathematical services act only as computational engine, decision procedure or oracle.

Additionaly, many groups improve the power of their own CAS or TPS by allowing external calls to other $\mathcal{MS}$. CAS are extended by links to TPS to verify certain conditions or type restrictions [15], and TPS are extended by links to CAS or numerical software to deal with arithmetics [13], mathematical objects, or to guide their proofs.

4.2. SUBPACKAGE

To avoid communication and common knowledge representation some $\mathcal{MS}$ are designed to work within the environment or language of another service (figure 4).

An example for a subpackage of CAS is Analytica [7] written in the framework of Mathematica [17]. Another example is Otter [9] which allows external function calls out of proofs. User-defined algorithms are introduced with an identification by a special character (e.g. $GCD). The extension of the prover requires the recompilation of the whole system and each algorithm has to be implemented in C. CAS provide an extensive collection of very efficient mathematical algorithms, thus reimplementation is neither necessary nor meaningful.

4.3. EXTENSIBILITY AND INTERCHANGEABILITY

General interfaces are to a certain extent system-independent and may be connected to another or many other $\mathcal{MS}$ as illustrated in figure 5. A general communication language must be adopted by each of the involved systems (3.1).

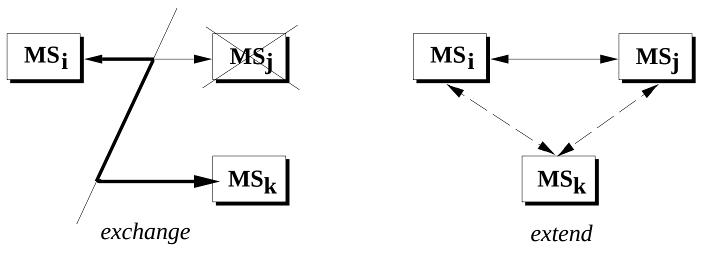

Figure 5. Extending and interchanging mathematical services

In case of master/slave cooperation, it is typically easy to change the slave by replacing the syntax translations and if necessary the evaluation tactics. To replace the master is difficult because it hosts the complete interface. One example is the cooperation between Maple and Isabelle [4] where the CAS remains unchanged and is exchangeable. The interface is part of Isabelle's extended simplifier. Extending the interface to provide communication with other $\mathcal{MS}$ is also usually easy.

To extend or change one $\mathcal{MS}$ in subpackage cooperation is more difficult. Since one system must provide the language and representation of both systems, extension or change is only meaningful between compatible packages. For example, it is impossible to replace Mathematica by Maple to run the Analytica package.

4.4. BLACK BOX

To cooperate among each other mathematical services must be able to interact in non-trivial ways. Deciders and black boxes (yes/no/result type packages) are not adequate in general because they do not provide internal mathematical information. Cooperating $\mathcal{MS}$ may need to accept information and produce results incrementally. Black boxes – often called oracles – are commonly implemented in interfaces among $\mathcal{MS}$ as they allow to combine systems as tools (clients) in master/slave cooperations [4, 13].

Components like *handles* for open mechanized reasoning systems [11] give access to proofs and derivation structures of TPS and open the black boxes. They can be extended to *contexts* [5] which provide the necessary intermediate information to incremental and cooperative problem solving. One example of symbolic calculators providing internal information is by schemata ([14], 3.4).

4.5. CONSISTENCY AND CLOSURE

Systems exchanging terms face two problems: does another service understand all transmitted terms and are the resulting terms defined in the signature of the service.

To guarantee that a mathematical service *understands* transmitted terms the term algebra must provide consistency w.r.t. the signature of that service.

Definition 1
The term algebra $T_\Sigma(X)$ *is called* consistent w.r.t. the signature A*, iff for all* $f \in A$ *any term* $f(a_1, \ldots, a_n) \in T_\Sigma(X)$ *already lies in* $T_A(X)$*.*

In a consistent term algebra all subterms of a term are in the subsignature $T_A(X)$, provided that the outermost connective belongs to the subsignature A. It is thus easy to recognize terms that can be passed to a service during cooperation. One has to verify that the outermost connective lies in A. Consistent term algebras are a strong limitation because they prevent to generate terms containing objects of foreign mathematical services. However, nowadays systems were designed as stand-alone systems and typically cannot handle such objects, e.g. they cannot reduce parts of expressions which also contain unknown objects like logical connectives between polynomial expressions of CAS.

The corresponding problem to understand the result computed by another service occurs in bidirectional communication and master/slave cooperation. To ensure that a term returned by a mathematical service (viewed as an operator E) does not contain unknown symbols is ensured by **closure w.r.t. signatures.** Even if $T_\Sigma(X)$ is consistent w.r.t. A the result returned by the service may lie in a signature $B \not\subseteq \Sigma$.

Definition 2
The operator $E : T_A(X) \rightarrow T_B(X)$ *is* closed w.r.t. A and B *if* $A \subseteq B \subseteq \Sigma$.

The OpenMath model introduces a common communication language for CAS which can be transformed into OpenMath objects. The mathematical service tries to compile these objects into the application specific representation. Objects containing unknown terms are rejected and can not be represented. In the case of communication in terms of input syntax of one of the systems problems with consistency and closure can be avoided by restriction to common subsets of both signatures. The examples illustrated in this paper require both consistency and closure.

4.6. TRUST

[13] introduces levels of trust between CAS and TPS which can be generalized to classify any cooperation among $\mathcal{MS}$. Depending on the confidence on the accuracy of the answers given by another service a system trusts completely, partially, or not at all.

No trust at all may force one system to verify some conditions or results by another, or – if possible – use the results as an aid to compute the result by itself. This is especially useful in guiding a proof where verification is much less computationally complex than computation. Additionaly, $\mathcal{MS}$ may sometimes generate incorrect answers which is often unacceptable, i.e. in theorem proving.

Partial trust can occur when accepting results are accepted during interactive or temporary computations but these results have to be checked before being recorded. Another technique is to mark each result in the computation and communication with a rational representing its confidence.

Complete trust ([15, 4]) means that the result given to any request is accepted as truth. This is commonly implemented in current prototypes among $\mathcal{MS}$ and numerical packages because of its simplicity and efficiency.

For example, the advantage of fast computations by symbolic or numerical software may be jeopardized by the slow arithmetics of theorem provers.

5. Conclusion

The development of general techniques for the integration of systems performing mathematical computations has not yet led to the definition of common languages, protocols, and standards. The classification of communication and cooperation methods given in this paper provides and surveys methodologies for combining $\mathcal{MS}$ and their characteristics, capabilities, requirements, differences. Its purpose is to guide the selection of methods and developments in this ongoing research.

Cooperation by distributing tasks between mathematical services is a subject of ongoing research. Among the arising problems is the black box behaviour of almost any current system. To plan and control such environments requires to represent meta-knowledge in local or global bridges or supervisors.

Among the work in progress is the design of an intelligent assistant – an environment whose semantics allows a consistent treatment of algorithms and theorems. A result of this work is the integration of the tactical theorem prover Isabelle and Maple [4]. The extension of contexts [5] is another step towards environments performing distributed mathematical problem solving.

There are obviously different approaches to what can be seen as interoperability of heterogeneous systems. For instance, we are presently investigating the feasibility of designing communication protocols based upon the types of the objects to be exchanged. This can be set in the framework of an agent approach to software engineering.

References

1. J. Abbott
PossoXDR Specifications. In Posso project internal report, 1994.
2. J. Abbott, A. van Leeuwen, A. Strotmann
Objectives of OpenMath. Submitted to Journal of Symbolic Computation, 1995.
3. D. Arnon, R. Beach, K. McIsaac, C. Waldspurger
CaminoReal: an Interactive Mathematical Notebook. In Proceedings of International Conference on Electronic Publishing, Document Manipulation and Typography, Cambridge University Press, 1988.
4. C. Ballarin, K. Homann, J. Calmet
Theorems and Algorithms: An Interface between Isabelle and Maple. In A.H.M. Levelt (Ed.), Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95), pp. 150–157, ACM Press, 1995.
5. J. Calmet, K. Homann
Distributed Mathematical Problem Solving. In E. Shamir, M. Koppel (Eds.), Pro-

ceedings of 4th Bar-Ilan Symposium on Foundations of Artificial Intelligence (BISFAI'95), pp. 222–230, 1995.

6. B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt
Maple V Language Reference Manual, Springer-Verlag, 1992.

7. E. Clarke, X. Zhao
Analytica – An Experiment in Combining Theorem Proving and Symbolic Computation. Carnegie Mellon University, Technical Report CMU-CS-92-147, 1992.

8. J. Denzinger, M. Fuchs
Goal oriented equational theorem proving using team work. In B. Nebel (Ed.), Proceedings of 18th German Annual Conference on Artificial Intelligence (KI'94), pp. 343-354, Lecture Notes in Artificial Intelligence 861, Springer-Verlag, 1994.

9. W.W. McCune
OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Labaratory, 1994.

10. S. Dalmas, M. Gaëtano, A. Sausse
Distributed Computer Algebra: the Central Control Approach. In H. Hong (Ed.), Proceedings of 1st International Conference on Parallel Symbolic Computation (PASCO'94), pp. 104–113, Lecture Notes in Computing 5, World Science, 1994.

11. F. Giunchiglia, P. Pecchiari, C. Talcott
Reasoning Theories – Towards an Architecture for Open Mechanized Reasoning Systems. In Proceedings of First International Workshop Frontiers of Combining Systems (FroCoS'96), Kluwer, 1996.

12. S. Gray, N. Kajler, P. Wang
MP: A Protocol for Efficient Exchange of Mathematical Expressions. In Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'94), ACM Press, 1994.

13. J. Harrison, L. Théry
Extending the HOL Theorem Prover with a Computer Algebra System to Reason About the Reals. In J.J. Joyce, C.-J.H. Seger (Eds.), Higher Order Logic Theorem Proving and its Applications (HUG'93), pp. 174–184, Lecture Notes in Computer Science 780, Springer-Verlag, 1993.

14. K. Homann, J. Calmet
Combining Theorem Proving and Symbolic Mathematical Computing. In J. Calmet, J.A. Campbell (Eds.), Integrating Symbolic Mathematical Computation and Artificial Intelligence (AISMC-2), pp. 18–29, Lecture Notes in Computer Science 958, Springer-Verlag, 1995.

15. P. Jackson
Exploring Abstract Algebra in Constructive Type Theory. In A. Bundy (Ed.), Automated Deduction (CADE-12), pp. 590–604, Lecture Notes in Artificial Intelligence 814, Springer-Verlag, 1994.

16. N. Kajler
CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems. In Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'92), pp. 376–386, ACM Press, 1992.

17. S. Wolfram
Mathematica: a System for Doing Mathematics by Computer, Addison-Wesley, 1991.

18. Wolfram Research Inc.
MathLink Reference Guide Version 2.2, Mathematica Technical Report, Wolfram Research, 1993.