Workshop on Term Subsumption Languages in Knowledge Representation
Extended Statement of Interest

G. Bittencourt
Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Haid-und-Neu-Strasse 7 - D-7500 Karlsruhe 1 - FRG
electronic-mail: gb@ira.uka.de
phone: (49) 721 6084043
1989

The topics I am mainly interested in discussing during the workshop are: (i) The role of computational intractability and undecidability, (ii) The questions related to the expressive power of terminological languages, and (iii) The implementation issues. The compromise between expressive power and intractability has played an important role in the definition of our hybrid system for knowledge representation. The adopted solution was to use the four-valued semantics, introduced by Belnap (1975, 1977), to guarantee the tractability of the inference algorithms, and to allow the interaction between the different formalisms to enhance the expressive power. The goal of including a inheritance oriented semantic network module, instead of only the assertional and terminological modules, was exactly to provide our language with more expressive power. From the implementation issues, I am mainly interested in the problem of redefinition of concepts and roles in the terminological languages, and how the system should be implemented to provide a complete dynamic redefinition capability.

Presently, my contributions to the field are the following: A first version of a formalism for the semantic definition of knowledge representation
methods (Bittencourt, 1988), the description of an implemented hybrid system to be used as an aid in the specification of knowledge-based systems (Bittencourt, 1989a), the semantic definition of a hybrid architecture using the four-valued approach (Bittencourt, 1989b), and the proposition of an N-valued terminological language and its semantic definition using the same four-valued approach (Bittencourt, 1989c). These contributions are part of my Ph.D. Thesis work, which consists in the specification and implementation of a hybrid system called MANTRA\textsuperscript{1}.

In the following sections, I introduce informally the characteristics of the knowledge representation language implemented in the MANTRA system. Initially, I present a general description of the language and next I describe each of the modules composing it, and present some examples. Finally, I briefly discuss the implementation of the system, and some directions for future work.

1. General Description

The upper level specification of the language is to provide a flexible Knowledge Base facility to be integrated into knowledge-based systems. We adopted an open architecture incorporating several interconnected formalisms. The division of the language into several formalisms has two advantages: the computability problems associated to each formalism can be solved independently and the integration of new formalisms to the language is facilitated.

The language can be thought of as an Abstract Data Type allowing the creation and manipulation of knowledge bases. The knowledge bases consist of a set of knowledge base partitions, each associated to an independent formalism. Knowledge is stored into the knowledge base partitions. Each formalism is used to store knowledge only into the partition associated to the formalism, but queries can be directed to this partition or to a combination of two or more partitions of the knowledge base. The answer to a query directed simultaneously to more than one knowledge base partitions should take into account the knowledge stored in all these partitions. The interaction of one given formalism with the other formalisms in the language is defined by specifying semantically how the knowledge associated to the knowledge base

\textsuperscript{1}For Modular Assertional, Semantic Network and Terminological Representation Approach.
partitions of the other formalisms affects the answers to queries associated to the given formalism.

Two design requirements guided the specification of the formalisms integrated into the language: The semantical motivation of their features and the tight interconnection between them, in the sense that the inference mechanisms provided by each formalism should be aware of the knowledge associated to all other formalisms. The following abstract data type summarizes the specification of the language:

Abstract type KBase

Functions:
New-Base : → KBase
Tell : KBase × Fact → KBase
Ask : KBase × Query → Boolean
Execute : KBase × { < Query, Fact >* } → KBase
end type.

The Query and Fact abstract types provide the Module Access Primitives. These primitives are associated to the epistemological methods, and are used to represent domain knowledge. The Execute function provides the heuristic methods used to apply a set of rules to modify a given knowledge base. Every knowledge representation method integrated into the language is characterized by a set of Definitions and by a set of Questions. Definitions and questions are primitives allowing the utilization of the method to represent knowledge. Given a method M, characterized by the definitions d₀, ..., dₙ and the questions q₀, ..., qₘ, the syntax of the module access primitives corresponding to the facts and queries associated to M is the following:

Fact ::= to-M(d₀) | ... | to-M(dₙ)
Query ::= from-M(q₀) | ... | from-M(qₘ)

These facts and queries allow the utilization of method M as an independent knowledge representation method. The interaction of this method with other methods in the language is provided by taking into account the knowledge associated to other methods when answering questions associated to method M. Given a different method M₀, the interaction between this method and the method M is defined by the following module access primitives:

Query ::= from-M-M₀(q₀) | ... | from-M-M₀(qₘ)

Each knowledge representation method introduced into the language is provided with a semantics defining the intended meaning of its definitions
and questions. This semantic account is used to define the semantics of all module access primitives involving the definitions and questions associated to each method.

Module access primitives are used to construct production rules. These rules are joined together into production systems. These production systems, when executed, can be used to modify their associated knowledge bases according to specific domain knowledge or to shadow the existence of multiple representation methods by directing the user’s queries and facts to the appropriate methods. The architecture of the system reflects the above specification. This architecture consists of three levels: the epistemological level, the logical level and the heuristic level. In the following sections, these levels are described.

2. The Epistemological Level

The first level consists of three modules: an assertional module, based on a decidable first-order logic language, a frame module, based on a terminological language, and a semantic network module, providing inheritance with exceptions. The primitives of the modules of this first level define the epistemological primitives of the language. These primitives are not complete expressions of the language but are used as parameters of the module access primitives, which form the Fact and Query abstract data types used in the Ask and Tell primitives of the logical level. In the following sections, each module of the first level is introduced and some simple examples of their utilization for knowledge representation are presented.

2.1. Assertional Module

This module is based on the work of Patel-Schneider (1985) and Frish (1987). It consists of a first-order logic language allowing the statement of facts and the verification, given a set of stated facts, of the truth value of an assertion. The advantage of adopting this approach is the existence of a decidable algorithm for the verification of the truth value of an assertion. This algorithm is sound and complete with respect to a four-valued semantics. This semantics is weaker than the classical tarskian semantics for first-order
logic, in particular this four-valued semantics does not allow chaining of facts, thus ruling out *modus ponens* as an inference rule. The entailment algorithm in this logic executes a special form of inference called by Frish (1987) *Knowledge Retrieval*.

This module is intended to be used to express assertional knowledge about the domain. The first-order logic syntax is a well adapted language for the expression of knowledge concerning facts about some domain. The following example gives an idea of how facts are expressed in this language.

**Example 1.** Assertional Module.

Robin(Tweety)
Size(Tweety, Small)
Circus-Elephant(Clyde)
Size(Clyde, Big)

### 2.2. Frame Module

The definition of concepts and their interrelations, i.e. a *Terminology*, is an important part of the knowledge representation task of knowledge-based systems. The *Terminological Languages* are part of several knowledge representation systems, e.g. KRYPTON (Brachman et al., 1985), KL-TWO (Vilain, 1985), KANDOR (Patel-Schneider, 1984), BACK (Luck et al., 1987) and TERMLOG (Domini and Lenzerrini, 1988).

Usually, terminological languages consist of a set of primitives allowing the description of *Concepts* and *Roles* given some primitive set of concept and role symbols. These languages are given a semantics where concepts are associated to sets of elements of an arbitrary domain of world objects and roles are associated to binary relations over this domain. The definitions associated to concepts and roles can be automatically classified according to the *Subsumption* relation. A concept or a role subsumes another if all instances of the latter are necessarily instances of the former. A problem with subsumption in terminological languages is that its computational complexity, and even its decidability, is extremely sensitive to the choice of the primitives of the language (Brachman and Levesque, 1984; Schmidt-Schaß, 1988; Patel-Schneider, 1989). A solution for the complexity problem has been proposed by Patel-Schneider (1986): To adopt a four-valued semantics that does not allow all the subsumptions but which provides a semantically motivated
tractable subset of the “classical” subsumptions. We have adopted and extended this solution.

In the proposed terminological language the notion of role has been extended to allow arbitrary \textit{N-Valued Relations} between concepts. This extension allows the direct representation of relations which are naturally n-valued, without partitioning them into binary relations as it is the case in the usual terminological languages. One example of such a relation is the meaning of the English preposition \textit{between}: \textbf{X is between Y and Z}. Using our extended language the concept of all cities between Paris and Lyon can be represented by:

\[(\text{AND city (ALL (RESTR between city city) (Paris Lyon)))}\]

This can be read as: “all the cities such that the values of the relation \textit{between}, restricted to cities, is \textit{(Paris Lyon)}”. Other relations for which finding natural binary decompositions can present difficulties, as it was noticed by Woods (1975), are the \textit{Act Primitives} of Schank and Rieger III (1985), used to represent English sentences. Another advantage of allowing n-valued relations is the possibility of a smooth integration between the terminological language and the assertional module language based on first-order logic, where n-ary logic predicates are associated to n-valued relations. The integration of terminological and assertional modules is typical of \textit{Hybrid Systems}. In these systems, concepts are usually associated to unary predicates and roles are associated to binary predicates, leaving all the higher arity predicates without correspondence in the terminological language. This situation is not very satisfactory, principally if n-valued relations have been partitioned into binary roles to which no meaningful predicate corresponds.

The proposed terminological language has some additional characteristics usually not present in the terminological languages of the hybrid systems: (i) it presents a rich set of primitives, including disjunction and negation of both concepts and relations, (ii) it provides special symbols for the universal concept, for the bottom concept, for the universal relation and for the bottom relation. A concept or a relation is universal when it subsumes all other concepts or relations, and it is a bottom when it is subsumed by all other concepts or relations, (iii) it includes tests for subsumption and for equality between concepts and between relation, and finally, (iv) it introduces a new syntax, the Lisp oriented syntax resulting in clumsy expressions when n-valued relations are allowed. The following example provides some insight on how the primitives of this module can be used to define a terminology in
the domain of biology.

**Example 2.** Frame Module.

Robin := Bird ⊓ ∀ Size : [Small] ⊓ ∃ Organ : [Wing]
Elephant := Mammal ⊓ ∀ Food : [Plant] ⊓ ∃ Organ : [Trunk]
Carnivore := Animal ⊓ ∀ Food : [Animal]
Herbivore := Animal ⊓ ∀ Food : [Plant]

### 2.3. Semantic Network Module

The use of *Semantic Networks* as a tool for knowledge representation can be traced back to the early days of artificial intelligence. The semantic networks have been introduced as a model to the human memory (Quillian, 1968), since they have been used in many different ways: As a different syntax for first-order logic, as a basis for *Object-Oriented Programming*, as a data structure for representing *Search Problems* and as a representation for the *Inheritance* of properties in knowledge representation systems.

In this module the semantic networks are used as a representation tool for the inheritance of properties. In their most general form, the semantic networks provided by this module are *Bipolar, Nonmonotonic, Homogeneous, Multiple Inheritance* semantic networks. A semantic network is said to be bipolar if it allows positive (IS-A) and negative (IS-NOT-A) links. A semantic network is nonmonotonic if it allows exceptions to inherited properties, i.e. if its links are not all strict. A semantic network is homogeneous if all its links are strict (and then it is necessarily monotonic) or if all its links are defeasible. Heterogeneous semantic networks allow both types of links. Finally, a semantic network allows multiple inheritance if each node can have several predecessors in the hierarchy represented by the semantic network.

Early systems allowing multiple inheritance and exceptions were defined through its inference procedures and were theoretically unsound. More recently, several theoretical sound definitions of multiple inheritance have been proposed, e.g. Etherington (1986), Touretzky (1986), Sandewall (1986), Horty et al. (1987) and Doherty (1989). These definitions do agree in all cases where our intuitions are sure but differ in more ambiguous cases. Our approach is to provide a model theoretical semantics for inheritance based
on the four-valued semantics. This approach has already been applied to the monotonic case (Thomason et al., 1986) and is suitable to capture the meaning of specific properties of semantic network inheritance, like the possibility of direct contradictory links.

The module provides a language for defining inheritance hierarchies and for questioning this hierarchies about the relations between their classes. No distinction is made between objects and classes: The nodes allowed in the hierarchies are all classes. This is due to fact that the assertional module provides the object to class relationships through the logical predicates. Two situations where Shortest Path inheritance systems, based on breadth first or depth first search procedures, failed to provide sound answers are: networks with redundant links and ambiguous networks. These two situations can arise in multiple inheritance networks and are presented in figure 1 through the classical “the color of Clyde” and “the Nixon diamond” networks.

![Inheritance Hierarchies](image)

**Fig. 1. Example of Inheritance Hierarchies**

The intuition explaining why Clyde should be non gray, even in the presence of the redundant link from Clyde to Elephant, is that more specific classes should be consider before less specific classes when deciding about the
inheritance of some property. In the Nixon case there are two equally valid paths and two approaches are possible: The Credulous Inheritance approach, to consider both paths in different extensions, and the Skeptical Inheritance, to refuse any conclusion. The skeptical approach has the advantage of avoiding multiple extensions.

The proposed semantic structure can be refined to correspond to different inheritance strategies. Three particular strategies are defined: A monotonic strategy corresponding to the inheritance relation defined in Thomason et al. (1986), a credulous strategy and a skeptical strategy corresponding to the skeptical inheritance strategy defined in Hory et al. (1987). The following example shows how the primitives of this module can be used to defined a hierarchy.

**Example 3.** Semantic Network Module.

Color := Elephant → Gray ⊕ Royal-Elephant ⊕ Gray
Circus := African-Elephant → Elephant ⊕
Royal-Elephant → Elephant ⊕
Circus-Elephant → Royal-Elephant

### 3. The Logical Level

The second level of the architecture introduces the notion of knowledge base. The proposed knowledge representation language can be thought of as an abstract data type whose access functions are the primitives of this level. Two types of primitives are provided, **Tell** and **Ask**, these primitives are used, respectively, to store facts and to interrogate knowledge bases. These primitives receive two parameters: A knowledge base and a module access primitive, i.e. a function of the form to- $M(d)$ and from-M($q$). The module access primitives are the interface between the logical level and the primitives associated to the first level modules.

The Ask primitives are defined in such a way that new facts can be inferred from evidence provided by the knowledge acquired only by one or by a combination of two of the first level modules. The weak first-order logic module assures decidability, and the terminological and semantic network modules are used to increase the inference power of the language.

The four-valued approach has been used to define the interactions between these different epistemological methods. The idea underlying these
interactions is to semantically define a knowledge base for a given method which takes into account the knowledge represented through the other methods. There are nine different interactions: Logic-frames, logic-semantic network, frame-semantic network, frame-logic, semantic network-logic, semantic network-frame, logic-frame-semantic network, frame-logic-semantic network and semantic network-logic-frame. All these interactions have been semantically defined and algorithms for the first three have been proposed and proved sound and complete with respect to the semantic definition. In the following example, the use of the primitives of this level is presented.

**Example 4. The Logical Level.**

* Definition and manipulation of knowledge bases.
  
  K := New-kbase()
  
  Tell(K, to-logic(D)), where D are the expressions in example 1.
  Tell(K, to-frame(D)), where D are the expressions in example 2.
  Tell(K, to-snet(D)), where D are the expressions in example 3.

* Inference using only the assertional knowledge: a weak logical entailment where no chaining of facts is allowed.

  Ask(K, from-logic(∃ x Size(x, Small) ∧ Robin(x))) ⇒ T

  **Because:**
  
  LOGIC: Size(Tweety, Small) and Robin(Tweety)

  Ask(K, from-logic(∃ x Size(x, Small) ∧ Circus-Elephant(x))) ⇒ F

* Inference using terminological knowledge: the subsumption relation. A concept subsumes another if all instances of the latter are also instances of the former.

  Ask(K, from-frame(Herbivore ⊇ Elephant)) ⇒ T

  **Because:**
  
  FRAMES: Elephant := Mammal ⊈ Food : [Plant] ⊈ Organ : [Trunk]
  Herbivore := Animal ⊈ Food : [Plant]

  Ask(K, from-frame(Mammal ⊈ Robin)) ⇒ F

* Inheritance: the existence of a path between two classes of objects inside a hierarchy, taking into account the explicit exceptions.

  Ask(K, from-snet(Color ⊕ Circus : Circus-Elephant ⊈ Gray)) ⇒ T

  **Because:**
  
  SNET: Color ⊕ Circus : Circus-Elephant → Royal-Elephant ⊈ Gray
Ask(K, from-snet (Color ⊕ Circus : African-Elephant → Gray)) ⇒ T

Because:
SNET: Color ⊕ Circus : African-Elephant → Elephant → Gray

Ask(K, from-snet (Color ⊕ Circus : Circus-Elephant → Gray)) ⇒ F

- Interaction between modules.

- Logic and frames: uses the subsumption relation as a implicit implication.
  Ask(K, from-logic-frame(∃ x (Size(x, Small) ∧ Animal(x)))) ⇒ T
  Because:
  LOGIC: Size(Tweety, Small) and Robin(Tweety)
  FRAMES: Animal □ Bird □ Robin

- Logic and semantic networks: uses a given hierarchy as an explicit notation for implication.
  Ask(K, from-logic-snet(Color ⊕ Circus, ∃ x (Size(x, Big) ∧ ¬ Gray(x)))) ⇒ T
  Because:
  LOGIC: Size(Clyde, Big) and Circus-Elephant(Clyde)
  SNET: Color ⊕ Circus : Circus-Elephant → Royal-Elephant ∉ Gray

- Frame and semantic networks: uses a given hierarchy to store the subsumption relation.
  Ask(K, from-frame-snet(Circus, Animal □ African-Elephant)) ⇒ T
  Because:
  FRAME: Animal □ Elephant
  SNET: Circus : African-Elephant → Elephant

4. The Heuristic Level

Finally, the third level consists of primitives allowing the definition of Production Systems (Lenat and McDermott, 1977). The logical and epistemological levels form an abstract data type for the manipulation of knowledge bases. The functions allowing the manipulation of this abstract data type are the Ask and Tell primitives of the logical level. The idea behind the heuristic level is to allow the definition of a new function allowing the manipulation
of the knowledge base abstract data type. This function, denoted **Execute**, is intended to execute a production system where the rules are constructed using the module access primitives, i.e. the functions of the form to-M(d) and from-M(q). This primitive would then be able to automatically manipulate the knowledge bases. The *Conflict Resolution* strategies and the *Flow Control* strategies applied during the execution of these production systems can be explicitly chosen by special primitives. The rules can be used to state facts about the domain knowledge or to control the interaction between the different modules of the first level. Below we present some examples of both types of rules.

**Example 5.** Heuristic programming.

- Rules about the domain:
  Introduction of special instances of modus ponens:
  \[ R1: \text{from-logic}(\text{Man}(x)), \text{to-logic}(\text{Mortal}(x)) \]

  Transitive closure calculation of the subclass relation in hierarchies:
  \[ R2: \text{from-snet}(\text{Greater}(x, y)) \text{ AND } \text{from-snet}(\text{Greater}(y, z)), \text{to-snet}(\text{Greater} := \text{Greater} + x \rightarrow z) \]

- Rules to control the interaction between modules:
  Storage of subsumption relation in a hierarchy:
  \[ I1: \text{from-frame}(x \sqsubset y), \text{to-snet}(S := S + y \rightarrow x) \]
  Transfer of knowledge from semantic network to logic:
  \[ I2: \text{from-snet}(I(c \rightarrow P)), \text{to-logic}(P(c)) \]

  Transfer of knowledge from semantic network to logic:
  \[ I3: \text{from-snet}(I(c \not\rightarrow P)), \text{to-logic}(\neg P(c)) \]

## 5. The implementation

The MANTRA system, described above, has been implemented in Kyoto Common Lisp (KCL), a complete implementation of the standard Common Lisp (Steele Jr., 1984), together with an object-oriented extension called Common ORBIT (De Smedt, 1987). The use of an object-oriented programming paradigm increases the modularity of the system and makes it easy to modify. The interface has been developed using KYACC-KLEX (Vigouroux, 1988), an interface between KCL and the compiler constructor environment.
YACC (Johnson, 1975) and LEX (Lesk, 1975). The use of this environment makes the interface very easy to modify and to adapt to changes in the syntax of the representation language.

The system presents two interface languages: One interactive interface based on menus and a programming interface allowing the use of the system primitives inside Lisp programs. 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 (Bittencourt, 1984), 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 due to Martelli and Montanari (1982).

6. Future Research

The work described above can proceed in three different directions: (i) The continuation of the theoretical study of the semantic definition of knowledge representation methods and their interactions, (ii) the improvement of the MANTRA system, and (iii) the development of applications using the MANTRA system as a tool for knowledge representation. In the following sections, we discuss each of these three directions.

6.1. The Theory

A first research goal in the theoretical part is to study the consequences of rising some restrictions imposed on the representation methods. In the assertional module, predicates can be defined but the recursive definitions are forbidden. The possibility of recursive definitions would lead to a kind of logic programming based on a four-valued semantics. The utility of such a tool is an open question that deserves some research efforts. The same restriction precluding recursive definitions is present in the terminological language, but
in this case even the meaning of a concept or relation recursively defined is not clear. Some research is this direction is also necessary. The semantic network module provide a quite simple language for the definition of hierarchies containing classes and their relationships. The possibility of a richer language containing operators allowing the manipulation of classes - class intersection, class difference, etc - should augment the expressiveness of the representation language of this module. Another point deserving future research is the possibility of integration of more epistemological methods into the system. Some possible methods are: Logic programming languages, database management systems and computer algebra systems. The methodology of integration of the different modules into the logical level allows the easy integration of more methods. It is enough to define the definitions and questions of these new methods and how they interact with the other methods through the correspondent module access primitives.

6.2. The System

The first goal from the implementation point of view is to develop the aspects already formally defined in the hybrid architecture which are not yet implemented: The predicate definition facility of the assertional module, the integration of the production systems at the heuristic level, and the implementation of the six interactions between modules which are not yet included in the system. Another enhancement to be brought to the system concerns the user interface. The main goal of the implemented system being the study of the interactions between the representation methods, less importance has been assigned to the development of a friendly user interface. Some immediate improvements to this interface would include extending the representation language with composite operators that, although not augmenting the expressiveness of the language, will facilitate its use, and the development of a graphical interface to the system, allowing the inherent graphical character of the internal representation of knowledge in the form of graphs to be exploited in profit to the user. To improve the time efficiency of the system, a possible enhancement is the reimplementation of the unification algorithm, knowingly the bottle neck in the execution time of the interaction procedures, in a language such as C. The language C would accelerate the program execution without loosing the transportability guarantee by the
choice of Common Lisp as the implementation language.

6.3. The Applications

A promising application of the system capabilities is the definition of metamodules for nonmonotonic reasoning and possible worlds modeling using the capability of the semantic network module to model logic entailment through the subclass relation in hierarchies. The possibility of including exceptions in this hierarchies facilitates the task of representing nonmonotonic reasoning, and the possibility of manipulating several different named hierarchies allows the implementation of a possible worlds reasoner, where each different set of beliefs would be represented by a different hierarchy. Presently, the system is being used as the base of a project for the implementation of a system for the representation of algebraic knowledge (Calmet et al., 1989).

References


17


