ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 2 — Приложна математика и информатика
Том 91, 1997

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

FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 2 — Mathématiques Appliquée et Informatique
Tome 91, 1997

KNOWLEDGE REPRESENTATION AND PROBLEM SOLVING IN THE INTELLIGENT COMPUTER ALGEBRA SYSTEM STRAMS

MARIA M. NISHEVA-PAVLOVA

The paper discusses the intelligent computer algebra system STRAMS being under development at the Faculty of Mathematics and Informatics, Sofia University. The functional facilities and the architecture of STRAMS are briefly described. The presentation focuses on issues related to the suggested knowledge representation formalism, the structure and the contents of the knowledge base of STRAMS and the implemented mathematical problem solving and learning mechanisms.

Keywords: mathematical knowledge representation, problem solving, intelligent computer algebra system

1995 Math. Subject Classification: main 68T30, secondary 68T35

1. INTRODUCTION

In the last three decades Computer Algebra Systems (CAS) have been widely used in the automation of scientific computation and design. These systems can help in the solution of various problems connected with the execution of complicated and labour-consuming transformations of mathematical expressions. However, irrespective of their good capabilities, "classical" CAS like Reduce, Maple, Mathematica etc. are sometimes difficult for use. The most serious problem here [2, 3] is that "classical" CAS behave as black boxes and therefore the interpretation of the suggested solutions can call for significant efforts.

The reason for this problem is that "classical" CAS have no mathematical knowledge represented in an explicit, declarative way. Their knowledge is embedded

implicitly in the algorithms and is inaccessible to the user.

Therefore a series of successful attempts have been made in order to build various kinds of the so-called intelligent CAS. 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. Because of adequacy and efficiency considerations, intelligent CAS usually are hybrid [1, 4] by means of combining several formalisms and paradigms.

A set of projects aimed at the investigation of different aspects of the integration of the classical approaches for developing CAS with Artificial Intelligence methods and tools have been under development at the Faculty of Mathematics and Informatics, Sofia University. An approach to building intelligent CAS has been developed [5]. The first version of a knowledge-based tool for developing CAS called KAM [6] has been implemented. The experimental intelligent computer algebra system STRAMS discussed in this paper has been under development using KAM. In general, STRAMS is a knowledge-based CAS that can solve various types of mathematical problems, learn and explain the results of its work. The approach used for the development of STRAMS and the major features of this system are described and argumented in details in [5, 6]. Therefore we emphasize here on the analysis of the architecture of STRAMS, the contents of its knowledge base and the implemented mathematical problem solving mechanisms.

2. FUNCTIONAL FACILITIES AND ARCHITECTURE OF STRAMS

STRAMS is a general-purpose intelligent CAS. The definition domain D of the expressions that can be manipulated in STRAMS includes all expressions containing numbers, symbols and the functions: $+$, $-$, $*$, $/$, power function, exponential, logarithmic and trigonometric functions. STRAMS is intended for solving the following main problem types:

— simplification (reduction to a canonical form) of expressions from D;

— symbolic equation solving (solving equations of the form $expr_1 = expr_2$, where $expr_1$ and $expr_2$ are expressions from D);

— symbolic differentiation of expressions from D;

— symbolic integration (formal integration of functions belonging to a particular subset of D).

The architecture of STRAMS is determined by its functional facilities and some additional design requirements like transparency and learning and explanation generation capability. The architecture of the environment KAM used as a tool for the implementation of STRAMS exerts a considerable influence as well.

STRAMS includes the following functional components: a mathematical problem solving engine, an explanation module, an interface module, a control block.

The architecture of STRAMS is shown in Fig. 1.

The mathematical problem solving engine consists of two modules: a knowledge engine and a learning module realizing respectively the problem solving and the

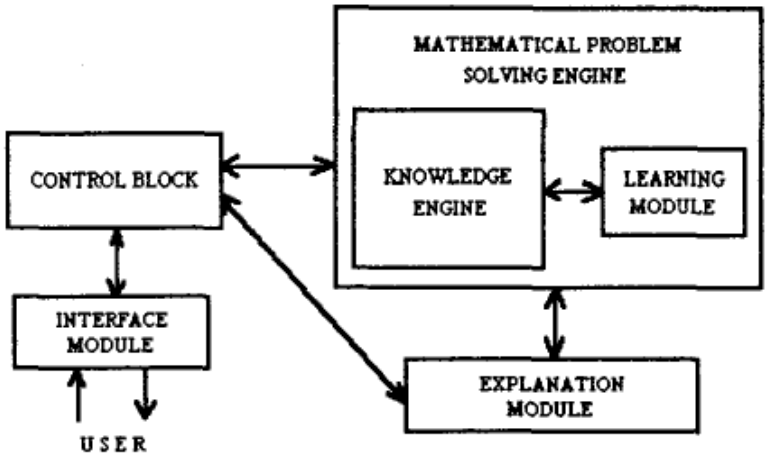

Fig. 1. Architecture of STRAMS

learning capabilities of STRAMS. The structure and the functioning mechanisms of the mathematical problem solving engine are discussed in Section 4 of this paper.

The explanation module realizes the explanation generation capabilities of STRAMS. These capabilities are described in [5, 6] and will not be discussed in this paper.

The interface module is the component of STRAMS the users are in touch with. It analyzes the user requests, converts them into the corresponding internal form and sends appropriate messages to the control block. The current version of the interface module provides only some relatively primitive communication means that will be improved in the further versions.

The control block realizes the general control of the system's work and the interaction between the other components.

3. KNOWLEDGE REPRESENTATION IN STRAMS

The formalism supported by the environment KAM is used for the knowledge representation in STRAMS. It is oriented to the description of knowledge about the properties of the manipulated functions and the methods for mathematical problem solving defined by these properties. This formalism is described and analyzed in details in [5, 6]. Here we present it in brief and give appropriate examples.

The knowledge of STRAMS about the properties of the manipulated functions is described using a special type of rules called rewrite rules. The structure of each rewrite rule includes the description of a correct transformation of a class of mathematical expressions and the formulation of some general preconditions for its performance (if there are any). Examples of rewrite rules:

$(x+y)(x-y) = x^2 - y^2$

$e^x e^y = e^{x+y}$

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

The description of the methods for transformation of the expressions and equations STRAMS can manipulate is realized by the so-called generalized rules

(methods). Each generalized rule describes a sequence of transformations of the given expression aimed at its conversion into a particular form. In this sense usually generalized rules contain sequences of properly grouped rewrite rules. More precisely, 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. The body of a generalized rule may contain:

— a sequence of rewrite rules. Each of them can include some additional control information. In this case the generalized rule is called *declarative*;

— the code of a procedure realizing the application of the rule. Such generalized rules are called *procedural*;

— a set of pairs (*pattern*, *procedure*) such that when the examined expression matches one of the patterns, the corresponding procedure is executed. These generalized rules are called *hybrid*.

Declarative generalized rules are most numerous in STRAMS. As it was mentioned above, 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. In this sense 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 (reduction to a canonical form). As an example of such a method we can examine the transformation called expansion.

This transformation can be defined by the equality

$$\begin{aligned}(x_1 + x_2 + \cdots + x_m)(y_1 + y_2 + \cdots + y_n) = {} & x_1y_1 + x_1y_2 + \cdots + x_1y_n \\ & + x_2y_1 + x_2y_2 + \cdots + x_2y_n \\ & \cdots\cdots\cdots\cdots \\ & + x_my_1 + x_my_2 + \cdots + x_my_n.\end{aligned}$$

It is described in STRAMS by a cyclic recursive generalized rule with a body containing the following basic rule:

$$\prod_{i=1}^{n} A_i(x_1 + x_2 + \cdots + x_m) \prod_{j=1}^{k} B_j = \prod_{i=1}^{n} A_i x_1 \prod_{j=1}^{k} B_j + \prod_{i=1}^{n} A_i(x_2 + \cdots + x_m) \prod_{j=1}^{k} B_j.$$

Another classification criterion of the generalized rules is the role they play in the problem solving process of a given, relatively complex task (such tasks in STRAMS are equation solving and symbolic integration). In this sense they can be 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. In the role of examples of key and non-key generalized rules we give here the descriptions of two generalized rules included in the knowledge base of the 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 execution of the body of the generalized rule consists in the application of the inverse of f to $expr_1$ and $expr_2$. The precondition of the generalized rule is: the unknown occurs in only one of the arguments of f and $expr_2$ does not contain the unknown. The goal is to remain in the left-hand side of eq only the argument containing the unknown.

This generalized rule is a key one and is implemented procedurally due to effectiveness considerations.

Example 2. *Collection.* The goal of this generalized rule is to reduce the number of occurrences of the unknown. Collection is a non-key generalized rule with no explicit precondition. STRAMS applies it only if none of the key generalized rules can be applied. So the precondition of Collection (and of all non-key generalized rules) is: there is no key generalized rule with satisfied preconditions.

This generalized rule is declarative, normal. One of its rewrite rules is:

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

The knowledge of STRAMS about the problem solving methods for the included types of tasks is 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 (equation solving or symbolic integration). For a more precise definition of the concept of a schema one can use the following additional considerations:

- each schema is a sequence of at least two non-key generalized rules;

• each schema begins either with the first generalized rule used in the problem solving process or with a generalized rule applied after the application of a key generalized rule;

• after the application of a schema either the corresponding problem is found to be solved or a key generalized rule can be applied.

Thus schemata are a natural generalization of 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.

4. STRUCTURE AND FUNCTIONING MECHANISMS OF THE MATHEMATICAL PROBLEM SOLVING ENGINE

As it was mentioned in Section 2, the mathematical problem solving engine of STRAMS consists of two modules: a knowledge engine and a learning module. The knowledge engine includes the so-called inference control block and the following processing subsystems:

— a simplification subsystem;
— an equation solving subsystem;
— a symbolic differentiation subsystem;
— a symbolic integration subsystem.

The structure of the knowledge engine is shown in Fig. 2.

The processing subsystems realize the main functional facilities of STRAMS listed in Section 2. Each of these subsystems is a relatively autonomous knowledge-based system with its own knowledge base and problem solving program. The typical structure of the processing subsystems of STRAMS is presented in Fig. 3.

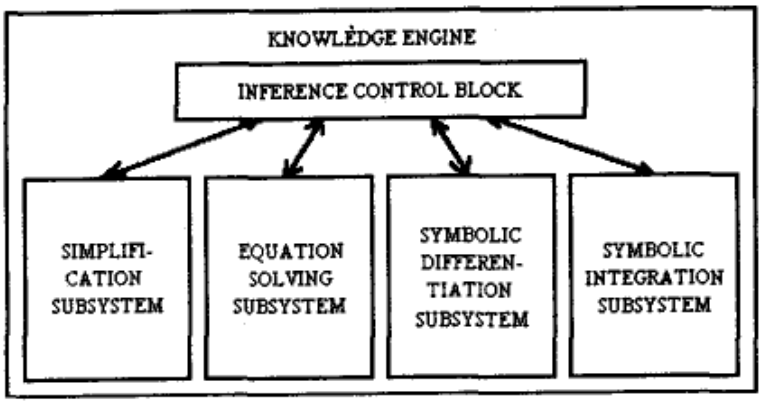

Fig. 2. Structure of the knowledge engine

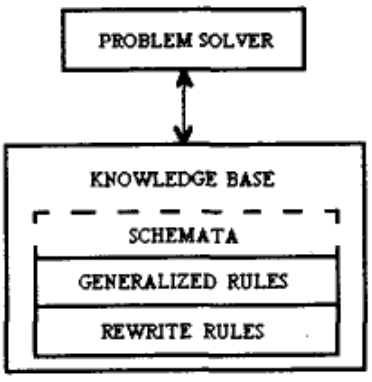

Fig. 3. Structure of the processing subsystems of STRAMS

The knowledge base of each processing subsystem includes the set of generalized rules and rewrite rules that have been used in solving the corresponding type of problems. Additionally, the knowledge bases of the equation solving subsystem and the symbolic integration subsystem contain the corresponding schemata accumulated by the learning module of STRAMS during the system's work.

The problem solver of each processing subsystem realizes the search in the state space of the current problem of the corresponding type. This problem can either be formulated by the user or be generated by some of the processing subsystems. In the role of operators in the state space search the problem solvers use the schemata and generalized rules available at the moment. The application of the chosen generalized rules is performed by the generalized rule interpreter supported by the environment KAM. Some additional search control knowledge has also been used by the problem solvers. It is formulated as a result of some experiments carried out with the particular processing subsystems.

The inference control block realizes the interaction between the knowledge engine and the learning module of STRAMS. The second main function of the inference control block is to manage the interaction between the particular processing subsystems (for example, all processing subsystems generate canonization problems that are solved by the simplification subsystem).

In terms of the functioning mechanisms of the mathematical problem solving engine, the method of work of the problem solvers is most interesting. The particular problem solvers are adjusted copies or simplified versions of one and the same prototype (the control block of the inference engine of KAM [6]). Therefore they perform modifications of one and the same algorithm. The differences are in the kind of the used operators (only generalized rules or schemata and generalized rules) and in the form of the used search control knowledge.

Let us consider as an example the method of work of the problem solver of the equation solving subsystem. There are at least two reasons causing our special interest to this subsystem:

- its problem domain is appropriate for the application of the schemata formalism. Therefore it can do a kind of learning based on the capability for discovering and memorizing the schemata used in the problem solving process;
- it is well known [7] that the state space of some of the types of equations admissible in STRAMS is enormous (includes of the order of 10^{10} states). Therefore the use of some strategic knowledge in order to avoid the exhaustive search is necessary from the point of view of the practical applicability of STRAMS.

The discussed problem solver uses for search control purposes a special evaluation function *Complexity(eq,var)*. *Complexity(eq,var)* is a linear combination of the number of occurrences *VarOccur(eq,var)* of the symbol *var* in the equation *eq* and the sum of the nesting depths *CommonVarDepth(eq,var)* of *var* in *eq*:

$Complexity(eq, var) = c_1 VarOccur(eq, var) + c_2 CommonVarDepth(eq, var)$.

This function is used in the examination of all equations. Initially, the value of *Complexity(eq,x)*, where *eq* is the given equation and x is the unknown, is computed and the variable *initial_complexity_factor* gets this value:

initial_complexity_factor = *Complexity(eq,x)*.

Then the problem solver does not explore all equations eq' in the state graph of *eq* that do not satisfy the so-called simplicity criterion:

f(steps_done). Complexity(eq',x)-initial_complexity_factor
$< c_3$.*initial_complexity_factor*,

where *steps_done* is the number of transformations reducing *eq* to *eq'*. In this way the simplicity criterion plays the role of a heuristics for pruning a part of the state graph of the current equation in order to avoid the exhaustive search.

Another heuristics used for search control purpose states that the application of a key generalized rule as an operator can significantly shorten the path to the solution. Therefore, when a key generalized rule is applied at a given step, the discussed problem solver continues its work with the exploration of the equation obtained as a result of the application of this generalized rule. If no key generalized rule can be applied at the current step, the problem solver looks for a proper schema leading to the applicability of a key generalized rule.

The general form of the function *Complexity(eq,var)* and the simplicity criterion, the definition of the function $f(n)$ and the concrete values of the parameters c_1, c_2, c_3 are suggested in [9].

Let us assume that an equation *eq* has to be solved with respect to the symbol *var*. During its working cycle the discussed problem solver supports a list of equations belonging to the state graph of *eq* that have to be explored. We shall refer to this list as *eq_list* and to its first element as *current_eq*. Then the algorithm of work of our problem solver can be formulated in general as follows.

S1. Initialize *eq_list* to the list containing only *eq*. Initialize *initial_complexity_factor* to *Complexity(eq,var)*.

S2. If *eq_list* is the empty list, then report failure and quit.

S3. If the equation *current_eq* is solved, then return *current_eq* and quit.

S4. If *current_eq* does not satisfy the simplicity criterion, then remove *current_eq* from *eq_list* and go to S2.

S5. If a key generalized rule is applicable to *current_eq*, then replace *current_eq* in *eq_list* by the equation obtained as a result of the application of the found generalized rule to *current_eq*. Go to S3.

S6. If an existing schema is applicable to *current_eq*, then modify *eq_list* by analogy with S5 and go to S3.

S7. Remove *current_eq* from *eq_list* and add to the end of *eq_list* the equations that can be produced by the application of all non-key generalized rules to *current_eq*. Go to S2.

Whenever an equation is successfully solved, an attempt for the extraction of new schemata is made. For that purpose the inference control block activates the learning module of STRAMS. The learning module analyzes the used sequence of generalized rules, constructs the new schemata candidates (in accordance with the definition of the schema concept) and merges them with the set of existing schemata. In this way STRAMS does a kind of unsupervised learning by accumulation in the corresponding knowledge base of new, successfully applied schemata that can be used in its further work.

5. IMPLEMENTATION OF STRAMS

The implementation of STRAMS has been realized using the environment KAM. The program modules of the mathematical problem solving engine of STRAMS are either exact copies or simplified versions of some of the program modules of KAM. The knowledge bases of the processing subsystems of STRAMS are built by direct recording of the corresponding rewrite rules and generalized rules in internal form. The knowledge base of the simplification subsystem includes rewrite rules and generalized rules described in [8, 9], and the knowledge base of the equation solving subsystem includes rewrite rules and generalized rules described in [9].

The explanation module of STRAMS is a copy of the module of the same name of KAM. The interface module and the control block of STRAMS are developed especially for the purpose. All program modules of STRAMS are written in Common Lisp.

6. SUMMARY AND CONCLUSION

STRAMS is a knowledge-based CAS with the following main features:

- it can solve various types of problems using a set of methods and techniques, traditionally taught in the secondary school and in the introductory university courses;
- it is able to do a kind of learning and explanation generation;
- it can easily be integrated with other software packages;
- its functional facilities can easily be extended.

These features of STRAMS determine its potential applicability in building expert systems, intelligent tutoring systems etc.

Our current activities are directed to the improvement of the user interface of STRAMS and to the extension of its functional facilities.

ACKNOWLEDGEMENTS. This work is partly funded by the Sofia University SRF under Contract No. 274/1997.

REFERENCES

1. Calmet, J., K. Homann, I. Tjandra. Hybrid Representation for Specification and Communication of Mathematical Knowledge. In: K. Homann, S. Jacob, M. Kerber, H. Stoyan (Eds.), *Proceedings of the Workshop on Representation of Mathematical Knowledge*, 12th European Conference on Artificial Intelligence, Budapest, 1996.
2. Homann, K., J. Calmet. Combining Theorem Proving and Symbolic Mathematical Computing. *LNCS*, **958**, Springer-Verlag, 1995, 18–29.
3. Homann, K., J. Calmet. Structures for Symbolic Mathematical Reasoning and Computation. *LNCS*, **1128**, Springer-Verlag, 1996, 216–227.

4. 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.
5. Nisheva-Pavlova, M. A Knowledge-Based Approach to Building Computer Algebra Systems. In: *Proceedings of JCKBSE'96*, Sozopol, 1996, 222–225.
6. Nisheva-Pavlova, M. KAM — A Knowledge-Based Tool for Developing Computer Algebra Systems. *Ann. Sof. Univ., Fac. Math. and Inf.*, **90**, 1996 (to appear).
7. Silver, B. Precondition Analysis: Learning Control Information. In: R. Michalski, J. Carbonell, T. Mitchell (Eds.), *Machine Learning*, Vol. 2, Morgan-Kaufmann, 1986, 647–670.
8. Todorov, B. An Environment for Building Special-Purpose Knowledge-Based Systems for Computer Algebra. MSc Thesis, Sofia University, 1994 (in Bulgarian).
9. Vatchkov, V. Learning in Equation Solving. MSc Thesis, Sofia University, 1993 (in Bulgarian).

Received March 4, 1998
Revised May 12, 1998

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