INPE-5334-PRE/1733

MANTRA: A SHELL FOR HYBRID KNOWLEDGE REPRESENTATION

J. Calmet
I.A. Tjandra
Guilherme Bittencourt

INPE
São José dos Campos
Dezembro de 1991

SECRETARIA DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-5334-PRE/1733

MANTRA: A SHELL FOR HYBRID KNOWLEDGE REPRESENTATION

J. Calmet*
I. A. Tjandra*
Guilherme Bittencourt

Aceito para apresentação na 3rd. International Conference on
Tools for Artificial Inteligence,
San José, CA, Nov. 10-13, 1991

* Inst. of Algorithms and Cognitive Systems

INPE
São José dos Campos
Dezembro de 1991

CDU: 681.3

Key Words: Knowlwdge representation; Hydrid systems; expert systems

MANTRA: A Shell for Hybrid Knowledge Representation*

J. Calmet, I.A. Tjandra
Inst. of Algorithms and Cognitive Systems
Am Fasanengarten 5
W-7500 Karlsruhe 1
Germany

G. Bittencourt
Instituto De Pesquisas Espaciais
Caixa Postal 515 – 12.201
São José dos Campos - SP
Brasil

Abstract

In this paper, we present a shell for hybrid knowledge representation. This system supports four different knowledge representation formalisms: First-order logic, terminological language, semantic networks and production systems.

The system automatically supports hybrid inferences taking into account the knowledge represented in different formalisms. All the algorithms involved in the inferences, supported by the system, are decidable, and have the property of being sound and complete according to a four-valued semantics. Besides these facilities, the shell allows the user to represent procedural knowledge of a domain using the primitives embedded at the heuristic level. The system has been implemented in Common Lisp, and makes use of an object-oriented extension for this language.

1 Introduction

To represent an expert's knowledge through the knowledge base of an expert system one is faced with choosing among a rather broad repertoire of formalisms to achieve this goal. One of the crucial issues in representing an expert's knowledge is to take into account the *semantic correctness* of the representation and its corresponding inference mechanisms. Engineering the expert's knowledge, however, would mean to choose the appropriate representation, if any is available, and in many cases it is convenient to use several formalisms.

We have designed and implemented a shell for hybrid knowledge representation, MANTRA, based upon the following design principles: (i) *several co-operating formalisms* are better than a unique representation formalism, (ii) a *clear semantics* explaining the meaning of the knowledge representation language is fundamental and (iii) all algorithms involved must be *decidable* and reasonably fast. From a knowledge engineering point of view MANTRA could also be regarded as a general-purpose shell for building large knowledge-based systems. This statement is illustrated by its architecture.

MANTRA is a multi-layered system and its architecture is made up of three levels:

(i) **the epistemological level**, that is concerned with formalisms showing how facts about the world can be represented in the memory of a computer,

(ii) **the logical level**[1], that consists of a knowledge base management including primitives for storing and legitimating conclusions to be drawn from the facts stored in the knowledge bases, and

(iii) **the heuristic level**, which is concerned with mechanisms to search spaces of possible solutions, to resolve conflict situations, to match patterns and to give explanations if desired.

We adopt a knowledge representation approach consisting of a representational theory, explaining which knowledge is to be represented by which formalisms, and of a common semantics to define the relationship between expressions of different formalisms in a semantically sound manner. The decidability of all algorithms involved is achieved by adopting a four-valued semantics based on the works of Belnap [1] Patel-Schneider [14] [15], Frisch [11] and Thomason et al. [17]. The language includes four different knowledge representation formalisms: First order logic, frames, semantic nets and production systems

*This paper was accepted for publication in the Proceedings of the 3rd International Conference on Tools for Artificial Inteligence, held in San José, California, USA, November 10-13, 1991.

[1] According to the notion of epistemology as introduced in [13], one could regard the following logical level as being also part of the epistemological level.

which are embedded in this system in which assertional reasoning, terminological reasoning, inheritance with exceptions and heuristic programming are integrated.

The paper is organized as follows. Instead of describing separately the semantics of each formalism embedded in the system in full details, which would be too long and can be found in [2], we present in section 2 the theoretical background of a decidable first-order logic relying on the notions described in [14]. This approach is used in the logic, frame and semantic network modules. In section 3, we describe the architecture and, in particular, the language embedded in the system. The implementation is described in section 4. In section 5, we present some concluding remarks. In this section, we shall present a very concise report on a possible, real application of MANTRA, and it is better described in [7].

2 A decidable first-order logic

The standard first-oder logic is widely used in knowledge representation systems due to its expressive power. Adopting this standard logic implies that one faces an undecidable problem when reasoning about a given formula. Therefore, there are some modifications, e.g., extending the first order logic and changing its inference mechanism, and some restrictions, e.g., the length of derivations and elapsed time, which have to be imposed within such an approach. In our work, we adopt a four-valued approach based on the work of Patel Schneider [14] such that it is possible to devise a decidable algorithm for determining whether a formula follows from a set of formulae. The semantics of this approach is *weaker* than that of first order logic. This approach is a variant of first-order relevance logic [1].

First of all, we briefly introduce propositional tautological entailments — a simple type of propositional relevance logic. The syntax of the logic of propositional tautological entailment is the same as that of standard propositional logic, but without an implication operator. Besides the standard two-valued assignment, formulae can also be assigned *neither* true nor false or *both* true and false. Its semantics is based on the four-valued *setups* of propositional relevance logic [1], i.e., $\mathcal{B} = \{\{T\}, \{F\}, \{T, F\}, \{\}\}$. The propositional tautological entailment is defined as follows: α entails β, written $\alpha \rightarrow \beta$, iff β is true whenever α is and α is false whenever β is. This entailment is a much weaker notion than implication as known in standard propositional logic due to the four-valued *setups* which include the set of two-valued assignments. For example, $a \wedge \neg a \not\rightarrow b$ and $a \not\rightarrow b \vee \neg b$, which mean that the classical unsatisfiability and tautologies are not defined in this semantics. In this entailment *modus ponens* is not a valid rule due to $a \wedge (\neg a \vee b) \not\rightarrow b$.

Definition 1 *A situation s consists of a triplet containing a non empty set D, the domain of the situation, a function ε_s, the environment function of s, and a function ξ_s, the extension function of s, i.e. $s = \langle D, \varepsilon_s, \xi_s \rangle$. ε_s maps each function letter, f_j^n, into a function from D^n to D and ξ_s consists of a pair of functions $\langle \xi_s^+, \xi_s^- \rangle$ associating to each predicate a* positive extension, ξ_s^+, *the tuples in the domain known to possess the property of the predicate, and a* negative extension, ξ_s^-, *the tuples known not to possess this property.*

Definition 2 *A variable map is a mapping from variables into some set. If v is a variable map into D, x is a variable, and d is an element of D, then v_d^x is a variable map into D with $v_d^x(y) = d$, if $y = x$, and $v_d^x(y) = v(y)$, otherwise. Given a situation, s, and a variable map, v, a mapping, v_s^*, from terms into domain of s can be defined as follows: $v_s^*(x) = v(x)$, if x is a variable, $v_s^*(f_j^n(t_1, \cdots, t_n)) = (\varepsilon_s(f_j^n))(v_s^*(t_1), \cdots, v_s^*(t_n))$, otherwise.*

Definition 3 *The support relationships of first-order relevance logic for atomic formulae are defined as follows: $s, v \models_t A_j^n(t_1, \cdots, t_n)$ iff $(v_s^*(t_1), \cdots, v_s^*(t_n)) \in \xi_s^+(A_j^n)$, s supports the truth of $A_j^n(t_1, \cdots, t_n)$ under v, and $s, v \models_f A_j^n(t_1, \cdots, t_n)$ iff $(v_s^*(t_1), \cdots, v_s^*(t_n)) \in \xi_s^-(A_j^n)$, s supports the falsity of $A_j^n(t_1, \cdots, t_n)$ under v.*

The relationships are extended to arbitrary first-order formulae — very similar to standard tarskian semantics — by the following rules:

1. $s, v \models_t \neg\alpha$ iff $s, v \models_f \alpha$
   $s, v \models_f \neg\alpha$ iff $s, v \models_t \alpha$

2. $s, v \models_t \alpha \vee \beta$ iff $s, v \models_t \alpha$ or $s, v \models_t \beta$
   $s, v \models_f \alpha \vee \beta$ iff $s, v \models_f \alpha$ and $s, v \models_f \beta$

3. $s, v \models_t \alpha \wedge \beta$ iff $s, v \models_t \alpha$ and $s, v \models_t \beta$
   $s, v \models_f \alpha \wedge \beta$ iff $s, v \models_f \alpha$ or $s, v \models_f \beta$

4. $s, v \models_t \forall x \alpha$ iff for all $d \in D$ $s, v_d^x \models_t \alpha$
   $s, v \models_f \forall x \alpha$ iff for some $d \in D$ $s, v_d^x \models_f \alpha$

5. $s, v \models_t \exists x \alpha$ iff for some $d \in D$ $s, v_d^x \models_t \alpha$
   $s, v \models_f \exists x \alpha$ iff for all $d \in D$ $s, v_d^x \models_f \alpha$

Definition 4 *If α and β are first-order sentences, α entails β iff for all situations, s, and all variable maps, v, if $s, v \models_t \alpha$ then $s, v \models_t \beta$ and if $s, v \models_f \beta$ then $s, v \models_f \alpha$.*

The drawback of first-order tautological entailment is that it can be used to simulate first-order implication and, thus, it is *undecidable*.

Now, we deal with a variant of relevance logic. Initially, we need to introduce the notion of compatible sets of situations. A compatible set of situations is a set of situations with the same domain and the same environment function. Given S, a compatible set of situations each with domain D, and v, a variable map into D, the two support relations for this logic, $S, v \models_t \alpha$ and $S, v \models_f \alpha$ are defined as follows.

1. $S, v \models_t \forall x\alpha$ iff for all $d \in D$ $S, v_d^x \models_t \alpha$
   $S, v \models_f \forall x\alpha$ iff for some $d \in D$ $S, v_d^x \models_f \alpha$
2. $S, v \models_t \exists x\alpha$ iff for some $d \in D$ $S, v_d^x \models_t \alpha$
   $S, v \models_f \exists x\alpha$ iff for all $d \in D$ $S, v_d^x \models_f \alpha$
3. $S, v \models_t \alpha$ iff for all $s \in S$ $s, v \models_t \alpha$
   $S, v \models_f \alpha$ iff for all $s \in S$ $s, v \models_f \alpha$

The interpretation of the formula $\exists x Px$ would be: There exists a known individual for which the P is true, i.e., for some domain element x Px is true in each situation.

There are three different versions of entailment of $\alpha \rightarrow \beta$: (i) β must be true whenever α is, t-entailment (written $\rightarrow_t$), (ii) α must be false whenever β is, f-entailment (written $\rightarrow_f$) and (iii) Both conditions must be fulfilled, tf-entailment (written $\rightarrow$). The entailments for quantifiers can be expressed as follows:

$\forall x Px \rightarrow Pa$	$Pa \rightarrow \exists x Px$
$\forall x Px \rightarrow_t Pa \wedge Pb$	$Pa \vee Pb \not\rightarrow_t \exists x Px$
$\forall x Px \not\rightarrow_f Pa \wedge Pb$	$Pa \vee Pb \rightarrow_f \exists x Px$
$\forall x Px \not\rightarrow Pa \wedge Pb$	$Pa \vee Pb \not\rightarrow \exists x Px$

Thus, the t-entailment is best-suited for knowledge representation since a universal quantifier (t)-entails the conjunction of any number of instantiations whereas a disjunction of instantiations does not (t)-entails an existential quantifier.

Finally, using the following theorem we are able to devise a decidable algorithm to compute t-entailment as described above.

Theorem 1 *If α and β are sentences in skolemized prenex conjunctive normal form, i.e., $\alpha = \forall \vec{z} \bigwedge \alpha_j$ and $\beta = \exists \vec{x} \bigwedge \beta_i$, where $\vec{z}$ is some ordering of the universally quantified variables in α and $\vec{x}$ is some ordering of the existential quantified variables in β, then $\alpha \rightarrow_t \beta$ iff there exists θ, a substitution for $\vec{x}$, such that for each β_i there exist some α_j and ψ, a substitution for $\vec{z}$, such that $\alpha_j\psi \subseteq \beta_i\theta$ where α_j and $\beta_i\theta$ are treated as sets of literals.*

3 The Architecture

In this section, we introduce the three levels which are built modularly. First of all, we describe the epistemological and the logical level, and then the heuristic level of MANTRA.

3.1 The epistemological and the logical levels

The epistemological level consists of three modules: An assertional module, based on the logic as described in section 2, a frame module, based on the terminological box of Krypton [4], and a semantic network module providing inheritance with exceptions [9]. The primitives of these modules are used as parameters of the **Tell** and **Ask** primitives of the logical level. The **Tell** and **Ask** primitives are used to store facts and to interrogate knowledge bases, respectively.

In the logical level two kinds of interfaces are implemented: The interactive interface and the programming interface. The syntax of the programming interface allows the applications of the **Tell** and **Ask** primitives like ordinary Lisp functions. The syntax of the interactive interface is easier to be understood and, therefore, it is more appropriate for users who are not familiar with Lisp. In the sequel, the syntax of the language is presented according to the interactive interface and in section 3.2. we shall present the lisp-like syntax of the language for the heuristic level.

The syntax of these two primitives, which can be regarded as commands, is the following:

command	::= tell(*knowledge base, Fact*) |
	ask(*knowledge base, Query*)
Fact	::= to-logic(*formula*) |
	to-frame(*frame-def*) |
	to-snet(*snet-def*)
Query	::= from-logic(*formula*) |
	from-frame(*frame-question*) |
	from-snet(*snet-question*)|
	from-logic-frame(*logic-frame-question*) |

from-logic-snet(*logic-snet-question*) |
from-frame-snet(*frame-snet-question*)

where *knowledge base* is the name of a particular knowledge base.

As described above, the expressions of Query can be formed either by using a specific module or by using one of three combinations of the modules currently available: logic+frame, logic+snet and frame+snet. Other interactions between modules, e.g., logic-frame-snet, are currently still being developed. The idea of the interaction between the three modules is that the functionalities of one module can be used in order to increase the inference power of another module. For example, to bypass the invalidity of modus ponens in the assertional module one can use the frame or the semantic networks module to represent the chaining of a predicate in an appropriate way. In this way, the user is given a possibility not only to represent a specific domain by means of several knowledge representation formalisms, but the user can also make use of the hybrid reasoning in order to get a semantically motivated answer from a specific knowledge base.

3.1.1 The assertional module

This module is intended to be used to represent assertional knowledge about a particular domain. The expressions of this language are first-order logic formulae. The reasoning of this logic is based on t-entailment, cf. section 2. The syntax of the primitives embedded in this module is the following:

formula ::= (*formula*) |
!E *identifier formula* |
!V *identifier formula* |
formula | *formula* |
formula & *formula* |
∼*formula* |
identifier(*term*,···,*term*)

term ::= *identifier* |
identifier(*term*,···,*term*)

The symbols !E and !V are used to represent the existential and universal quantifiers, respectively. The symbols &, | and ∼ correspond to the logical conjunction, disjunction and negation, respectively.

To give an idea of using the **Tell** and **Ask** primitives according to the assertional module, we give some simple examples. Consider the following commands:

```
tell(kbase0,to-logic(robin(tweety)))
tell(kbase0,to-logic(size(tweety,small)))
tell(kbase1,to-logic(number(n0)
 & first(n0,Paul) & name(n0,Smith)
 & sex(n0,male) & profession(n0,lawyer)
 & ~married(n0)
 & address(n0,Madison41)))
tell(kbase1,to-logic(number(n1)
 & first(n1,Paul) \& name(n1,McCartney)
 & sex(n1,male) \& profession(n1,Singer)
 & married(n1)\& address(n1,Abbeyroad)))
```

The following questions can be given to MANTRA and the answer are given below.

```
Ask(kbase0,from-logic(!Ex size(x,small)
 & robin(x)))
---> The answer is YES,
     with substitution (((x.tweety)))
Ask(kbase1,from-logic(!En !Ea address(n,a)
 & !Ex name(n,x)))
---> The answer is YES
     with substitution
 (((x.Smith) (a.Madison41) (n.n0))
  ((x.McCartney) (a.Abbeyroad72) (n.n1)))
```

To give an idea of the results of the entailment calculation we sketch the algorithm performing this task: Given a set of asserted facts, F_i, and a question $Q = \bigwedge_i Q_i$. The algorithm searches for the set of all substitutions such that, for each Q_i in the query, there is at least one F_i which implies, according to the classical semantics, this Q_i when one of the substitutions is applied. Once this set is calculated the algorithm tries to find a compatible subset, i.e., where the same variables are substituted by the same terms. If this subset is not empty then we say that F_i entails Q.

3.1.2 The frame module

This module is intended to be used to represent a terminology by means of concepts, the categories of objects, and relations, the properties of objects. The notion of relations is an extension of the notion of roles, usually used in terminological languages. Roles are binary relations and relations are arbitrary n-place relations. The main idea of extending roles is that it provides a better integration of this module with the assertional module: The correspondence of n-place relations to n-ary predicates. The principal operation in this module is the subsumption relation which verifies whether a concept or relation subsumes another concept or relation.

The terminological language embedded into the system has some additional characteristics usually not possessed by other terminological languages or hybrid systems: (i) It possesses a rich set of primitives, including disjunction and negation of both concepts and relations, (ii) It provides special symbols for the universal concept and for the bottom concept as well as for the universal relation and for the bottom relation and (iii) It includes tests for subsumption and for equality between concepts and between relations.

The syntax of the terminological language is the following:

```
frame-def      ::= identifier :c=concept |
                   identifier :r=relation

concept        ::= (concept) |
                   ~concept |
                   concept | concept |
                   concept & concept |
                   * |
                   _ |
                   identifier |
                   !E relation:[concept,···,concept] |
                   !V relation:[concept,···,concept]

relation       ::= (relation) |
                   ~relation |
                   relation || relation |
                   relation && relation |
                   < * > |
                   <_> |
                   identifier |
                   relation:[concept,···,concept]

frame-question ::= concept > concept |
                   relation >> relation |
                   concept < concept |
                   relation << relation |
                   concept = concept |
                   relation == relation
```

The symbols * and < * > represent the universal concept and the universal relation, respectively. Analogously, the symbol _ and <_ > represent the bottom concept and relation. The operator ~ represents negation. The operators & and | represent conjunction and disjunction of concepts, respectively. Analogously, the operators && und || represent conjunction and disjunction of relations. The primitive **!V** *relation*:[*concept*,···,*concept*] restricts the values of the n-valued relation *relation* to the set of concepts [*concept*,···,*concept*]. The meaning of this primitive can be interpreted as follows: "All entities such that, if they have property *relation*, then this property takes its values in the concept list [*concept*,···,*concept*]".

Analogously, the primitive **!E** *relation* : [*concept*, ··· ,*concept*] represents an entity whose relation *relation* necessarily takes values in the concept list [*concept*,···,*concept*]. The meaning of this primitive is: "All entities which necessarily present the property *relation* with values taken from the list of concepts [*concept*,···,*concept*]". The primitive *relation*:[*concept*, ···, *concept*] represents the sub-relation of *relation* with values taken from the list of concepts [*concept*, ···, *concept*]

Two different primitives are provided: *identifier* :c= *concept* is used to associate a concept description with an identifier and *identifier* :r=*relation* is used to associate a relation description with an identifier.

To give an idea of using the **Tell** and **Ask** primitives according to this module, we give a simple example.

The terminology of fauna (part) can be represented in the following way:

```
Tell(kbase0,to-frame(blood :r= body-part :
 [liquid] && body-part : [red]))
Tell(kbase0,to-frame(mammal :c=animal
 & !V blood : [warm]
 & !V reproduction : [viviparous]))
Tell(kbase0,to-frame(bird :c= animal :
 & !V blood : [warm]
 & !V reproduction : [oviparous]))
Tell(kbase0,to-frame(elephant :c= mammal :
 & !V food : [plant]
 & !E organ : [trunk]))
Tell(kbase0,to-frame(robin :c= bird
 & !V size : [small] & !E organ : [wing]))
Tell(kbase0,to-frame(carnivore :c= animal
 & !V food : [animal]))
Tell(kbase0,to-frame(herbivore :c= animal
 & !V food : [plant]))
```

The reasoning in this module can be shown by the following questions:

```
Ask(kbase0,from-frame(herbivore > elephant))
---> The answer is YES.
Ask(kbase0,from-frame(mammal > robin))
---> The answer is YES.
```

Using the assertional module and the terminological module according to the given examples one can make use of the *hybrid reasoning* possessed by the system, for instance, in the following way:

```
Ask(kbase0,from-logic-frame(!Ex size(x,small)
 & animal(x)))
---> The answer is YES, with substitution
 (((X.TWEETY)))
```

The idea of the interaction algorithm is to determine all the frame entities subsumed by the predicates appearing in a logical question and to use this subsumed entities as they were predicates t-entailed by the original predicates.

3.1.3 The semantic network module

This module manipulates the notions of classes and hierarchies. The hierarchies can be explicitly created by defining links among classes. Two types of links are provided: *Default* links and *Exception* links. The hierarchies are used as inheritance paths between classes. The main inference procedure of this module calculates the *Subclasses* relation taking into account the explicit exception. The syntax of the primitives in this module is the following:

snet-def ::=
identifier :**k**= *class* |
identifier :**h**= *hierarchy*

class ::=
identifier |
class+ ⋯ +*class*

hierarchy ::=
identifier |
identifier ---> *identifier* |
identifier -/-> *identifier* |
hierarchy + + ⋯ + + *hierarchy*

snet-question ::=
hierarchy(*identifier* ---> *identifier*) |
hierarchy(*identifier* -/-> *identifier identifier*)

The infix operator + takes two classes and creates a new class which is more specific than the two given classes. The infix operators ---> and -/-> take two classes and construct a hierarchy consisting of a single positive or negative link, respectively. The infix operator ++ takes two hierarchies and constructs a new hierarchy consisting of all positive and negative links occurring in these hierarchies. *identifier* :**k**= *class* and *identifier* :**h**= *hierarchy* are used to associate a class description with an identifier and to associate a hierarchy description with an identifier, respectively. Two kinds of questions are allowed in the module in order to verify whether a class is a sub-class of another class in a given hierarchy or not.

The following example shows how the primitives of this module can be used to define hierarchies.

```
Tell(kbase0,to-snet(circus-elephant :k=
 normal-elephant + flying-elephant))
Tell(kbase0,to-snet(color :h=
 elephant --> gray
 ++ royal-elephant -/-> gray))
Tell(kbase0,to-snet(circus :h=
 african-elephant --> elephant
 ++ royal-elephant --> elephant
 ++ circus-elephant --> royal-elephant))
```

The reasoning using only inheritance can be shown by the following examples:

```
Ask(kbase0,form-snet(color
 ++ circus(circus-elephant -/-> gray)))
---> The answer is YES
Ask(kbase0,from-snet(color
 ++ circus(african-elephant ---> gray)))
---> The answer is YES.
Ask(kbase0,from-snet(color
 ++ circus(circus-elephant ---> gray)))
---> The answer is NO.
```

The *interaction algorithm* for the assertional and semantic network modules is very similar to the previous algorithm, but in the present case a hierarchy is used to represent an explicit entailment between first-order logic predicates according to the subclass relation represented in the hierarchy. The following example shows the hybrid reasoning using these two modules.

```
Ask(kbase0,from-logic-snet(color ++circus,
 !Ex size(x,big) & ~gray(x)))
---> The answer is YES, with substitution
 (((X.Clyde)))
```

The next *hybrid reasoning* is the interaction between frame and semantic network modules. The idea of this algorithm is to explicitly construct the subsumption graph of the frame hierarchy, and to use the union of this graph and of the given hierarchy graph to calculate subsumptions of primitive concepts during the subsumption calculation. The next example presents this hybrid reasoning.

```
Ask(kbase0,from-frame-snet(circus,
 animal > african-elephant))
---> The answer is YES.
```

3.2 The heuristic level

Nowadays, there is a controversy between declarativists, believing that the essence of knowledge does not lie in procedures, and proceduralists, asserting that our knowledge is primarily a "knowing how" [18]. As described above our system can appropriately be used for representing declarative knowledge using the primitives at the logical level. In order to synthesize the advantages of the two approaches, declarative and procedural, we integrate the heuristic level into the system in the hope that the user can also represent the procedural knowledge of a domain under consideration using the primitives at this level. Furthermore, these primitives allow the introduction of ad hoc rules in the inference process. These rules can specify strategies for the utilization of the logical level **Ask** and **Tell** primitives.

At this level, the primitives that allow the definition of production systems for the automatic manipulation of knowledge bases are defined. The syntax of the language at this level is given below. A rule of a rule base is made up of the following parts: (i) rule identifier, (ii) a list of variables, (iii) condition part and (iv) action part. The condition and the action parts mainly rely on the **Tell** and **Ask** primitives as defined at the logical level. We allow the user to encapsulate a set of rules in a context, i.e., the rules are valid or can fire if the context is active. Activating, or deactivating, a context can be performed by an appropriate primitive embedded in the action part. The major goal of introducing such contexts is to exclude "redundant" rules while the rule interpreter is selecting rules to be executed, i.e., to minimize the set of conflict rules.

The interpretation of rules is performed by invoking the primitive **Execute**. Presently, the interpretation is performed merely by means of forward chaining. The conflict resolution strategy can explicitly be given by the user. Three kinds of strategies, which are embodied by the following three filters, are currently available:

(i) **context-filter**: this filter works in such a way that the following strategies are taken into account successively:

uniqueness → context order → recency → generality → rule order.

(ii) **recency-filter**:

uniqueness → recency → generality → rule order.

(iii) **rule-filter**:

uniqueness → rule order → recency.

The meaning of each strategy, e.g., recency or generality, is defined as usual [10].

Moreover, the user can explicitly determine the flow strategy which is either **all-rules** (breadth-first search) or **first-rule** (depth-first search).

The lisp-like syntax of the language at this level including the explanation facilities is the following:

```
heuristic-level ::=
rbase-declaration |
rbase-statement |
rbase-query |
interpret |
explanation

rbase-declaration::=
(Decl-rbase identifier ··· identifier)

rbase-statement ::=
(Tell-rbase identifier context ··· context) |
(Tell-rbase identifier identifier) |
(Remove-rbase identifier ··· identifier) |
(Remove-context identifier identifier ··· identifier) |
(Remove-rule identifier identifier ··· identifier) |
(Rule-order identifier identifier ··· identifier) |
(Context-order identifier identifier ··· identifier)

rbase-query     ::=
(Ask-rbase identifier) |
(Ask-context identifier identifier) |
(Ask-rule identifier identifier)

interpret       ::=
(Execute identifier identifier) |
(Execute identifier identifier goal) |
(Execute identifier identifier flow-strategy goal) |
(Execute identifier identifier goal
 flow-strategy conflict-strategy)

flow-strategy   ::=
all-rules |
first-rule

conflict-strategy ::=
context-filter |
recency-filter |
rule-filter

context         ::=
(identifier rule ··· rule) |
(rule ··· rule)

goal            ::=
(condition-part)

rule            ::=
(identifier variables condition-part action-part) |
identifier
```

variables ::=
(*identifier* ··· *identifier*) | ()

condition-part ::=
condition ··· *condition*

condition ::=
identifier : *kbase-rule-quest* |
kbase-rule-quest

action-part ::=
action ··· *action*

action ::=
identifier : *kbase-definition restriction* |
identifier : *kbase-definition* |
kbase-definition restriction |
kbase-definition |
(to-lisp *lisp-expression*) |
(activate *identifier* ··· *identifier*) |
(deactivate *identifier* ··· *identifier*)

restriction ::=
(to-context *identifier* ··· *identifier*)

explanation ::=
(Explain how *identifier*) |
(Explain when *identifier*) |
(Explain why *identifier*) |
(Explain history)

kbase-rule-quest and *kbase-definition* coincide with the language of **Ask** and **Tell** primitives as described in section 3.1, respectively.

4 The Implementation

The system, described above, has been implemented in Kyoto Common Lisp (KCL), a complete implementation of the standard Common Lisp [16], together with an object-oriented extension called Common ORBIT [8]. The use of an object-oriented programming paradigm increases the modularity of the system and makes it easy to modify. In the earlier version of this system, the interface had been developed using KYACC-KLEX [19], an interface between KCL and the compiler generator YACC and LEX. Due to the portability difficulty of YACC and LEX we have replaced this part by a deterministic syntax analyzer that we have implemented directly in KCL.

To facilitate the interconnection between the different methods a single data abstraction has been adopted. This data abstraction consists of a set of *Directed Graphs*. Directed graphs subsumes several of the most commonly used data structures and is also suitable to be used in an interactive system due to their inherent graphical character. The system Grasp, a graph manipulation package, has been adopted as the programming tool implementing this data abstraction. The base of all inference procedures implemented into the system is the unification function. A special unification package has been implemented in Common Lisp. The adopted algorithm is the almost linear algorithm of Martelli and Montanari [12].

One crucial requirement to use the system for developing a knowledge-based system for real applications is that the run time must be considerably fast. This consideration concerns mainly the efficiency of the system. In order to achieve this goal a compiler able to generate "fast" binary codes must be used. Presently, we are developing a "faster" version of the system using Lucid Common Lisp for SUN-Workstations and, in particular, we replace Common ORBIT by the flavor system which is part of Lucid Common Lisp.

5 conclusion

We have given an overview of the MANTRA system, a shell for hybrid knowledge representation. The system is part of the hybrid systems research trend. Its main contribution is the introduction of user-controllable interactions between three different knowledge representation formalisms.

One of many important enhancements to be achieved concerns the user interface. An intelligent, graphical user interface would aid the knowledge engineer in building knowledge bases. She/he could design the knowledge base, using such a user interface, in a visual, easier, and more perspicuous form. We are implementing a cooperative graphical user interface for MANTRA based on X-Windows. A graph editor which can be used to visualize, for instance, hierarchies or terminologies would aid the user for representing expert's knowledge by means of frames or semantic networks. The other possible interactions among the modules are also being implemented. The theoretical studies of these interactions have shown that the algorithms being implemented are sound and complete according to four-valued semantics. The rule interpreter, in the heuristic level, is also being extended to be capable of performing backward chaining.

The semantic soundness is mandatory for one of the application of MANTRA which is to design an environment for mathematical knowledge representation suitable for Computer Algebra Systems [6] [5] [7]. Considering the fact that mathematical domains of computation are inherently modular and that there are inter-relationships among the domains computer algebra is seen , in this environment, as another sort of knowledge that is called mathematical knowledge. The proposed representation of mathematical domains of computation is based on the notion of abstract computational structures. In this way we make use of all knowledge representation formalisms in order to represent mathematical domains, e.g. the assertional module to represent the laws of an abstract domain, the frame module to represent the terminologies of a domain and the semantic network module for representing the whole hierarchy.

This application to symbolic computation in Mathematics is an on-going project where one represents and manipulate highly non-trivial knowledge. MANTRA is proving itself to be very well suited to such an elaborated application. The shell concept on which MANTRA is based enables also to use it to develop expert systems. Another possible application lies in teaching Knowledge Representations to students since it encompasses several cooperating formalisms.

Due to space limitation some features of MANTRA have been only barely described or even overlooked. Among them are for instance: searching algorithms and the theoretical background. They are more thoroughly described in [2] [3]. For the same reason we have limited the number of examples of applications. This is particularly true for the heuristic level.

Acknowledgements

We owe thanks to Anita Lulay and Karsten Homann for implementing the parser and the heuristic level of MANTRA. Thanks are also due to the referees of the paper for their comments.

References

[1] N.D. Belnap, *A Useful Four-Valued Logic*, in "Modern Uses of Multiple-Valued Logic", ed. G. Epstein and J.M. Dunn, Boston: Reidel, 1977, pp. 8 – 37.

[2] G. Bittencourt, *An Architecture for Hybrid Knowledge Representation*, Ph.D. Dissertation, University of Karlsruhe, Department of Computer Science, 1990.

[3] G. Bittencourt, *The Integration of Terminological and Logical Knowledge Representation Languages*, in Z.W Ras et al (Eds.) Proceedings of Fifth International Symposium on Methodologies for Intelligent Systems, October 25 – 27, 1990, Knoxville USA, North-Holland.

[4] R.J. Brachman, V.P. Gilbert, H.J. Levesque, *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON*, Proceeding of IJCAI 9, 1985.

[5] J. Calmet, I.A. Tjandra, *An AI Environment for Computer Algebra*, in J. Johnson (Ed.), Proceedings of International Conference on Artificial Intelligence in Mathematics, Glasgow, pp. 73 – 82, 1991.

[6] J. Calmet, I.A. Tjandra and G. Bittencourt, *A Framework for Representing Algebraic Knowledge Using a Hybrid Knowledge Representation System*, Proceeding of The Fourth International Symposium on Knowledge Engineering, 1990.

[7] J. Calmet, I.A. Tjandra, *An Expert System for Correctness of Symbolic Computation*, to be published in Proceedings of the World Congress on Expert System, Orlando USA, December 16 – 18, 1991, Pergamon Press.

[8] K. De Smedt, *Object-Oriented Programming in FLAVORS and Common ORBIT.* In "R. Hawley (Editors), Artificial Intelligence Programming Environments", Ellis Horwood Limited, pp. 157–176, 1987.

[9] D.W. Etherington, *Reasoning with Incomplete Information: Investigations of Nonmonotonic Reasoning.*, Ph.D. Dissertation, University of British Columbia, ,Vencouver, BC, 1986. Computer Science,

[10] A. Barr, E.A. Feigenbaum, *The Handbook of Artificial Intelligence*, William Kaufmann, 1982.

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

[12] A. Martelli, U. Montanari, *An Efficient Unification Algorithm.* ACM Transactions on Programming Languages and Systems, Vol. 4, No. 2, pp. 258–282, April 1982.

[13] J. McCarthy, *Epistemological Problems of Artificial Intelligence*, in R.J. Brachman, H.J. Levesque (Eds.) "Reading in Knowledge Representation", Morgan Kaufmann, 1985.

[14] P.F. Patel-Schneider, *A Decidable First-Order Logic for Knowledge Representation*, Proceeding of IJCAI 9, pp.455 - 458, 1985.

[15] P.F. Patel-Schneider, *A four-Valued Semantics for Frame-Based Description Languages*, Proceeding of AAAI-86, pp. 344 - 348, 1986.

[16] G.L. Steele Jr., *Common LISP*. Digital Press, Burlington, 1984.

[17] R.H. Thomason, J.F. Horty, D.S. Touretzky, *A Calculus for Inheritance in Monotonic Semantic Nets*, Technical Report CMU-CS-86-138, Carnegie-Mellon University, Department of Computer Science, 1986.

[18] T. Winograd, *Frame Representation and the Declarative/Procedural Controversy*, in R.J. Brachman, H.J. Levesque (Eds.) "Reading in Knowledge Representation", Morgan Kaufmann, 1985.

[19] J.R. Vigouroux, *KYACC-KLEX, The Integration of LEX-YACC into Kyoto Common Lisp*. Personal Communication, 1988.