Algebra, ISL Report PR 801/96, Saint-Louis, pp. 27.1–27.3, 1996.

Languages for Cooperative Problem Solving in Mathematics

Karsten Homann

Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5 · 76131 Karlsruhe · Germany
homann@ira.uka.de

ABSTRACT

The design of languages 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. An overview of recent well-known projects on such cooperations is given in the references cited in [BHC95]. 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.

The classification of communication and cooperation methods for logical and symbolic computation systems [CaHo96] provides a general framework for methodologies for combining mathematical services and their characteristics, capabilities, requirements, and differences. The advantages of combining systems performing any kind of mathematical computation (*mathematical services*) 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.

A communication language defines how mathematical information can be exchanged among services. It 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.

Several communication languages for interfaces between software systems exchanging mathematical information have been developed. OpenMath [AvLS95] classifies these projects according to the framework given in the basic OpenMath model. Figure 1 illustrates an interface which is part of any mathematical service. Some applications may not distinguish between some of the levels.

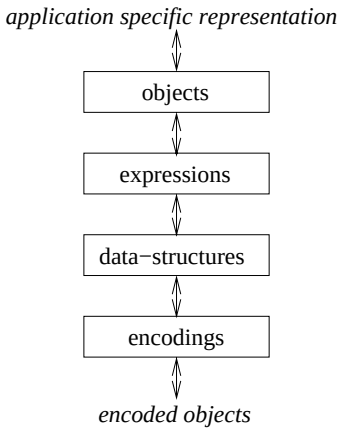

Figure 1: An Interface for Mathematical Services

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.

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 [BHC95]. The extension of contexts [CaHo95] is another step towards environments performing distributed mathematical problem solving.

References

[AvLS95] J. Abbott, A. van Leeuwen, A. Strotmann
Objectives of OpenMath. Submitted to Journal of Symbolic Computation, 1995.

[BHC95] 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.

[CaHo95] J. Calmet, K. Homann
Distributed Mathematical Problem Solving. In E. Shamir, M. Koppel (Eds.), Proceedings of 4th Bar-Ilan Symposium on Foundations of Artificial Intelligence (BISFAI'95), pp. 222–230, 1995.

[CaHo96] J. Calmet, K. Homann
Classification of Communication and Cooperation Mechanisms for Logical and Symbolic Computation Systems. To appear in Proceedings of First International Workshop Frontiers of Combining Systems (FroCoS'96), Kluwer Series on Applied Logic, 1996.