The Architecture of Smart Help: 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) 721 608-4328
BitNet: kg07@dkauni2 and kg03@dkauni2

Abstract

We present here the architecture, operation and tools concerning the design 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.

1 Introduction

In the last twenty years very powerful and sophisticated computing systems, known as Computer Algebra Systems, have become available for helping engineers, mathematicians and scientists in doing their hard professional calculations in different fields[1]. These systems are having also a great impact on education[2]. But to profit from them one needs 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.

REDUCE[3, 4], like many modern computer algebra systems (MACSYMA, Mathematica, Maple, Scratchpad to name a few), embodies a large amount of mathematical knowledge which is spread out through thousands of procedures in the source code of the system. Most of 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 CAS may be done reading throughout the manual available for the system, a sort of user's guide or in the practical *hands on* approach. However, soon the begginers get stuck and the most confortable and easiest way to solve the problem is consulting an *expert* on the system. They, in general, do not want to waste their time looking either a book or manual up, searching (sometimes in vain) through an enormous amount of terminology, concepts and syntaxes, for the information they need just to be able to start using the system to solve a simple problem: an integral, for instance. That is, perhaps, the reason why many of the potential users with no previous experience with computers keep avoiding to use them. In fact, human consultation is normally scarce and/or expensive, especially in the field of computer algebra where the community is still not very large.

This kind of consultation could, however, be accomplished by an on-line help system (and some progress in this direction has been attained[5, 6]) or even better,

[*]This research was supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Brazil.

[†]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.

by an Intelligent Tutorial System (ITS) [7]. Therefore, any one of these facilities would be most welcomed.

The idea behind a help system is that existing a stored knowledge base, consisting of a large amount 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 to discuss the architecture, operation and tools concerning the design of a prototype of *Smart Help*, an expert help system with these features, as well as to present an implementation of it. Presently, the *Smart Help* domain knowledge base concerns REDUCE. It may well be used, however, to implement different CAS bases. 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 comment briefly on the computer algebra system REDUCE, identify the knowledge embodied by this system and propose a taxonomy for it. In section 3 we describe the knowledge representation shell MANTRA[8] which we have made use of. Section 4 is concerned with the *Smart Help* system design. Finally, in section 5, we give some conclusions to the paper and suggest further upgrades which would turn *Smart Help* into a truly *Intelligent Tutorial System*[7]. In section 6 we give our acknowledgements. In the Appendices, we give an example of the interaction and the syntax.

2 The CAS Reduce

This section concerns the identification and conceptualization of the knowledge concerning REDUCE to be embodied by *Smart Help*. As we noted in the introduction, we take REDUCE as an example of CAS and its choice for this study is a mere consequence of our previous familiarity with it.

REDUCE[3] 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, computational, etc.). The representation of the mathematical domain is the object of another work of our group[9]. In order to represent in *Smart Help* the computational domain, we classified the corresponding knowledge in the following taxonomy of categories:

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

Terminology: Definitions of the various terms like *identifier*, *operator*, *kernel*, etc., and of their interrelations, 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, or a note that *integration* is related to *differentiation*, or even a reference for the algorithm which it implements.[1]

Procedures: General sequences of commands which should be given to perform a certain task like, for instance, defining a new infix operator: one has to declare the operator infix through the INFIX declaration and then give it 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 resulting 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.

In subsection 4.2 we discuss this matter in more detail and present the mapping currently implemented in the prototype of *Smart Help*.

[1]This *knowledge category* should not be confused with *concepts* which are defined through frames. To avoid this possibility, except for one mention in the next section, all other ones will refer to the knowledge category rather than to the usual meaning.

3 The KRS Mantra[2]

MANTRA[8] is a hybrid knowledge representation shell which integrates three different knowledge representation methods:

Four-valued first-order logic language, used to express *assertional knowledge* concerning facts about a domain. It also allows one to verify if an assertion is *entailed* (i.e. implicitly represented) by this set of facts. The entailment algorithm is decidable[10] but presents a weaker semantics which excludes the chaining of independent facts, thus ruling out *modus ponens*[11] but allowing quantifiers.

Terminological language [12] (a kind of *frame* method), used to define *concepts* (not to be confused with the knowledge category with the same name) which are associated to sets of objects of a domain, which are described by restricting the values of their properties. In MANTRA, this language was extended to allow the definition of n-place relations over the concepts. The inference procedure of this method is the *subsumption* procedure.

Semantic network, which is a representation to define *hierarchies* of classes of objects and the *inheritance of properties* from class to subclass. It also allows one to verify the possible relations between classes, including *exception* and *default* links, as well as redundant links and ambiguous networks. No distinction is made here between *objects* and *classes* since this can be done making use of the *logic* method. The inference procedure available here follows the *skeptical inheritance* approach[13].

One could think that classical logic would be able to generate all knowledge entailed 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 allows the modeling of the aspects of *ignorance* and *inconsistency*, useful for representing incomplete and/or incoherent knowledge. The integration of *frame* and *semantic network* methods in addition to *logic* allows *modus ponens* in a controlled way.

[2]This description of the MANTRA System is based on [8]

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 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 syntax, in much the same way as an expert colleague would do. As we have pointed out in the introductory section, such an on-line help would be invaluable for the begginer, by saving a lot of time searching back an forth through the system User's Manual.

4.1 The Conception of Smart Help

The conception of *Smart Help* follows the tradition of help systems being passive. This means that the user learns how to use REDUCE playing freely with it without being interrupted by the *Smart Help*, in contrast to tutorial systems which direct the learning.

It looks like a normal REDUCE session but has *Smart Help* (and his knowledge base, as well as MANTRA) loaded and accessible by a call to it. 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 (see interaction example in Appendix A).

Note, however, that the problem might have been caused by earlier mistakes (a forgotten variable assignment long before, for example). An explanation facility[14] to trace the session history and find the exact place at which the misconception first had its effect was not included in *Smart Help* because we considered it more appropriate to a tutoring system. That is, in case of an error interruption, progress can be made only if, from the correct definition, usage, prerequisites, etc., of the queried topic, as returned by the *Smart Help*, the user can pinpoint himself the misconception which caused the problem.

4.2 The Representation of Reduce's Knowledge in Smart Help

A question could be raised, namely why do we use a knowledge representation shell instead of simpler devices. The answer is twofold: first, *Smart Help* was not conceived as a final product in itself, rather as a step towards the development of more complex and useful environments, like an ITS, as mentioned in the introduction. There plays the semantic of the concepts a decisive role. Second, *Smart Help* should constitute an appropriate environment to verify the taxonomy proposed for the REDUCE knowledge and, for that, we should have a versatile representation device, preferably incorporating Artificial Intelligence techniques.

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 to its adaptiveness to the specific representation method.

The mapping used to match the knowledge categories to the methods and vice-versa have suffered continuous changes since the beginning of this project, as a consequence of our search of efficiency and of our increasing understanding of the nature of the problem. We reached a mapping which seemed reasonable to start building a prototype implementing it. With this prototype in hand we can now start rethinking this mapping. We have exploited all the MANTRA's implemented knowledge representation methods for safety but are not sure (as yet!) that the best mapping include all of them. The current version of it is presented in the following

- **from knowledge types to representation methods:** In some extent this direction of the map corresponds to the *storing* of knowledge into the knowledge base.

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

$$\begin{aligned} \mathit{integration} := {} & \exists\ \mathit{name}{:}[\mathrm{INT}] \sqcap \\ & \exists\ \mathit{argument}{:}[\mathrm{T}] \sqcap \\ & \forall\ (\mathit{argument}{:}[\mathit{first}]){:} \\ & \qquad [\mathit{expression}] \sqcap \\ & \forall\ (\mathit{argument}{:}[\mathit{second}]){:} \\ & \qquad [\mathit{variable}] \end{aligned}$$

Terminology: Terminological definitions have been represented very conveniently through the *frame* method available in MANTRA as a terminological language. Each term (e.g., *right-operator*) is defined as a *specialization* of a more generic one (e.g., *infix-operator*) by means of *slots*, corresponding to attributes, that are filled with the values which specify the former with respect to the last one (e.g., the *flag* RIGHT set). On the other hand, some terms (e.g., *left-operator*) can be seen as *defaults* (or *exceptions*) to the more generic one and have no distinguishing characteristic from it; in this case, the term is defined by a default link "KIND-OF" connecting them according to the *semantic network* method. As mentioned earlier, the relation between terms corresponding to objects (e.g., WHERE) and terms corresponding to classes to which the former belong are represented through the *logic* method. For example, these three knowledge units can be represented as

$$\begin{aligned} & \mathit{right\text{-}operator} := \mathit{infix\text{-}operator} \sqcap \\ & \qquad\qquad \exists\ \mathit{flag}{:}[\mathrm{RIGHT}] \\ & \mathit{kind\text{-}of} := \mathit{left\text{-}operator} \rightarrow \mathit{infix\text{-}operator} \\ & \mathit{infix\text{-}operator}(\mathrm{WHERE}) \end{aligned}$$

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, REFERENCE, etc.). For instance, the examples presented in section 2 above have been represented as

is-represented-by :=
  integration → INT
is-related-to :=
  integration → *differentiation*
reference :=
  integration → *Reduce Manual, section 7.4*

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*[3]. 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. For instance, the example presented in section 2 was represented 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** This direction of the map corresponds somewhat to the *recovering* of knowledge from the knowledge base.

Logic: We used this method to represent specific attributes of *terminological* and *syntactical* nature of individual entities of REDUCE. In addition, individual entities, as instances of concepts, were considered knowledge unities relevant to the *example* link of the *concept* knowledge category.

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*). Like the case of the *logic* method, terms which are subsumed by others were considered instances of these and, therefore, relevant when recovering for the *example* link of *concept* category.

[3]See subsection 4.3

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 (defaults and exceptions), which was not fitted to frame was also stored making use of this method through the link KIND-OF (see again examples above under *syntax* and *terminology*).

4.3 The Architecture 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[4]). 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 same LISP session of MANTRA. To the user it looks like a normal REDUCE session but *Smart Help* is loaded and accessible through the operator `shelp`.

The domain knowledge base was defined as an *object* by means of the object-oriented extension of COMMON LISP called CORBIT[15]. The five knowledge categories defined in section 2 were implemented as *aspects* of the knowledge base object as below

```
(defobject ob-kbase
 (ap-new :function #'kbase-new)
 (ap-show :function #'kbase-show)
 (ap-terminology :object (an ob-terminology))
 (ap-syntax :object (an ob-syntax))
 (ap-concept :object (an ob-concept))
 (ap-procedure :object (an ob-procedure))
 (ap-heuristic :object (an ob-heuristic))
 (ap-term :value '() ) )
```

[4]The architeture of *Smart Help* is shown in more detail than MANTRA or REDUCE since the former is the main concern here. Better descriptions of both can be found in [8] and [3] respectively.

Figure 1: The Structure of *Smart Help*

The aspects *ap-new* and *ap-show* are used to initializate and to exhibit the contents of the knowledge base, and the aspect *ap-term* contains a list of topics defined and available in the knowledge base.

Each knowledge category aspect being also defined as *objects*, has its own storing and recovering procedure corresponding to the aspects *ap-set* and *ap-get* below.

```
(defobject ob-terminology
 (ap-set :function #'terminology-set)
 (ap-get :function #'terminology-get) )

(defobject ob-syntax
 (ap-set :function #'syntax-set)
 (ap-get :function #'syntax-get) )

(defobject ob-concept
 (ap-definition :value nil)
 (ap-set :function #'concept-set)
 (ap-get :function #'concept-get) )

(defobject ob-procedure
 (ap-definition :value nil)
 (ap-set :function #'procedure-set)
 (ap-get :function #'procedure-get) )

(defobject ob-heuristic
 (ap-definition :value nil)
 (ap-set :function #'heuristic-set)
 (ap-get :function #'heuristic-get) )
```

The aspect *ap-definition* presented in the definition of the *concept*, *procedure* and *heuristic* objects are *hash-tables*. In the *concept* object case, it is used to store definitions which appear as *strings* in the input file, which is not accepted by MANTRA's syntax. They are indexed by a symbol generated by the `gensym` LISP procedure which is then passed to MANTRA. In the *procedure* and *heuristic* cases, these hash-tables are used to store the knowledge units themselves since this knowledge categories do not map to MANTRA, as mentioned in subsection 4.2.

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 mapping of the *procedures* and *heuristic* categories into LISP hash-tables is done 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 and defined as

```
(defstruct (st-answer
            (:conc-name nil)
            (:constructor build-answer)
            (:print-function print-answer) )
  (terminological-part (ap-get
             (ap-terminology *ob-kbase*)))
  (conceptual-part (create-st-concept))
  (syntactical-part (ap-get
             (ap-syntax *ob-kbase*)))
  (procedural-part (ap-get
             (ap-procedure *ob-kbase*)))
  (heuristical-part (ap-get
             (ap-heuristic *ob-kbase*))) )
```

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 in subsection 4.4. The slots *constructor* and *print-function* designate the procedures to generate and print the answer respectively. Note that the answer is initialized already filled with the knowledge concerning the queried topic by means of the calls to the recovering procedures corresponding to each knowledge category. 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. Its definition is the following:

```
(defstruct (st-concept
             (:conc-name nil)
             (:constructor
                 create-st-concept) )
  (is-represented-by-link (ap-get
              (ap-concept *ob-kbase*)
              'IS-REPRESENTED-BY))
  (is-tested-by-link (ap-get
              (ap-concept *ob-kbase*)
              'IS-TESTED-BY))
  (is-implemented-through-link (ap-get
              (ap-concept *ob-kbase*)
              'IS-IMPLEMENTED-THROUGH))
  (example-link (ap-get
              (ap-concept *ob-kbase*)
              'EXAMPLE))
  (reference-link (ap-get
              (ap-concept *ob-kbase*)
              'REFERENCE))
  (is-related-to-link (ap-get
              (ap-concept *ob-kbase*)
              'IS-RELATED-TO)) )
```

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*, by querying all the links of the defined *semantic network* links (except the KIND-OF link) about the topic, in an attempt to define it conceptually. As we have said, by querying the *logic* and *frame* methods, 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*, by querying the *frame*, *logic* and the *semantic network* (KIND-OF link) methods about the topic in an attempt to define it terminologically.

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

   (d) To query the *procedural aspect*, by searching the appropriate *hash-table*[5] about the topic, in an attempt to recover procedures associated to it.

   (e) To query the *heuristical aspect*, by searching the appropriate *hash-table*[6] 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 possible inadequate evaluation of the *student's model*). If a concept is too general, however, it is simply deleted.

[5]See subsection 4.3
[6]*idem*

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 report 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 CAS as well.

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[16, ?], 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 report, we intend afterwards to develop an ITS[7] 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, explanations[14], a dynamical reasonning on the student's model[7], etc.

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.

6 Acknowledgements

We would like to thank Prof. J. Calmet and I.A. Tjandra for enlightening discussions and the Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, Brazil, for the financial support.

References

[1] BOYLE, Ann and CAVINESS, B.F. (eds.), *Future Directions for Research in Symbolic Computation* – Report of a Workshop on Symbolic and Algebraic Computation, April 29–30, 1988, Washington, DC , chapter 3, and references therein, Society for Industrial and Applied Mathematics, Philadelphia, 1990.

[2] BUCHBERGER, Bruno (ed.), *Symbolic Mathematical Systems and their Effects on the Curriculum*, special session in the "5th. International Conference on Mathematical Education", Adelaide, Australia, 24–30/8, 1984. SIGSAM Bull. 18(4), 11/1984; BUCHBERGER, Bruno, *Should Students Learn Integration Rules?*, Technical Report, RISC Johannes Kepler University, Linz, Austria, 13/3/1989.

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

[4] MacCALLUM, M.A.H., and WRIGHT, Francis, *Algebraic Computing with* REDUCE, *in* REBOUÇAS, M.J. (ed.), "Lecture Notes from the I Brazilian School on Computer Algebra", vol. I, Oxford University Press (forthcoming book), 1990.

[5] GENESERETH, Michael R., *An Automated Consultant for* MACSYMA, *in* "IJCAI 5" (proceedings of International Joint Conference on Artificial Intelligence, Cambridge, Mass., 8/77):789, 1977.

[6] 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.

[7] SLEEMAN, D. and BROWN, J.S., *Intelligent Tutoring Systems*, Academic Press, London, 1982.

[8] BITTENCOURT, Guilherme, *The* MANTRA *Reference Manual*, Interner Bericht 2/90, Univ. Karslruhe, Fak.f.Informatik, Karlsruhe, West Germany, 1/1990; BITTENCOURT, Guilherme, *An Architecture for Hybrid Knowledge Representation*, PhD Thesis, Univ. Karslruhe, Fak.f.Informatik, Karlsruhe, West Germany, 31/1/1990.

[9] CALMET, Jacques, TJANDRA, I.A., and BITTENCOURT, G., *An Environment for Mathematical Knowledge Representation*, SIGSAM Bull. 24(3):47–48, 7/90.

[10] PATEL-SCHNEIDER, Peter 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.

[11] FRISCH, A.M., *Knowledge Retrieval as Specialized Inference*, Report No. 214, Department of Computer Science, University of Rochester, May 1987.

[12] 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.

[13] 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.

[14] MARTI, Jed, *The Role of Explanation in Symbolic Computation*, *in* INADA, N. and SOMA, T., (eds.), "The Second RIKEN International Symposium on Symbolic and Algebraic Computation by Computers":14–34, World Scientific, Philadelphia, PA, 1984.

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

[16] CALMET, Jacques, *Intelligent Computer Algebra System: Myth, Fancy or Reality? in* JANSSEN, R., (ed.), "Trends in Computer Algebra" (proc. International Symposium, Bad Neuenahr, 5/87, Lecture Notes in Computer Science 296):3–11, Springer-Verlag, Heidelberg, 1987.

A Example of Interaction

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[7] `shelp` , and *Smart Help* would promptly furnish a list of topics which can be queried (from which just a fragment is shown) and ask for a topic to be explained:

```
"SHelp - Version 2.2:  14 Sep 1990"
Information is available about the
   following topics:
 ABS            ACOS
 ANTISYMMETRIC  ANTISYMMETRIC-OPERATOR
 .
 .
 .
 INT            INTEGRATION
 .
 .
 .
Topic?  (Q to quit Smart Help)
```

Now the user can enter the desired topic ("integration" in this case) and get an explanation of it:

```
Topic?  (Q to quit Smart Help)
   integration
INDEFINITE-INTEGRATION is the default
   for INTEGRATION.
INTEGRATION is represented in Reduce
   by INT.
For DEFINITE INTEGRATION, one may try
to DO INDEFINITE-INTEGRATION,
to SAVE-RESULT-IN VARIABLE,
to LOCALLY-SUBSTITUTE VALUES.
References for INTEGRATION: Reduce
   Manual, section 7.4
Concerning INTEGRATION, see also:
   DIFFERENTIATION.
Topic?  (Q to quit Smart Help)
```

Additionally, having discovered that integration in REDUCE is done by means of an operator called "INT", the user can go further asking for an explanation of it:

[7]The user could also have typed `shelp integration` what would lead Smart Help to explain "integration" directly.

```
Topic?  (Q to quit Smart Help) int
INT is a SCALAR-OPERATOR, a kind of
   OPERATOR.
INTEGRATION is represented in Reduce
   by INT.
Its syntax is:
INT(scalar-expression,variable);
Examples of INT:
INT(LOG(X),X);
References for INT: Reduce Manual,
   section 7.4
MOSES, J., Commun.  ACM,
   14(8):548-560, 8/1971
Topic?  (Q to quit Smart Help) q
```

B The syntax of Smart Help

B.1 Asking

To get help from *Smart Help*, one invokes it through the call `shelp` which can be followed by a topic, in which case *Smart Help* will directly give an explanation of it. In the former case, it will furnish a list of topics which can be queried and starts a loop asking the user for topics untill he gives `q` which stops the loop. That is (in BNF)

```
<asking> ::=   (shelp) <topic> ... q |
               (shelp <topic>)
```

B.2 Learning

To insert knowledge into the domain knowledge base, one invokes *Smart Help* through the call `learn` which can be followed by the knowledge unit. In the former case, it will start a loop asking the user for knowledge units untill he gives `q` which stops the loop.

Note that, in the case of *concepts* category, in which links are stablished between concepts, no redundancy is necessary. That is, if a given link like $a \rightarrow b$ is entered, one does not need to enter also $b \rightarrow a$ in the same link. When relevant, *Smart Help* searches the links both ways.

The syntax of the knowledge unit is[8] (in BNF)

```
<learning> ::= (learn) <knowledge unit> ... q |
               (learn <knowledge unit>)

<knowledge unit> ::= '(syntax (<identifier>
                       <syntactical definition>)) |
                     '(terminology (<identifier>
                       <terminological definition>)) |
                     '(concept (<identifier>
                       <conceptual definition>)) |
                     '(procedure (<identifier>
                       <procedural definition>)) |
                     '(heuristic (<identifier>
                       <heuristical definition>))

<terminological definition> ::= <frame definition> |
                                (default
                                 <identifier>) |
                                (exception
                                 <identifier>) |
                                <identifier>

<frame definition> ::= (Not <frame definition>) |
                       (Or-concept
                        <frame definition> ...
                        <frame definition>) |
                       (And-concept
                        <frame definition> ...
                        <frame definition>) |
                       (Some <relation>
                        <frame definition> ...
                        <frame definition>) |
                       (All <relation>
                        <frame definition> ...
                        <frame definition>) |
                       <identifier> |
                       T |
                       NIL

<relation> ::= (Not <relation>) |
               (Or-relation
                <relation> ...
                <relation>) |
               (And-relation
                <relation> ...
                <relation>) |
               (Restriction <relation>
                <frame definition> ...
                <frame definition>) |
               <identifier> |
               T |
               NIL

<conceptual definition> ::= (<conceptual link>
                             <identifier>) |
                            (<conceptual link>
                             <string>)

<conceptual link> ::= is-represented-by |
                      is-tested-by |
                      is-implemented-through |
                      example |
                      reference |
                      is-related-to
```

[8]Presently, the *frame definition* syntax must follow the MANTRA *frame*'s one.

```
<procedural definition> ::= (<left side>
                             <right side>)

<heuristical definition> ::= (<left side>
                              <right side>)

<left side> ::= (<identifier> . . .
                 <identifier>)

<right side> ::= (<action>). . .

<action> ::= (<verb>
              <identifier>)
```