An Expert Help System for Computer Algebra Systems

R. P. dos Santos[*] and W. L. Roque[†]
Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5 – D7500 Karlsruhe 1 – FRG
Phone: (+49) 721 6084328
E-mail: kg07@dkauni2.bitnet and kg03@dkauni2.bitnet

Abstract

In this paper we discuss the main features of *Smart Help*, an expert help system for aiding users of Computer Algebra Systems. *Smart Help* is implemented in the knowledge representation system shell MANTRA, a hybrid system which supports the logic, frame, semantic networks and production rules knowledge representation methods. Presently, the *Smart Help* domain knowledge base concerns REDUCE.

AI topic: Design.

Domain area: Computer Algebra.

Language/Tool: Lisp/MANTRA.

Status: Development status.

Impact: An intelligent on-line help system for the Computer Algebra community and a contribution towards Intelligent Computer Algebra Systems.

1 Introduction

In the last twenty years very powerful, sophisticated computing systems have become available for helping engineers, mathematicians and scientists in doing their hard professional calculations. But to profit from them they need to know in addition to the mathematical concepts involved also the capabilities of these computing systems and how to operate them. On the other hand, having access to these systems one can avoid exploring the current bibliography looking for the best, latest and more sophisticated mathematical techniques required to perform the calculations since many of these have been already implemented in by the algorithm experts and systems designers. These new systems are known as Computer Algebra Systems (CAS).

REDUCE[1, 2], like many modern computer algebra systems (MACSYMA, MATHEMATICA, MAPLE, SCRATCHPAD to name a few), embodies a lot of mathematical knowledge which is spread through thousands of procedures. All this knowledge is, however, in an *implicit* form almost inaccessible to the user, who would have to decipher the source files to recover it. In fact, beginners are normally more interested in something like "What do I need to type to have my integral done?" instead of "What is the available character set?". Or very likely also in "Why the result I got is so ugly in comparison to the one I found in the tables?" and not in "How is the expression internally stored?".

The process of learning how to use a computer algebra system may be done reading throughout the manual available for the system, a sort of user's guide or in the most confortable and easiest way: consulting an *expert* on the system. The beginners, in general, do not want to waste his time reading either a book or manual looking for the terminological structure of the system. Particularly, to its semantics, syntax and/or examples just to be able to start using the system to solve a simple problem: an integral, for instance.

These kinds of informations can be easily provided by na on-line help system (and some progress in this direction have been attained[3]) or even better, by an Intelligent Tutorial System. Therefore, any one of these facilities would be most wellcome. In fact, the actual need for a good help facility was realized by ourselves as well as by many collegues and students when we first faced a CAS for non-trivial applications.

The idea behind a help system is that existing a

[*] On leave of Centro Brasileiro de Pesquisas Físicas, Rua Xavier Sigaud, 150, 22290 Rio de Janeiro, RJ, Brazil.

[†] On leave of Universidade de Brasília, Instituto de Ciências Exatas, 70910 Brasília, DF, Brazil.

stored knowledge base, consisting of a lot of information elements, structured in levels of specialization, an *expert help system* could generate by inference (not just by pure recovering) an information, which might not be explicity stored and that might be an adequate (in terms of abstraction, difficulty, detail, etc.) directive to the user. An intelligent help facility could reason about the user's query and then give a reasonable reply or else, suggest what has to be consulted or even what to do next.

It is our intention here only to discuss the nature, complexity and tools concerning the design of *Smart Help*, na expert help system with these features. Presently, the *Smart Help* domain knowledge base concerns REDUCE, however it may well be used to implement different CAS bases. Since the hybrid knowledge representation system (KRS) MANTRA[4, 5] can be seen as a knowledge representation shell, one could make use of its (formal) reasoning facilities to build the *Smart Help* system.

The ideas forwarded herein are in a prototypal basis, but we hope that they will raise the interest of the CAS community and evolve to reach at the end the full system implementation. In section 2 we describe briefly the computer algebra system REDUCE and propose a taxonomy for the knowledge embodied by this system. In section 3 we describe the knowledge representation system MANTRA. Section 4 is concerned with the *Smart Help* system design. Finally, in section 5, we give some conclusions to the paper and comment further upgrades which would turn *Smart Help* into a truly *Intelligent Tutorial System.* In the appendix, an example of the interaction is given.

2 The CAS REDUCE

REDUCE[1] is a fairly powerful system for carrying out a variety of mathematical calculations such as to manipulate polynomials in a variety of forms, simplify expressions, to differentiate and integrate algebraic expressions, do some modern differential geometry calculations, study the Lie-symmetries of systems of partial differential equations, and many others.

The system is made out of different knowledge domains (mathematical, terminological, computational, etc.). For our purposes here we classify the specific knowledge domain involved in operating REDUCE in the following categories:

syntax: Informations about the number and type of the arguments, and requisites and prerequisites of a REDUCE command, declaration or operator (that is the only type of information given by many help systems).

terminology: Definitions of the various terms like *identifier*, *operator*, *kernel*, etc., employed in documents related to REDUCE and, also important, in error messages from REDUCE (this matter can be quite confusing to the initial user).

concepts: Informations like the fact that *integration* in REDUCE is represented by an operator called **INT**, which is (presently) applicable only to scalars, or perhaps a reference for the algorithm implemented in a specific facility.

procedures: General sequences of commands which should be given to perform a certain task like defining a new infix operator: one has to declare the operator infix through the INFIX declaration and then give him a precedence by means of the PRECEDENCE declaration.

heuristics: General rules and tips that simplify and improve approachs to problem-solving (this kind of information comes with experience and is one of the most frequent in consultations from beginners). For example, "If you want to compute a definite integration, you can try evaluating the indefinite integral, saving the resultating expression in a variable and then substituting locally the limits of integration in it and finally subtracting the results".

All these different forms of knowledge require one or more knowledge representation formalism and the best outcome may only be found through the integration of various knowledge representation methods.

3 The KRS MANTRA[1]

MANTRA[4, 5] is a hybrid knowledge representation system which integrates three different knowledge representation methods:

Four-valued first-order logic language, used to express *assertional knowledge*, which is decidable[6] but presents a weaker entailment mechanism which excludes the chaining of independent facts, thus ruling out *modus ponens* but allowing quantifiers.

Terminological language [7] (a kind of *frame* method), extended to allow the definition of concepts and n-place relations over themselves.

[1]This description of the MANTRA System is based on [5]

Semantic network, which is a representation to define hierarchies with exceptions[8].

One could think that classical logic would be able to generate all knowledge entailed (i.e., implicity represented) by a given concept. Unfortunately, however, the systems based on the complete first-order logic, suffer from the inherent problem of *Combinatorial Explosion.* Also, the entailment problem is not decidable in first-order logic. In addition, the knowledge to be represented is often incomplete and incoherent.

Due to these facts, the knowledge representation system MANTRA has been designed associating the three knowledge representation methods above, based on a four-valued approach. The common four-valued semantic associated to these associated allows the modeling of the aspects of *ignorance* and *inconsistency*, useful for representing incomplete and/or incoherent knowledge.

MANTRA's architecture consists of three levels:

Epistemological level, where the knowledge representation methods are defined.

Logical level, where the concept of knowledge bases and the primitives to manipulate them are defined.

Heuristic level, where a production system can be defined through primitives allowing the specification of *ad hoc* inference steps.

At present only the epistemological and logical levels are implemented in the system and the heuristic level is at some extent being simulated by the supporting Lisp language itself.

MANTRA has also two interface modules which provide an *interactive* or a *programming* environment for the user. The latter is interesting as it allows commands mixing MANTRA and Lisp syntaxes and doing so, it allows one to build a production system from scratch or to adopt one already written in Lisp.

4 The *Smart Help* Expert System

This section is concerned with the identification, conceptualization and formalization of the knowledge involved in building an expert help system that can answer questions from the user of REDUCE concerning its terminology, structure, semantics, and, at the lowest level, also its syntax, in much the same way as an expert colleague would do.

4.1 The Conception of *Smart Help*

The conception of *Smart Help* follows the tradition of help systems being passive. That means that the user learns how to use REDUCE playing freely with it without being interrupted by the *Smart Help.* It looks like a normal REDUCE session but with MANTRA running behind. When the user gets a confusing answer or a meaningless error message (and in fact REDUCE users know how often this happens), or even when he does not know what to do next to get his calculations done, he invokes the *Smart Help* to clarify the point.

When queried by the user about a topic, *Smart Help* will work as a *problem-solver* searching in the state space, defined by all related concepts, untill the system decides that enough knowledge was recovered so that an adequate answer can be expressed. Afterwards it structures these units of knowledge in the form of a readable, understandable and enlightening, answer.

The problem might have been caused by earlier mistakes (a forgotten variable definition long before, for example). *Smart Help*, however, is not able to trace the session history to find the exact place at which the misconception first had its effect. It is then left to the user to do the right question to get the right answer. That is, progress can be made only if, from the correct definition, usage, prerequisites, etc., of the queried topic, as returned by the *Smart Help* (explanation), the user can pinpoint the misconception which caused the problem.

4.2 The Representation of REDUCE's Knowledge in *Smart Help*

Considering the taxonomy of the knowledge embodied by REDUCE, presented in section 2 above, and the knowledge representation methods available in MANTRA, as described previously in section 3, we had to find the best fit of both to guarantee efficiency in recovering knowledge from the base and in reasoning with it, according to the inherent structure of each knowledge category and its to adaptiveness to the specific representation method.

The mapping used to match the knowledge categories to the methods and vice-versa are as follows:

- **from knowledge types to representation methods**

  syntax: Syntactical descriptions of operators, statements, etc., were represented making use of the *frame* method. The *slots* provide a natural way to indicate the number and type of arguments as *expectation values.* For example, the syntax of the integration operator could be defined as

  integration := $\exists$ *name*:[INT] $\sqcap$

$\exists$ *argument*:
[$\forall$ (*argument*:[*first*]):[*expression*] $\sqcap$
$\forall$ (*argument*:[*second*]):[*variable*]]

terminology: Terminological definitions have been represented very conveniently through the *frame* method available in MANTRA as a terminological language. Each *concept* (e.g., *right-operator*) is defined as a *specialization* of a more generic one (e.g., *infix-operator*) by means of *slots* that are filled with the characteristics which specify the former with respect to the last one (e.g., the *flag* RIGHT set). On the other hand, some concepts (e.g., *left-operator*) can be seen as *defaults* to the more generic one and have no distinguishing characteristic from it; in this case, the concept is defined by a default link "KIND OF" connecting them according to the *semantic network* method. For example, these two concepts were entered in the knowledge base through the following statements

right-operator := *infix-operator* $\sqcap$
$\exists$ *flag*:[RIGHT]
kind-of :=
left-operator $\rightarrow$ *infix-operator*

concepts: The conceptual knowledge has been represented by associating concepts to conceptual definitions and to conceptually related ones. These associations were accomodated in a *semantic network* of hierarchies where each hierarchy represents a type of link between concepts (e.g., IS REPRESENTED BY, IS RELATED TO, etc.). For instance, the examples presented in section 2 above have been entered into these hierarchies as

is-represented-by :=
integration $\rightarrow$ INT
is-related-to :=
integration $\rightarrow$ *derivation*

procedures: The procedural knowledge has been represented as logic expressions consisting in a left-side and a right-side and have been stored in a hash-table. The left-side defines the context (usually the queried topic) in which the procedure, which is defined itself in the right-side, is pertinent. The recovering of this knowledge consists in searching the hash-table for entries whose left-side contain the present context and getting its right-sides. If succeded, the remaining elements of the left-side are passed to the user as conditions of applicability of the procedure. For instance, the example presented in section 2 was codified as follows

{(*define infix-operator*),
((*declare* (*infix operator*))
(*declare precedence*))}

heuristics: The heuristical knowledge has also been represented as logic expressions consisting in a left-side and a right-side but stored in a different hash-table. The recovering of this knowledge is done the same way as the *procedural* knowledge above. For example,

{(*definite integration*),
((*do indefinite-integration*)
(*save-result-in variable*)
(*locally-substitute values*))}

- **from representation methods to knowledge types**

logic: We used this method to represent specific attributes of *terminological* and *syntactical* nature of individual entities of REDUCE.

frame: The type of frame representation implemented in MANTRA led us to make use of this method to represent part of the *terminological* knowledge as well as the *syntactical* knowledge as defined in the taxonomy of section 2 (see examples above under *syntax* and *terminology*).

semantic network: We used this method as a kind of associative memory, useful to store the *conceptual* knowledge of REDUCE by linking related concepts in an efficient way. As we said before, the remaining *terminological* knowledge, which was not fitted to frame was also stored making use of this method (see again examples above under *syntax* and *terminology*).

4.3 The Architeture of *Smart Help*

Technically, *Smart Help* is a Production System on the top of a particular implementation of MANTRA which has REDUCE integrated as an additional knowledge representation module (see Fig. 1[2]). Since the heuristic level of MANTRA has not yet been implemented, being presently represented by the Lisp language itself, *Smart Help* is coded in Lisp and resides in the

[2]The architeture of *Smart Help* is shown here in more detail than MANTRA or Reduce since it is the main concern here. Better descriptions of both can be found in [4, 5] and [1] respectively.

Figure 1: The Structure of *Smart Help*

same Lisp session of MANTRA. To the user it looks like a normal REDUCE session but MANTRA is running behind and is accessible as an operator, by means of the interface MANTRA-REDUCE.

The domain knowledge base was defined as an *object* by means of the object-oriented extension of Common Lisp called CORBIT[9]. The five knowledge types defined in section 2 were implemented as *aspects* of the knowledge base object being also defined as *objects*. Each knowledge type object has its own recovery procedure. The mapping of the *terminology*, *syntax* and *concept* categories to the MANTRA's knowledge representation methods, as described above, was then easily implemented in the "mantra interface" block. The *procedure* and *heuristic* types map directly into Lisp hash-tables by the "kbase-management" block.

The knowledge recovered during the process of a query is stored in a Lisp structure called "answer" in the figure above. It possesses partitions corresponding to the five categories of knowlege. This structure is periodically inspected by the "query processor" unity in the process of adequating the answer to the user's needs, as described below. The partition corresponding to concepts is by its side defined as another structure possessing partitions corresponding to the various links (IS-REPRESENTED-BY, IS-RELATED-TO, etc.) mentioned above concerning to the mapping to MANTRA.

That all allows the easy manipulation of the recovered knowledge in a very structured way. It also makes simple the implementation of the mapping from REDUCE knowledge types to MANTRA knowledge representation methods.

4.4 The production system rules

A number of rules were implemented in the production system of *Smart Help*. We present them in the following. Presently, they are inserted in the code itself. We consider now to get them defined in a production rule base, what would give flexibility and clearness to our system.

When asked by the user about a topic, *Smart Help*

1. Assumes that the present level of familiarity of the user with REDUCE (student model) can be inferred from the level of specification of the queried topic, which is characterized by a numeric heuristical parameter, ranging from 1 to 3, associated to it. For example if someone queries about "integration" (very general – parameter = 1) it seems probable to be a very novice user but if one queries about "infix-operators" (more specialized – parameter = 2) it should be considered as an user with some familiarity. As it was mentioned before, *Smart Help* does not keep a history of previous queries and as a consequence we were not able to imagine a better way of estimating the user's familiarity with REDUCE.

2. Queries the domain knowledge base, to recover all informations related to the given topic. This process consists in the following steps:

    (a) To query the *conceptual aspect* about the topic in an attempt to define it conceptually. Here it is also prepared a list of subsumed topics to be given to the user as a suggestion for further queries. (This is another opportunity to take care of the student's model)

    (b) To query the *terminological aspect* about the topic in an attempt to define it terminologically.

    (c) To query the *syntactical aspect* about the topic in an attempt to recover its syntax of usage.

    (d) To query the *procedural aspect* about the topic in an attempt to recover procedures associated to it.

(e) To query the *heuristical aspect* about the topic in an attempt to recover heuristical rules associated to it.

3. Evaluates the level of generality of the terms recovered to see if they are in the same level of specialization (same value of the parameter). If a concept is too specific (much greater value of the parameter), *Smart Help* tries to redefine it in terms of more general concepts. (The idea here is to adequate the answer to the user's needs, stimulating him, on the other hand, to proceed with a deeper investigation into the system. Also it is an opportunity to circumvent a possibly inadequate evaluation of the user's needs). If a concept is too general, however, it is simply deleted.

4. Formats the recovered knowledge in the form of a readable answer, defining the queried concept and its usage in terms of the recovered knowledge. Presently this process is quite crude as it is a peripherical point of the implementation. It will be improved latter on.

5. Prints the answer returning control to REDUCE.

5 Conclusions

We have presented and proposed in this paper a fairly general design of an expert help facility for aiding users of Computer Algebra Systems. Although the expert help system presented here has been particularly oriented to REDUCE (as a consequence of our former experience with this system), we point out that the concept of *Smart Help* can be extended to other Computer Algebra Systems.

The reasons for introducing *Smart Help* facility include:

- It will provide an on-line help for the system, aiding the users to find specific informations about the system terminology, structure, syntax, etc.

- It will allow the potential user to access the whole capabilities of the system.

- It can contribute to the development of Intelligent Computer Algebra Systems, which more than simply being able to do calculations, could interact with the user and free him of many details concerning the specification of his problem.

The *Smart Help* has no intention to *teach* the user how to program efficiently in REDUCE or behave like a tutoring system. However, it can be used in the learning process as a complimentary teaching tool.

Following the ideas addressed in this paper, we intend afterwards to develop an *Intelligent Tutorial System* (ITS) for REDUCE. The ITS should inherit all the compatible facilities already available in the *Smart Help*. In addition, many other facilities would become available, such as, a deeper understanding of REDUCE's semantics, keeping track of history, a dynamical reasonning on the student's model, explanations, etc.

A prototype of *Smart Help* is now running on a SUN work-station on an experimental basis. The full implementation of *Smart Help* as a final product was not our main concern here. This task will certainly need few more people working in a close colaboration to build up a satisfactory knowledge base to reach at the end the principal objective that is helping a CAS user.

We would like to thank Prof. J. Calmet for enlightening discussions and for the warm hospitality provided by him and by all members of his group.

A Example

For better understanding, suppose that an user with no experience with REDUCE wants to do some calculations, for instance, integrate an expression in terms of a certain variable. Having access to an initialized session of REDUCE (in which the heading shows how to invoke *Smart Help*), the interaction would consist in typing `shelp integration` , and *Smart Help* would promptly reply:

```
"SHelp - Version 2.0:  23 Aug 1990"
INDEFINITE-INTEGRATION is the default
   for INTEGRATION.
INTEGRATION is represented in Reduce
   by INT.
Its syntax is:
INT(scalar-expression,variable)
Ex.:  INT(LOG(X),X);
INT is implemented through the
SIMPINT-PROCEDURE-IN-INT-SOURCE-FILE.
For DEFINITE INTEGRATION, one may try
to DO INDEFINITE-INTEGRATION,
to SAVE-RESULT-IN VARIABLE,
to LOCALLY-SUBSTITUTE VALUES.
References:  REDUCE MANUAL SECTION
   7.4.
See also:  DERIVATION
```

References

[1] HEARN, Anthony C., REDUCE User's Manual: Version 3.3, RAND Publication CP78, The Rand Corporation, Santa Barbara, Calif., 4/1987.

[2] MacCALLUM, M.A.H., and WRIGHT, Francis, REDUCE Lecture Notes, *in* REBOUÇAS, M.J. (ed.), proceedings of the I Brazilian School on Computer Algebra, vol. I, Oxford University Press (forthcoming book), 1990.

[3] HARPER, David, Reduce Forum, 23/8/1989; SCHOEPF, Rainer M., Reduce Forum, 24/8/1989; LAMBE, Larry A., Reduce Forum, 24/8/1989; AGER, Tryg, Reduce Forum, 24/8/1989; DEWAR, Mike, Reduce Forum, 24/8/1989; MARTI, Jed, Reduce Forum, 2/8/1990; WRIGHT, Francis, Reduce Forum, 2/8/1990; COPELAND, Gary, Reduce Forum, 2/8/1990.

[4] BITTENCOURT, Guilherme, The MANTRA Reference Manual, Interner Bericht 2/90, Univ. Karslruhe, Fak.f.Informatik, Karlsruhe, West Germany, 1/1990.

[5] BITTENCOURT, Guilherme, An Architeture for Hybrid Knowledge Representation, PhD Thesis, Univ. Karslruhe, Fak.f.Informatik, Karlsruhe, West Germany, 31/1/1990.

[6] PATEL-SCHNEIDER, P.F., A Decidable First-Order Logic for Knowledge Representation, *in* IJCAI 9 (proceedings of International Joint Conference on Artificial Intelligence, Los Angeles, Calif., 8/85):455–458, Morgan Kaufmann Publishers, Inc., Palo Alto, Calif., 1985.

[7] BRACHMAN, Ronald J., and LEVESQUE, Hector J., The Tractability of Subsumption in Frame-Based Description Languages, *in* proceedings AAAI-84 (Fifth National Conference on Artificial Intelligence, Austin, Texas, 1984), Morgan Kaufmann Publishers, Inc., 1984.

[8] HORTY, J.F. and THOMASON, R.H. and TOURETZKY, D.S., A Skeptical Theory of Inheritance in Nonmonotonic Semantic Nets, Technical Report CMU-CS-87-175, Carnegie-Mellon University, Computer Science Department, Pittsburgh, Pa., 10/1987.

[9] De SMEDT, Koenrad, Object-Oriented Programming in FLAVORS and CommonORBIT, *in* HAWLEY, R., (ed.), Artificial Intelligence Programming Environments: 157–176, Ellis Horwood Limited, 1987.