ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ"

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 2 — Приложна математика и информатика
Том 90, 1996

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI"

FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 2 — Mathématiques Appliquée et Informatique
Tome 90, 1996

KAM — A KNOWLEDGE-BASED TOOL FOR DEVELOPING COMPUTER ALGEBRA SYSTEMS

MARIA M. NISHEVA-PAVLOVA

The paper presents a description of KAM — a knowledge-based tool for building computer algebra systems developed at the Faculty of Mathematics and Informatics, Sofia University. The main features of KAM are analyzed. The architecture of KAM and the knowledge representation formalisms supported by the tool transformation rules, frames, rewrite rules, generalized rules, are briefly described. A presentation of the experimental computer algebra system STRAMS being under development as an application of KAM is given.

Keywords: intelligent computer algebra system, knowledge representation tools
1991/95 Math. Subject Classification: main 68T35; secondary 68T30, 68T05

1. INTRODUCTION

In the last 2–3 decades Computer Algebra Systems (CAS) have been successfully used in many fields of science and engineering. These systems can help in the solution of different types of problems connected with the execution of complicated and labour-consuming transformations of mathematical expressions. "Classical" CAS like Reduce, Macsyma, Maple, Mathematica etc. provide thousands of sophisticated algebraic algorithms, but sometimes they are difficult for use. On the one hand, it is often hard to select the appropriate algorithm from the amount of available algorithms. On the other hand, the interpretation of the solution sometimes needs significant efforts, because the system does not give any information

about the solution steps. In other words, the user does not receive any explanation or information about the problem solving process (for instance, how the solution of the problem is found, or why the output is the solution of the given problem). Moreover, the extension of the functional facilities of most of the "classical" CAS is a hard job that usually needs a great amount of programming efforts.

The problem here is that "classical" CAS have no mathematical knowledge about the properties of the functions and the problem solving methods represented in an explicit, declarative way. Their mathematical knowledge is embedded implicitly in the algorithms and is inaccessible to the user.

For that reason a series of successful attempts has been made to integrate the classical methods for developing CAS with Artificial Intelligence methods and tools. These attempts have been made in different directions.

On the one hand, many famous large CAS have been supplied with intelligent user interfaces. For example, the Praxis [5] system is implemented as a rule-based expert system for the computer algebra system Macsyma.

On the other hand, several systems like AXIOM [6] and MAGMA [1] are aimed at the integration of means for description of algebraic structures (and, more general, of database and deductive facilities) with computer algebra algorithms.

In the systems of the type of LP [11], APS [7, 8] etc. the emphasis basically falls on the representation of knowledge about the methods for mathematical problem solving.

The hybrid knowledge representation system MANTRA [2, 3] is a next step of such integration. It combines different formalisms for specification of mathematical domains and provides a computational environment for solving problems combining the strong mathematical algorithms with heuristic search for solutions.

Thus the term "intelligent CAS" becomes quite popular in the last years. In general, intelligent CAS are systems that are capable to manipulate different types of mathematical knowledge and use a large set of Artificial Intelligence methods and techniques.

A set of projects aimed at the investigation of different aspects of building intelligent CAS has been under development at the Faculty of Mathematics and Informatics, Sofia University. An approach to building CAS has been developed with the purpose of creating flexible, "open" CAS that:

— can easily be integrated with other software packages and can be used in the development of CAD systems, intelligent tutoring systems etc.;

— solve in a satisfactory way the problems discussed above, in particular, are able to do some kind of learning and explanation generation.

This approach is based on the representation of knowledge about the properties of the functions and the methods for mathematical problem solving defined by these properties.

The knowledge-based tool KAM described in this paper has been developed with a mainly experimental purpose. It is a software system supporting our approach to building CAS. KAM has been used as a tool for performing experiments with the suggestions presented in this paper and in [9], and as an instrument for building CAS that could have concrete applications. The Common Lisp interpreter

from the integrated environment POPLOG, version 14.5, has been used for its implementation [12].

2. MAIN CHARACTERISTICS OF THE APPROACH TO BUILDING CAS SUPPORTED BY KAM

The suggested approach is based on the conception for the knowledge representation in CAS discussed in Section 2.1. This conception can be considered as a modification and further development of some ideas and mechanisms suggested by B. Silver in [11].

2.1. KNOWLEDGE REPRESENTATION

The formalism we suggest for the representation of mathematical knowledge in CAS is a hybrid one. It includes several levels of representation.

The knowledge about the properties of the functions manipulated in the corresponding CAS can be described using a special type of rules called rewrite rules. The structure of each rewrite rule contains a description of a correct transformation of a definite class of mathematical expressions and a formulation of some general preconditions for its performance (if there are any).

The description of the methods for transformation of expressions and equations in the corresponding definition domain can be realized by the so-called generalized rules (methods). Each generalized rule describes a sequence of transformations of the given expression (equation) aimed at its conversion into a particular form. Usually, generalized rules contain sequences of properly grouped rewrite rules. Depending on their contents and application mode, they are classified as declarative, procedural and hybrid. Another classification criterion of the generalized rules is the role they play in the problem solving process of a given, relatively complex task (equation solving, symbolic integration etc.). In this sense they are classified as key and non-key ones. The key generalized rules play a significant role in the control of the search in the state graph of the corresponding problem.

The knowledge about the problem solving methods for the included types of tasks can be described either directly by proper generalized rules or using specific constructions called schemata. A schema is a sequence of non-key generalized rules. It describes a definite step in the problem solving process of a relatively complex task. Schemata are a natural generalization of methods (generalized rules). The precondition of a schema is the applicability of its first generalized rule. The goal is to solve the problem or to be able to apply a key generalized rule after the application of the schema.

During its working cycle each particular CAS realizes a search in the state space of the user's problem. In the role of operators it uses the schemata and generalized rules available at the current moment. Some additional search control knowledge is used with the purpose of avoiding the possible cycles and focusing the attention of the system on certain situations.

2.2. LEARNING FACILITIES

The formalism for the knowledge representation in CAS described in Section 2.1 is a good basis for the realization of some mechanisms for unsupervised learning that could considerably increase the effectiveness of these systems. In general, the unsupervised learning in the discussed type of CAS is based on the capability for discovering and memorizing the schemata used in the problem solving process of some complex tasks (such as factorization, equation solving or symbolic integration). These new schemata could be directly used in solving further problems. More precisely, the suggested unsupervised learning mechanism can be summarized as follows. Before the first run of the corresponding CAS the set of schemata included in its knowledge base is empty. During its working cycle the system uses the schemata and generalized rules available in the knowledge base at the moment and the built-in search control knowledge. The discovering and the application of a proper schema can considerably speed up the problem solving process. Whenever a given problem is successfully solved, the system can analyze the used sequence of generalized rules, construct the new schemata candidates and merge them with the set of existing schemata. In this way the CAS can perform some kind of self-perfection, i.e. some type of unsupervised learning.

2.3. EXPLANATION GENERATION

It is advisable for an intelligent CAS to be able to generate various kinds of explanations. The minimal requirement in this respect is the capability to explain the mode in which a given problem is solved. To ensure this capability, a CAS of the discussed type can keep in a special record the history of the current session. The history of a given session may contain the sequence of problems solved at the time of this session and the main steps of their problem solving process. These steps correspond to the methods (generalized rules) used by the system. The processing of a given explanation request can be performed in two steps: extracting the corresponding information from the record and generating the text of the explanation. This text ought to be in a natural language and to contain a description of the consequent steps of the problem solving process.

2.4. COMPARISON WITH SILVER'S APPROACH

As it was mentioned above, the approach to building knowledge-based CAS supported by KAM is a modification and further development of the ideas of the so-called Precondition Analysis suggested by B. Silver and realized in his system LP [11]. The main differences between our approach and Silver's one can be briefly brought to the following:

— our approach is intended for the development of general-purpose CAS with varied functional facilities (while LP is a special-purpose CAS for symbolic equation solving). Therefore our generalized rule mechanism is considerably more complicated;

— schemata in our formalism realize the separate steps in the problem solving process of the given mathematical task (but, in contrast with LP schemata, they do not realize the entire problem solving process of the given task from the beginning to the end). Thus our knowledge representation formalism is more flexible and relevant to the human problem solving process;

— our approach is aimed at the development of means for unsupervised learning while the learning process in LP is a supervised one.

3. ARCHITECTURE OF KAM

The final version of KAM will include the following functional components:

— a mathematical problem solving engine;

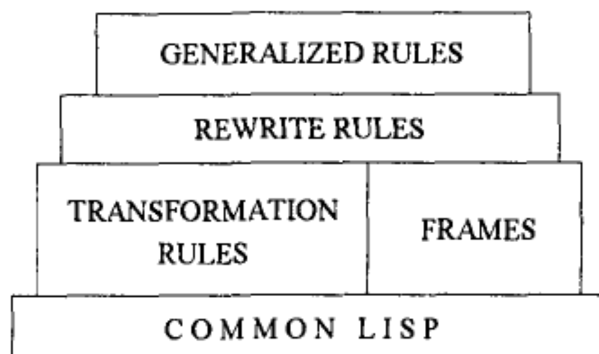
— an explanation module;

— a rule editor;

— an interface module.

The mathematical problem solving engine consists of two modules: an inference engine and a learning module realizing respectively the problem solving and unsupervised learning mechanisms discussed in Section 2.

From the implementation point of view the structure of the inference engine of KAM seems as follows:



Transformation rules are a declarative formalism intended for the description of list transformations. The transformation rule and the frame manipulation mechanisms have been used in the implementation of the rewrite rule interpreter. The generalized rule interpreter uses some basic means provided by the frame interpreter and the rewrite rule interpreter. The control block realizes the search process in the state space of the given problem. In particular, the search for proper schemata and generalized rules and their application are realized by the control block using the generalized rule interpreter. When a problem is successfully solved, the learning module can be activated. It discovers the new schemata used by the inference engine (if there are any) and adds them to the set of known schemata.

The explanation module realizes the explanation generation mechanisms described in Section 2.3.

The develpoment of the rule editor and the interface module of KAM is still at the designing phase.

The rule editor will be a tool assisting the users of KAM in building the knowledge bases of the concrete CAS they intend to create. It will suggest means for convenient input and editing of rewrite rules and generalized rules and procedures for automatical translation of these rules into the internal form that is "understandable" for the corresponding interpreter.

The interface module will be the component of KAM the typical users will be in touch with. It will enable the user to choose from a special menu the variuos functions of KAM relevant to the development of concrete CAS: using the rule editor (i.e. building or modification of the knowledge base of a given CAS), building the functional modules (the inference engine, the learning module or the explanation module) of a new CAS by adjustment of copies of the corresponding modules of KAM or their proper subsets, designing the interface module of a CAS etc.

As it was mentioned above, the current version of KAM includes working prototypes of the mathematical problem solving engine and the explanation module only. Nowadays the rule editor and the interface module are still at the designing phase of development. Therefore the current version of KAM is intended for users with good skills in Common Lisp programming. The completion of proper working versions of the interface module and the rule editor will help mathematicians, engineers, teachers and other subject specialists with no Lisp programming experience to work conveniently and successfully with KAM.

The knowledge representation formalisms supported by the inference engine of KAM are briefly described in Sections 4–7.

4. TRANSFORMATION RULES

Transformation rules are a formalism for description of list transformations. Each transformation rule consists of two parts — a left-hand part and a right-hand one. The left-hand part describes the class of lists the rule can be applied to. It is called the pattern. The right-hand part describes the method for the new list construction. It is called the constructor. The rule is applied to a list called the object list (the object). The application of a given transformation rule to a given object list is accomplished in the following way:

— an attempt for matching the object list and the pattern is made;

— if the matching succeeds, some elements of the object list are extracted in order to be used in the new list construction. Then the result list is constructed in a way described by the constructor;

— if the matching fails, the rule is not applicable to the given object. In this case *nil* is returned as a result.

Hence the transformation rule interpreter performs two main operations: pattern matching and construction of the result list.

Pattern matching is an operation in which each element of the pattern matches one or several elements of the object list in accordance with a definite set of rules. The pattern consists of two types of elements — ordinary elements and special ones. An ordinary element of the pattern matches only an equal to it element of

the object list. Special elements of the pattern match the corresponding elements of the object list using specific rules. Examples of admissible types of special elements:

(*?* ⟨*var*⟩).

If the variable ⟨*var*⟩ is not bound, this special element successfully matches the corresponding element of the object list and this element of the object list is assigned to ⟨*var*⟩. In the other case, the matching succeeds if the value of ⟨*var*⟩ is equal to the corresponding element of the object.

(*?* ⟨*var*⟩ ⟨*predicate-name*⟩).

Here ⟨*predicate-name*⟩ is a name of a Common Lisp function. This special element is interpreted as the previous one with one additional condition for the success of the matching: the evaluation of the function ⟨*predicate-name*⟩ with an argument equal to the corresponding element of the object must be different from *nil*.

The construction of the result list is an evaluation of some of the constructor elements. In other words, the result list can be obtained from the constructor by an evaluation of some of its elements and a substitution of these elements with the corresponding values.

5. FRAMES

As a module of KAM a frame system called FS is developed. FS gives a set of standard frame manipulation means that can be divided into the following groups: frame definition, creation of a frame instance, providing an access to the slots of a frame instance.

To define a frame, one has to determine the name of the frame and to create a description of the structure of this frame. The names of the frames and the slots are Lisp symbols. The possible facet names in FS are the following: *value, default, if-needed, if-added, if-removed.* The semantics of these facets is identical to the conventional one. The values associated to them are Lisp expressions (in particular, Common Lisp functions can be used).

A Common Lisp function called a service procedure or a method can be used as a slot value as well. Slots supplied with method values have no facets. Methods are performed by sending messages to them. A special type of methods are the so-called auto-methods. Auto-methods contain calls to the slots of the frame they belong to.

FS provides some means for inheritance of properties that is reduced to the possibility of appending a given subset of the slots and facets of the parent frame to the structure of the inheritor frame.

As it was mentioned above, the frame manipulation mechanisms in KAM have been used in the implementation of the rewrite rule interpreter and the generalized rule interpreter. Each particular rule of these two types has been described as an instance of a frame with proper structure and methods.

6. REWRITE RULES

Rewrite rules have been used to describe the properties of the functions that can be manipulated in the corresponding CAS. Each rewrite rule includes a description of a correct transformation of a given class of mathematical expressions and, if necessary, a general precondition for its performance. Examples:

$c_1a + c_2a = (c_1 + c_2)a$,

$e^a e^b = e^{a+b}$,

$\operatorname{tg}(a+b) = \dfrac{\operatorname{tg} a + \operatorname{tg} b}{1 - \operatorname{tg} a \operatorname{tg} b}$ with precondition a, b, $a+b$ different from $\dfrac{(2k+1)\pi}{2}$.

The implementation of the means for rewrite rule description and application uses mainly the mechanisms provided by the frame and the transformation rule formalisms. A possibility for procedural implementation of some rewrite rules aimed at reaching better effectiveness is provided as well.

7. GENERALIZED RULES

The generalized rule formalism has been used for the description of the transformation methods applicable to the types of expressions and equations the corresponding CAS can manipulate. A generalized rule can be considered as a description of a sequence of transformations of the given expression aimed at its conversion into a definite form. In this sense, most often generalized rules are sets of properly grouped rewrite rules. According to the contents of their description and the method of their application generalized rules can be classified as declarative, procedural and hybrid (combined).

Each generalized rule consists of two parts — a precondition and a body. The precondition is a predicate whose satisfaction is a necessary condition for the application of the generalized rule and for achieving its purpose. The evaluation of the precondition of a given generalized rule is the first step of its application. If the precondition is true, then the body of the generalized rule is performed. Depending on its type, the body of a generalized rule may contain:

— *in the case of a declarative rule*: a sequence of rewrite rules. Each of them can include some additional control information about the correct direction(s) of its application;

— *in the case of a procedural rule*: the code of a procedure realizing the application of this rule;

— *in the case of a hybrid rule*: a set of pairs (*pattern, procedure*). When the examined expression matches one of the patterns, the corresponding procedure is executed.

Most often declarative generalized rules have been used in building CAS. The body of such a rule consists of a sequence of rewrite rules that can be divided in three groups: pre-rules, basic rules, post-rules.

The pre-rules are intended to prepare the given expression for the performance of the basic rules. The post-rules are used to remove some "defects" remaining after the performance of the basic rules.

There are three basic types of declarative generalized rules according to the mode of application of their bodies: normal, cyclic and recursive. The body of a normal generalized rule is performed in the following way: First the pre-rules are consecutively applied to the given expression. Each of them is executed on the result returned by the previous one. Then the basic rules are applied in the same way on the result of the execution of the pre-rules. At last the post-rules are applied in the described way.

The body of a cyclic generalized rule contains only one basic rule. It is performed in the following way: First the pre-rules are executed as in the case of a normal rule. Then the basic rule is executed. If it has not changed its argument, the execution of the body of the generalized rule stops and the current result is returned. In the other case, the corresponding post-rules are performed and then a cyclic execution of the described sequence of steps is carried out until the basic rule returns its argument unchanged.

The body of a recursive generalized rule is first executed on the subexpressions of the given expression and then it is applied to the obtained new argument.

It is possible to construct some combinations between the basic types of declarative generalized rules. For example, very attractive are the so-called cyclic recursive generalized rules that can be used as a proper mean for the description of some methods for expression simplification.

8. ONE APPLICATION OF KAM: THE STRAMS COMPUTER ALGEBRA SYSTEM

An experimental CAS named STRAMS [9, 10] has been under development at the Faculty of Mathematics and Informatics, Sofia University, using the current version of KAM. STRAMS is a knowledge-based system for symbolic manipulations of expressions that may contain numbers, symbols and the functions: +, −, *, /, power function, exponential, logarithmic and trigonometric functions. It is intended for solving the following main problem types:

— expression simplification;

— symbolic equation solving;

— symbolic differentiation;

— symbolic integration.

The formalism discussed in Section 2 is used for the knowledge representation in STRAMS. The knowledge of STRAMS about the properties of the manipulated functions is described by rewrite rules. Examples of such rules can be found in Section 6.

The description of the methods for transformation of expressions and equations in STRAMS is realized by a set of generalized rules. Here we give several examples

of generalized rules (generalized rules are called methods in STRAMS) used in the symbolic equation solving subsystem of STRAMS.

Example 1. Isolation.

Let an equation $eq: \; expr_1 = expr_2$ be given and let f be the outermost function in $expr_1$. The method consists in the application of the inverse of f to $expr_1$ and $expr_2$. The precondition of the method is: the unknown occurs in only one of the arguments of f and $expr_2$ does not contain the unknown. The goal is in the left-hand side of eq to remain only the argument containing the unknown.

The method is a key one and is implemented procedurally due to effectiveness considerations.

Example 2. Collection.

The goal of this method is to reduce the number of occurrences of the unknown. Collection is a non-key method with no explicit precondition. STRAMS applies it only if none of the key methods can be applied. So the precondition of Collection (and of all non-key methods) is: there is no key method with satisfied preconditions.

The method is declarative, normal. One of its rewrite rules is

$AB + AC = A(B + C)$ with precondition A must contain the unknown.

Example 3. Attraction.

Attraction is a non-key method with no explicit precondition. The goal here is to move the occurrences of the unknown "closer" together in hope that another method (for example Collection) will then be applicable. One of the rewrite rules of Attraction is

$AC + BC = (A + B)C.$

In this rule the expressions A and B are attracted, so they must contain the unknown.

The problem solving process of some complex tasks (such as equation solving or symbolic integration) consists of a series of steps realized by the STRAMS schemata. According to the ideology of KAM a schema is a sequence of non-key generalized rules (methods) whose application directs to a definite aim in the problem solving process.

STRAMS has some means for unsupervised learning and explanation generation that entirely correspond to the ideas and mechanisms described in Section 2.

The mathematical problem solving engine and the explanation module of KAM were used without any effort for the implementation of STRAMS. The knowledge base of STRAMS was built by direct recording of the corresponding rewrite rules and generalized rules in internal form. The interface module of STRAMS that analyzes the user requests and realizes the general control of the system's work was developed especially for the purpose.

9. CONCLUSION

Our experience in developing and using STRAMS demonstrates that the current version of KAM and the approach to building CAS supported by it do really work. Our current activities are oriented to the development of working versions of the rule editor and the interface module of KAM.

At the same time some research activities have been carried out with the purpose of extending our approach to building CAS supported by KAM in the following directions:

— including some mechanisms for supervised learning. It is useful at that to provide means for learning both new schemata and new generalized rules;

— improvement of the explanation generation mechanisms. It is necessary to extend the range of explanations that can be generated by the discussed type of CAS and to develop some means for generating explanations with different degrees of circumstantiality;

— including some proper formalisms for problem solving and learning by analogy. Our hypothesis is that the methodology suggested by J. Carbonell in [4] can be used for this purpose.

ACKNOWLEDGEMENTS. This work is partly funded by the Sofia University SRF under Contract No. 173/1996.

REFERENCES

1. Bosma, W., J. Cannon. Handbook of MAGMA Functions. Sydney, 1994.
2. Calmet, J., I. Tjandra. On the Design of an Artificial Intelligence Environment for Computer Algebra Systems. In: *Computer Algebra in Physical Research*, eds. D. Shirkov, V. Rostovtsev, V. Gerdt, World Scientific, Singapore, 1991, 4–8.
3. Calmet, J., I. Tjandra. A Unified-Algebra-based Specification Language for Symbolic Computing. *LNCS*, **722**, Springer-Verlag, 1993, 122–133.
4. Carbonell, J. Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In: *Machine Learning*, eds. R. Michalski, J. Carbonell, T. Mitchell, Tioga, CA, 1983, 137–161.
5. Clarkson, M. Praxis: Rule-based Expert System for Macsyma. *LNCS*, **429**, Springer-Verlag, 1990, 264–265.
6. Jenks, R., R. Sutor. AXIOM. Springer-Verlag, 1992.
7. Kapitonova, Y., A. Letichevsky, M. L'vov, V. Volkov. Tools for Solving Problems in the Scope of Algebraic Programming. *LNCS*, **958**, Springer-Verlag, 1995, 30–47.
8. Letichevsky, A., J. Kapitonova, S. Konozenko. Computations in APS. *Theoretical Computer Science*, **119**, 1993, 145–171.
9. Nisheva-Pavlova, M. A Knowledge-Based Approach to Building Computer Algebra Systems. In: *Proceedings of JCKBSE'96*, Sozopol, 1996, 222–225.
10. Nisheva-Pavlova, M. Experimental Knowledge-Based System for Computer Algebra. *Mathematics and Education in Mathematics*, **25**, 1996, 205–208 (in Bulgarian).

11. Silver, B. Precondition Analysis: Learning Control Information. In: *Machine Learning*, vol. 2, eds. R. Michalski, J. Carbonell, T. Mitchell, Morgan-Kaufmann, 1986, 647–670.

12. Todorov, B. An Environment for Building Special-Purpose Knowledge-Based Systems for Computer Algebra. MSc Thesis, Sofia University, 1994 (in Bulgarian).

Received on May 26, 1997
Revised on September 11, 1997

Faculty of Mathematics and Informatics
"St. Kl. Ohridski" University of Sofia
5 Blvd. J. Bourchier
BG-1164 Sofia, Bulgaria
E-mail: marian@fmi.uni-sofia.bg