A Description Oriented Logic for Building Knowledge Bases

GIUSEPPE ATTARDI and MARIA SIMI

We discuss the advantages of using a logic system for knowledge representation which is based on descriptions, rather than predicates, and which embodies two fundamental ideas for structuring knowledge that are distilled from semantic networks and frame based languages: inheritance and attributions. Taxonomic reasoning on a lattice of descriptions combined with deduction strategies defined at the metalevel provides the knowlegde base with the capability to deal with complex problem solving tasks.

I. INTRODUCTION

The ability to represent knowledge is often considered essential to build systems with reasoning capabilities. We discuss the advantages of using a logic system for knowledge representation which is based on descriptions, rather than predicates, and which includes some fundamental structuring capabilities.

Examining the most common knowledge representation formalisms, we notice the following.

Among systems with sound logical foundations, Horn clause systems like Prolog, compared to systems based on full predicate logic, trade expressiveness of language for a more tractable proof procedure.

Frame based languages concentrate on epistemological adequacy and structuring mechanisms but deductive capabilities are either ill defined or procedurally defined. The same is true for object-oriented languages.

Omega embodies two ideas for structuring knowledge that are distilled from semantic networks and frame based languages: inheritance and attributions. The resulting language is simple and understandable, yet it retains the full power of the predicate calculus or of a simple set theory.

The logic of Omega has been provided with a formal semantics and has been proved to be sound and consistent [Attardi and Simi 81b].

Current work on Omega is pursuing these fundamental lines of investigation: *taxonomic reasoning*, *knowledge base functionalities* and *metalevel transfer*.

Manuscript received April 15, 1985; revised April 14, 1986. This research was supported in part by ESPRIT, project P440 and by M.P.I., project ASSI and is a contribution to workpackage 4 of EEC Cost-13 project n. 21 "Advanced Issues in Knowledge Representation".

G. Attardi is with DELPHI SpA, I-55049 Viareggio, Italy.

M. Simi is with the Dipartimento di Informatica, Università di Pisa, Pisa, Italy.

Several knowledge representation systems have been based on creating and maintaining taxonomic structures of concepts. The use of such taxonomies has been usually either delegated to specific procedures supplied by the user [Brachman and Schmolze 85, Mylopoulos and Wong 1980], or limited to very simple tasks as determining where to find attributes, values or properties of frames (e.g. KEE [Fikes and Kehler 85].

These frame-based representation systems often resort to hybrid solutions where a frame structuring facility is combined with either a procedural language (e.g. LOOPS [Stefik et al. 83]) or a production system (e.g. KEE). Most of the reasoning is performed by means of such additional components.

Omega explores the idea of *taxonomic reasoning*, that is to base all reasoning on traversal/instantiation of the lattice of descriptions. All knowledge in Omega is represented in a single lattice: from factual knowledge, to general rules, to dependencies and constraints. When a description needs to be accessed to answer a problem, all relevant and related facts and assertions can be found directly connected to it. Heavy use is made of *marker propagation* algorithms when traversing the network, to establish when a solution has been found or to determine when the search has run into a cyclic path. For a more detailed account of the techniques of taxonomic reasoning we refer to [Attardi et al. 1986].

We take the view that complex problem solving tasks should be addressed distinguishing a knowledge base component and a methodological component. The methodological component is what drives the problem solving task, by issuing queries to the knowledge base, acting and interacting with the external word. This component can be considered by and large algorithmical in nature. The knowledge base on the other hand is where deduction takes place, applying deduction rules to find answers.

Taxonomic reasoning is what provides the knowledge base with the fundamental reasoning capabilities, on top of which more sophisticated strategies can be programmed to handle complex problem solving tasks. Rather than being able to perform arbitrary complex theorem proving tasks, an Omega knowledge base is expected to be able to provide immediate answers to elementary questions, of the kind that humans solve with no effort. The deduction is performed traversing the lattice, but it is controlled by user specified strategies which determine which part of the lattice to consider, and how to move around it.

To tailor reasoning strategies to specific applications, strategies can be programmed in the metalanguage of Omega.

Metalanguage is also used as a foundation for the viewpoint mechanism. A viewpoint is a collection of statements

0018-9219/86/1000-1335$01.00 © 1986 IEEE

representing the assumptions of a theory. Multiple viewpoints provide the ability to handle different situations arising either from hypothetical reasoning or for evolution over time of situations, as well as reasoning about beliefs. A formal discussion of viewpoints appears in [6].

The ability to manufacture a metadescription out of any description provides a mean to Omega for accessing its internal structure, for instance to interrogate the system about the concepts that are present, or to examine and explore the goal structure that has been generated in the course of a deduction, for the purpose of generating an explanation of the conclusion.

The design of Omega has its origins in the studies performed at MIT Artificial Intelligence Laboratory on the languages AMORD [11], Ether [15], and Omega itself [4, 5, 17].

Omega is also used as Knowledge Representation system in a couple of Esprit projects, and in particular in Project P440 on "Message Passing Architectures and Description Systems".

In the following section we present an informal introduction to the language and the axiomatic theory of Omega. In section 3 we will discuss how an Omega knowledge base can be organized and how deduction can be implemented.

II. The Description Language Omega

Omega is a logic system, which consists of a language, an axiom system and a set of inference rules. In this section we informally introduce the language by means of examples. Occasionally some axiom will be presented to clarify certain properties of descriptions.

A. Descriptions

The simplest kind of description is the individual description, like:

Boston

or

3

Here "Boston" and "3" are names describing individual entities.

An *instance description* is the basic indefinite description: it is meant to represent a collection of individuals. For instance

(a City)

or

(an Integer)

represent the collection of individuals in the class of cities and of integers.[1]

The description operators and, or and not, interpreted as set intersection, union and complement respectively, allow one to build more complex descriptions, like in the following examples:

(true or false)

(an Even-Number) and (not zero)

Special descriptions are Nothing and Something, denoting the empty set and the universe respectively.

B. Statements

The most elementary sentence in Omega is a predication. A predication relates a *subject* to a *predicate* by the relation *is*. For instance the predication

Boston is (a City)

is understood to assert that the individual named Boston belongs to the class of cities.

Predication can be used to relate arbitrary descriptions. For instance the sentence:

(a Human) is (a Mortal)

states the fact that any individual of class human is also an individual of class mortal.

Note that descriptions can consistently be interpreted as sets of individuals (singletons in case of individual descriptions), and is as the subset relation among sets.

One of the fundamental properties of the relation is is *transitivity*, that allows one to conclude for instance that

Socrates is (a Mortal)

from

Socrates is (a Human)

and

(a Human) is (a Mortal)

Descriptions form a boolean lattice, induced by the partial ordering relation is. The bottom of the lattice is the description Nothing, a special constant which plays the role of the null entity. The top of the lattice is the description Something, another special constant which represents the most generic, universal description.

Composite statements can be built by combining statements with the logical connectives $\wedge$, $\vee$, $\neg$ and $\rightarrow$, as in:

Tom is ((a Cat) or (a Dog)) $\wedge$ $\neg$(Tom is (a Dog)) $\rightarrow$
Tom is (a Cat)

The difference between description operators and statement connectives is illustrated by the following examples:

(true $\wedge$ false) is false
(true and false) is Nothing

Since true, false are two individuals, representing the boolean constants, in the latter sentence "(true and false)" is a description denoting the intersection of two disjoint sets, therefore it is equivalent to Nothing.

[1] The fact that the singular form is used should not mislead the reader. An indefinite description like (a City) does not represent an unspecified element of the class of cities, but rather the whole collection of cities.

C. Attributions

An instance description can have *attributions* attached to it; this serves the purpose of specializing a class by describing some of its properties.

For example, in:

(a Car (with colour red))

the attribution (with colour red) restricts the description to represent just those cars which have color red. Here "color" is an attribute name for the concept "car", and "red" is the corresponding attribute.

Attributions provide also a mean to relate descriptions, as for instance in:

my-car is (a Car (with owner (an Italian)))

The attribute "owner" establishes a relationship between my-car and an Italian who is its owner.

Attributions embed a form of existential quantification; the previous example should indeed be read as saying that there exists an individual, who is an Italian, who is the owner of my-car.

According to this semantics, when the value of an attribute is Nothing, the whole description collapses to Nothing (axiom of *strictness*. For example:

(a Car (with owner Nothing)) is Nothing

This property provides a way to perform type checking or to discover inconsistencies in values of attributes. If we wanted to describe the set of cars with no owner, we would say instead:

(a Car) and not (a Car (with owner Something))

Predicate logic expresses relationships of this kind by means of predicates. The relational data base model in computer science uses a similar concept of relations indexed by attribute names. The semantics of both Predicate Logic and relational databases depend on the fact that relations or predicates have a fixed number of arguments. In Omega attributes can be added or omitted from a description, with no fixed limit. By adding one attribution we obtain a more specific description, i.e. we restrict its extension; omitting an attribution we obtain a more general description. So for instance, according to the axiom of *Omission*:

(a Car (with owner (an Italian)) (with make VW)) is
(a Car (with owner (an Italian)))

Omega provides the capability to supply knowledge in an incremental fashion. Therefore Omega handles partial descriptions and incremental refinement of concepts is possible. For example we can say:

(a Car) is (a Car (with number-of-weels 4))

and later assert:

(a Car) is (a Car (with owner (a Person))

By the axiom of *merging* the following can be concluded:

(a Car) is (a Car (with number-of-weels 4)
(with owner (a Person)))

Properties or attributes are to be considered always relative to a concept, and not as properties of an individual. In other words, an individual may have some attributes when it is considered as member of a class and different attributes as member of another class. It follows that attributes are not allowed to migrate from one concept to another as it is the case with most knowledge representation systems [12, 20, 18, 10], thus avoiding conflicts generated by homonymous attributes and multiple inheritance. Therefore the relation is among classes does not mean that attributes of the superclass are also attributes of the subclass. If this is the case it can be explicitly stated as in:

(a Student (with age = x)) is (a Person (with age = x))

where identifiers starting with "=" denote universally quantified variables. Nevertheless all the functionality of the traditional inheritance mechanism is provided. We will illustrate this with an example.

Most knowledge representation system provide two inheritance relations: IS-A among classes and INSTANCE-OF between a class and its instances. So for example one would describe the class of cars, or rather the concept of a car, as follows:

```
CAR:
  Number_of_weels: 4
  engine: faulty or not-faulty
  colour: red or black or blue
RED-Car IS-A CAR
  colour: red
MY-CAR INSTANCE-OF CAR
  colour: red
  engine: faulty.
```

A red car, described by the concept RED-CAR, would inherit from CAR its attributes. The same is true for MY-CAR. Therefore the inheritance mechanism allows one to deduce, for example, that the number of weels of MY-CAR is 4.

In Omega the same example is expressed as follows:

```
(a Car) is (a Car (with number-of-weels 4)
                  (with engine (faulty or not-faulty))
                  (with colour (red or black or blue)))

(a Red-Car) is (a Car (with colour red))

my-car is (a Car (with colour red)
                 (with engine faulty))
```

Reasoning according to the axioms and inference rules of Omega, we could deduce:

```
my-car is (a Car (with number-of-weels 4)
                 (with engine faulty)
                 (with colour red))
```

More precisely the axioms used are *Omission*, *Transitivity* and *Merging*.

Note however that redefinition is not allowed in subclasses but only refinement of attribute values. For some of the problems related to overriding or cancellations of properties in frame based languages see [7].

Several individual descriptions are allowed as attributes for the same attribute name, as in:

(a Car (with driver John) (with driver Jane))

As a consequence it seems convenient to introduce a notation for attributions allowing a unique individual value for the attribute, like for example in the attribute mother of a person. We call this kind of attributions *projective*. We will express this fact by using an attribution of type with-unique. So we will say:

(a Person (with-unique mother Sarah))

In addition a notation will be introduced for constraining values for the attribute, like for example when we want to describe a person who owns just american cars. For this purpose we introduce the with-every type of attributions.

(a Driver (with-every car (an American-car)))

For with-unique and with-every a stronger axiom than *merging* exists, to recompose partial descriptions. For example:

(a Person (with-unique mother Sarah)
(with mother (a Doctor))) is
(a Person (with mother Sarah and (a Doctor)))

(a Product (with-every $factor_1$ (an Integer))
(with $factor_1$ 2)) is
(a Product (with-every $factor_1$ (an Integer))
(with $factor_1$ 2 and (an Integer)))

Despite the proliferation of the notation, it turns out however that there is only one primitive kind of attribution. The other types of attributions have been defined in terms of the primitive one and their properties derived as theorems.

The primitive type of attribution is called of and its formal semantics is given in terms of n-ary relations. A semantic function associates to a concept a n-ary relation; for example the set of tuples of the relation associated to the concept "Product" could be:

< 0 < $factor_1$ 0> < $factor_2$ 0>>
< 0 < $factor_1$ 0> < $factor_2$ 1>>
< 0 < $factor_1$ 0> < $factor_2$ 2>>
...
< 1 < $factor_1$ 1> < $factor_2$ 1>>
< 2 < $factor_1$ 2> < $factor_2$ 1>>
< 2 < $factor_1$ 1> < $factor_2$ 2>>
< 3 < $factor_1$ 3> < $factor_2$ 1>>
< 3 < $factor_1$ 1> < $factor_2$ 3>>
...

where the first number in each tuple is the product of the second and third numbers.

In this case, the denotation of (a Product) would be all integers, in fact any integer number can be the result of a product. The denotation of (a Product (of $factor_1$ 2)) will be instead the set of even numbers, in fact only even numbers can be obtained as the result of a product in which one of the factors is 2.

The with kind of attribution, corresponding to binary relations, is defined as a special case:

(a c (with a δ)) same (a c (of a δ))

where "same" is defined as is in both directions.

In addition, with-unique and with-every are with's with additional constraints.

D. Variables

The use of description variables enables general facts to be asserted. For example, from:

(a Teacher (with subject =x)) is (an Expert (with field =x))

one can deduce that:

(a Teacher (with subject music)) is (an Expert (with field music))

An interesting example is the following:

((=a is (a Natural)) ∧
(0 is =a) ∧ (=n is =a) →
((a Successor (with-unique pred =n)) is =a)) →
((a Natural) is =a)

This statement is a formulation of the full second order induction principle for natural numbers. It can be read as follows: if =a is a set of numbers containing 0, and for each subset of =a, the set of its successors is contained in =a, then =a contains all the natural numbers.

E Description abstraction

Description abstraction is a powerful construct provided in Omega. In the following example it is used to revert a teacher-student relation:

(any =x such-that (Carl is (a Teacher (with student =x))))

denotes the set of individuals for which the predication

(Carl is (a Teacher (with student =x)))

is true, i.e. all the students of Carl.

An extensive discussion on the inference rules for description abstraction can be found in [Attardi and Simi 81b].

F. Viewpoints

Reasoning on a knowledge base may often require considering different situations, either pertaining to separate worlds (real or hypothetical), or arising because of changes over time.

A viewpoint capability is provided in Omega to enable the deal with such circumstances. For instance to perform the diagnosis of a piece of equipment, one might proceed by a method of hypothesize and test. Each hypothesis is asserted in a new viewpoint, and its consequences are examined. If the consequences are in disagreement with the observations on the equipment, then the hypothesis must be discarded, returning to the previous situation and selecting a different hypothesis.

The viewpoint mechanism of Omega relies fully on a logical basis. In fact a viewpoint is defined as a collection of Omega assertions. The facts that hold in a certain viewpoint are those that derive logically from the assertions in the viewpoint by application of deduction rules.

In the empty viewpoint, containing no assertions, only facts that are tautologies hold. For this reason it is called tautological viewpoint. A new viewpoint can be created by extending a previous viewpoint with an additional assertion, or by combining two viewpoints. As a consequence viewpoints form themselves a (tangled) tree structure, whose root is the tautological viewpoint.

Every statement is asserted in Omega within a certain viewpoint, either implicitely, or explicitely, with the form:

R is (a Resistor (with working YES)) in test-3

III. The Axiomatic Theory

An axiomatic theory of Omega was presented in [Attardi and Simi 81b]. in the style of natural deduction [Kalish and Montague 64].

The axiomatic theory consists in a set of axioms defining a calculus of descriptions and, in close correspondence, a set of axioms defining a calculus of statements. For example the transitivity of is has a dual correspondent in the transitivity of the logical implication.

$$\frac{\sigma_1 \rightarrow \sigma_2,\ \sigma_2 \rightarrow \sigma_3}{\sigma_1 \rightarrow \sigma_3} \qquad \frac{\delta_1 \text{ is } \delta_2,\ \delta_2 \text{ is } \delta_3}{\delta_1 \text{ is } \delta_3}$$

A further set of axioms defines the behaviour of attributions, examples of which (omission, strictness, merging, fusing) have been presented.

The consistency of the Omega logic system has been established in [5], together with a result about completeness: a formula is valid is and only if it is a formal theorem.

Since this logic system is a second order system, and in it a finite categorical set of axioms can be formulated for the natural numbers, this result may appear to contradict Gödel incompleteness theorem. This difficulty is overcome by using a wider class of models then the standard models to interpret the formalism [16].

IV. The Omega Knowledge Base

Omega is primarily a calculus of descriptions. Descriptions and the structuring mechanisms embedded in the logic (inheritance and attributions) can be exploited to build a knowledge base organized as a network where each node corresponds to a description and links to the is relationship. In this section we discuss some of the techniques that are used in the implementation of an Omega knowledge base.

A. Classification

The Omega knowledge base is arranged as a lattice of descriptions. The structure of the lattice can be exploited in several ways: as an efficient access path to descriptions, and as a structure to be explored during reasoning and search.

A major goal of the implementation has been to design the structure of the lattice so that operations like classification of new descriptions, access to descriptions, and selection of statements that are applicable during deduction can all be performed exploiting the same structure.

Another characteristic of the implementation is to provide means to attach to the lattice objects which are external to the Omega system, thereby providing ways to interface Omega to external data bases, or to the underlying Lisp system.

The lattice of classification is designed to contain all kinds of Omega descriptions, from atomic individuals, to instance descriptions with individual attribute values to statement containing variables. The inclusion of statements with variables in the lattice enables to overcome the limitations of traditional semantic networks, which can only represent specific facts and not general ones, i.e. universally quantified assertions.

1) *Classification algorithm*

The classification algorithm is invoked when a new description has to be inserted in the lattice.

The formal semantics of Omega [5] defines the is relationship between descriptions.

The problem of determining subsumption, and therefore the is relationship, is undecidable in Omega and complex in any language with a significant expressive power [9].

Therefore the classification algorithm implemented restricts the attention to that part of the is relation which is explicitely represented by links or paths in the lattice and which is induced by a few structural properties of descriptions.

Direct access points in the network are available to reach individual constants and the most generic representative of instance descriptions with the same concept (the one with no attributions). For instance, when trying to locate (a Man (with dog Fido)), one starts from the node (a man) which is directly accessible through the concept "man".

The most interesting task of the classification algorithm is the classification of instance descriptions. To classify an instance description δ, the classification algorithm starts from the description, which is directly accessible from the concept name of δ and searches the lattice downward. For each child s of this description, a test is done to determine whether s is more general then δ (i.e. δ is s). In particular, given two instance descriptions:

(a C [attr1 δ1] [attr2 δ2] [attr3 δ3])[2]

[2] The notation [attr δ] is an abbreviation for (with-unique attr δ)

(a C [attr1 δ4] [attr3 δ5])

we can prove that

(a C [attr1 δ1] [attr2 δ2] [attr3 δ3]) is (a C [attr1 δ4] [attr3 δ5])

if the following holds:

δ1 is δ4
δ3 is δ5

that is, if two instance descriptions are compared during the classification process, the corresponding attribute values are also compared, according to the monotonicity of attributions.

If (δ is s) is not true, then none of the children of s could be more specific than δ either, so it remains just to consider whether the opposite is relation holds between s and δ, so that δ could be classified as a parent of s. On the other hand, if (δ is s), we recursively classify δ under s.

Since in general a description might have several children, a simple-minded classification algorithm would have to search through all the children in some order and since a description could be a child (or a parent) of more than one description, all children will have to be considered.

This is a potential source of combinatorial explosion. Therefore we have devised a schema by which children of an instance description are ordered in such a way that the path to an already existing description (or to the place in which to insert a new description) can be determined without backtracking.

2) *Statements with variables*

We describe the technique that is used to let descriptions containing variables appear similar to other descriptions, so that they can be classified and dealt similarly to ordinary descriptions.

A predication statement is represented by classifying its subject and predicate "at the right place" in the lattice, and connecting the two parts by an is link.

Suppose we have

δ1[=x] is δ2[=x]

the right place to put δ1[=x] in the lattice is where δ1[Something] would appear, so that it can be reached and instantiated moving upwards in the lattice from descriptions below it, i.e. from less general descriptions.

With a similar argument the right place for δ2[=x] is where δ2[Nothing] would go.[3]

The advantage of this solution is that the unification process can be split in two parts: half of the job is done by the classification algorithm which places a description containing variables connected only to those descriptions which can possibly match it, the second part is the instantiation of variables with suitable values, which is done when traversing the lattice in either direction.

[3] Actually the construction of the description to classify is slightly more complex, in the sense that one has to take into account whether the variable appears inside some negation.

As an example, consider description δ1[δ] which would appear in the lattice as a descendant of δ1[=x]. When we ask for a parent of δ1[δ], the description δ1[=x] is encountered, and this causes =x to be instantiated with δ and the description δ2[δ] to be returned.

With this schema unification is not used for selecting among applicable rules but just to instantiate an applicable rule.

The ability to handle statements with variables means for instance that Prolog clauses can be directly mapped to Omega predications with variables. A backward chaining strategy suitable for emulation of Prolog in Omega is expressed quite easily.

Composite statements

In Omega composite statements can be built using the traditional logical operators ($\wedge$, $\vee$, ...) starting from the elementary is predication. A calculus of statements is axiomatized side by side with the calculus of descriptions.

We will define a sort of canonical form for statements (similar to clausal form of first order predicate calculus) and discuss how a statement expressed in this form can be assimilated in the network.

The theoretical results presented here constitute a necessary step towards the construction of the knowledge base.

Definition: NotEmpty[δ] $\leftrightarrow \exists$ i. Individual[i] $\wedge$ i is δ

Lemma: $\neg$ (δ_1 is δ_2) $\leftrightarrow$ NotEmpty[δ_1 and not δ_2]

Given that not corresponds to set complement and and to set intersection this is a rather obvious set theoretic statement. What is relevant here is that logical negation can be expressed in terms of the description operators (not and and) plus the existential quantifier.

Theorem:

Each Omega statement can be reduced to the following canonical form:

$\sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_{k-1} \rightarrow \sigma_k$

where each σ_i is an elementary predication of the form (δ_1 is δ_2) or a NotEmpty predicate.

A statement of the form

$\sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_{k-1} \rightarrow \delta_1$ is δ_2

Will be assimilated in the network of descriptions as follows. The two nodes corresponding to δ_1 and δ_2 will be connected by a special conditional link (is-cond) whose associated conditions will be the statements σ_1, σ_2, ... σ_{k-1}. The meaning is that the link can be followed only if we are able to verify the associated conditions, which in turn are elementary is predications or NotEmpty predicates.

In addition NotEmpty[δ] can be asserted by introducing a fake individual connected by an is link to δ and checked by searching for some individual starting from δ and following is links downwards.

B. Taxonomic Reasoning

Taxonomic reasoning is a deduction process based on traversal of the lattice of descriptions.

Algorithms for taxonomic reasoning rely upon the structure built by the classification.

As a trivial example of taxonomic reasoning, we can consider the classical syllogism:

(a Man) is (a Mortal) Socrates is (a Man)

The three descriptions will be linked in the lattice with one another, and to determine whether Socrates is a mortal, two links must be traversed to determine the existence of a path between the two descriptions.

It has been argued in [Aït-Kaci and Nasr 85] that integration of such kind of connection can be very beneficial also in a traditional into a logic programming system in terms of improved efficiency due to a significant reduction in the number of resolution/unification steps.

In our approach, since not only constant terms, but also assertions with variables are arranged in the structure of the lattice, all deductions can be performed in this way, without need for full unification, just by a process of lattice traversal/marking/instantiation.

C. Metalevel and deduction strategies

Taxonomic reasoning provides one level of deductive capabilities for Omega.

Since problem-solving methods often are to be tailored to the specific application, in Omega they may be programmed by writing strategies.

Strategies are procedural mechanisms that are attached to statements or classes of statements.

To be able to handle knowledge about strategies, Omega is provided with a meta-level capability [Attardi-Simi 1984]. The syntactical entities of the Omega language, like constants, instance descriptions or predications can be referred to in Omega itself. For instance the description:

(d1 and d2)

can be referred as:

(an And (with-unique arg1 d1) (with-unique arg2 d2))

or, with a short-hand notation, as:

↑(d1 and d2)

Strategies are expressed as attributions of statement descriptions, as in the following example:

```
↑(=d is (=d1 and =d2))
    is (a Predication (with backward-strategy
          '(let ((attempt (new-attempt)))
                (goal ↑(=d is =d1) attempt))
       (goal ↑(=d is =d2) attempt)))
```

The above meta-level statement says that one strategy which can be applied when the goal is to prove a statement of the form "d is (d1 and d2)", is to generate a single attempt to prove the two distinct subgoals that (d is d1) and (d is d2).

This is just one strategy that can be applied to solve inheritance problems, and in fact, exploiting the fact that with attributions can have multiple values, we can also add the following strategy:

```
↑(=d1 is =d2)
    is (a Predication (with backward-strategy
          (foreach d' being the son of =d2
                (goal ↑ (=d1 is d') (new-attempt)))))
```

which would generate one attempt for each child d' of d2 to prove the subgoal that (d1 is d').

The two strategies above are all that is necessary to simulate the backward chaining deductive strategy of Prolog. A Prolog program can be translated into Omega in a straightforward way and then executed according to the above strategy.

This particular strategy is the only one which is provided by Prolog. This strategy works well in all situations where for instance an exhaustive search of the space of solution is desired. In many other cases though this may not be the best strategy.

D. Metalevel and Viewpoints

The following is a strategy corresponding to the axiom of implication introduction which is interesting since it involves hypotethical reasoning. Informally it can be formulated as follows:

if you want to prove "$\sigma 1 \rightarrow \sigma 2$", *assume* $\sigma 1$ *and prove* $\sigma 2$.

The corresponding strategy can be expressed as:

```
↑ (=s1 → =s2) is
   (a Statement
      (with backward-strategy
       '(let ((nvp (create-viewpoint))
              (attempt (new-attempt)))
         (vp-goto nvp)
         (assert =s1)
         (goal =s2 attempt)))))
```

The hypothetical assumption =s1 is formulated here in a new viewpoint nvp, which is a son of the viewpoint where the goal was tackled. If =s2 is proved in nvp, then the original goal succeds, and one returns to the original viewpoint, where =s1 is no longer visible.

In the implementation of Omega, each statement is tagged with the viewpoint to which it belongs, and is visible only from descendant viewpoints. To obtain adequate efficiency in dealing with viewpoints, viewpoints are implemented using a bit array representation for sets. Each bit however, rather then represent a single assertion, represents a collection of assertions describing the incremental difference between a parent and son viewpoint.

V. An Example of Diagnosys

To illustrate the use and the reasoning capabilities of the system, we present in this section a small prototype of expert system for the diagnosis of hardware.

We will consider a very simple device, namely a hair-dryer.

Let us describe how a hair-dryer works. First we describe its internal structure, and admissible values for all its components and attributes.

A. Structure

A hair-dryer consists of a power cable, a three-position power switch, a fan and a resistor. Each of these components will be described in turn. We use the notation "[attr δ]" as an abbreviation for "(with-unique attr δ)".

(a Dryer) is (a Dryer [power-cable (a Power-cable)]
(Ass. 1)
[switch (a Switch)]
[fan (a Fan)]
[resistor (a Resistor)])

(a Power-cable) is (a Power-cable [working (YES or NO)] (Ass. 2) [status (plugged or unplugged)])

(a Switch) is (a Switch [working (YES or NO)]
(Ass. 3)
[status (off or 1 or 2)])

(a Fan) is (a Fan [working (YES or NO)]
(Ass. 4)
[status (spinning or (not spinning))])

(a Resistor) is (a Resistor [working (YES or NO)]
(Ass. 5)
[status (hot or cool)])

Note that a statement like:

(a Dryer) is (a Dryer ...

provides a mean to assert properties valid for all dryers.

B. Interrelations

The proper working of each of the parts of the dryer depends on the working of other components.

What we describe here is just the proper behavior of the system, not all its possible misbehaviors. We consider more convenient to describe what is known of the internal working of the system, rather than trying to formulate all possible conditions of fault in the system.

Many expert systems for diagnosis do instead represent explicitely the information about faults, and how sympthoms are related to a fault or to other sympthoms. Trying to construct such intercausal relationships turns out to be the most difficult task in building an expert system for diagnosis.

In our approach, this information is deduced by the system, which reasons about its model of the working of the system, and draws conclusions about possible sources of fault. This approach follows more closely the approach that an engineer would apply in trying to find the cause of a fault.

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]
[switch (a Switch [working YES] [status (1 or 2)])]
[fan (a Fan [working YES])]) is (Ass. 6)
(a Dryer [fan (a Fan [status spinning])])

(a Dryer [fan (a Fan [status spinning])]) is
(Ass. 7)
[switch (a Switch [working YES] [status (1 or 2)])]
[fan (a Fan [working YES])])

(a Dryer [switch (a Switch [working YES] [status off])]) is
(Ass. 8)
(a Dryer [fan (a Fan [status (not spinning)])])

(a Dryer [switch (a Switch [working YES] [status (off or 1)])]) is
(Ass. 9)
(a Dryer [resistor (a Resistor [status cool])])

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])] [switch (a Switch [working YES] [status 2])]
[resistor (a Resistor [working YES])]) is
(Ass. 10)
(a Dryer [resistor (a Resistor [status hot])])

C. Investigation

We wish to discover the fault in a dryer which blows air but does not produce heat.

We will ask whether a dryer like that can exist:

```
(is-there?
  (a Dryer
    [power-cable
      (a Power-cable [status plugged] [working = pw])]
    [switch (a Switch [status 2] [working = sw])]
    [fan (a Fan [status spinning] [working = fw])]
    [resistor (a Resistor [status cool] [working = rw])]))
```

If such a dryer can exist, the query will find values for the variables = pw, = sw, = fw, and = rw, which describe the working condition of all components of such dryer.

D. Strategy

Our strategy for finding what is wrong with our dryer will be the following. First, we collect all we know about this dryer, gathering all information we have about dryers.

This is done by taxonomic reasoning, by exploring the inheritance network and merging all information that is so collected.

This effect is provided by the basic query strategy *is-there?* As a result of this strategy, some variables present in the query can receive a value.

The second step will be to simplify as much as possible the descriptions obtained, applying algebraic properties of descriptions.

E Finding the solution

We will follow here the above strategy. The deductive steps that we obtain are the following:

(a Dryer [power-cable (a Power-cable [working = pw] [status plugged])]

[switch (a Switch [working = sw] [status 2])] [fan (a Fan [working = fw] [status spinning])]

[resistor (a Resistor [working = rw] [status cool])])

is

(by Axiom of Dropping) (a Dryer [fan (a Fan [working = fw] [status spinning])])

is

(by assertion 7)

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])] [switch (a Switch [working YES] [status (1 or 2)])

is

(from assertion 5)

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES] [status spinning])]

[resistor (a Resistor [working (YES or NO)] [status cool])])

F. Simplification

We can now simplify the description we have obtained so far of our dryer. The simplification rules are those deriving from the axioms of Omega. For instance, we know that:

(individual-1 and individual-2) is Nothing

since the intersection of two singletons consisting of different individuals is the empty set. Similarly:

(Nothing or δ) is δ

According to the axiom of strictness of attributions, if the value of an attribution is Nothing, then the whole description is Nothing.

(a Dryer [power-cable Nothing]) is Nothing

Also we can distribute an or over attributes like in:

(a Resistor [working (YES or NO)]) same ((a Resistor [working YES]) or (a Resistor [working NO]))

Starting with the description obtained so far, simplification proceeds by applying the strategy of split by cases. Formally such strategy consists in distributing the description (YES or NO), obtaining:

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES] [status spinning])]

[resistor (a Resistor [working YES] [status cool])]) or (a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES] [status spinning])]

[resistor (a Resistor [working NO] [status cool])])

Analyzing the two parts separately we obtain:

1. (a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES]

[status spinning])]

[resistor (a Resistor [working YES] [status cool])])

Applying similar steps involving assertion 5, this description reduces to:

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES]

[status spinning])]

[resistor (a Resistor [working YES] [status (hot and cool)])])

which further reduces to:

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])]
[fan (a Fan [working YES] [status spinning])]

[resistor (a Resistor [working YES] [status Nothing])])

i.e.

Nothing

2. (a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])] [fan (a Fan [working YES]
[status spinning])]

[resistor (a Resistor [working NO]
[status cool])])

When putting back the two parts, the first one, which is Nothing will vanish, leaving just the following result:

(a Dryer [power-cable (a Power-cable [working YES] [status plugged])]

[switch (a Switch [working YES [status 2])]
[fan (a Fan [working YES] [status spinning])]

[resistor (a Resistor [working NO] [status cool])])

The variables present in the query have been suitably instantiated during this process, therefore the answer is:

= pw: YES = sw: YES = fw: YES = rw: NO

It is interesting to note that if we translated the above problem in Prolog, we would get something like this:

```
Dryer (_c, _s, _f, _r) :- IsCable(_c), IsSwitch(_s), IsFan(_f),
        IsResistor(_r), CableSwitchFan (_c, _s, _f),
            CableSwitchResistor(_c, _f, _r)

IsCable(cable(_w, _s)) :- YesNo(_w), Plug (_s)
IsSwitch(switch(_w, _s)) :- YesNo(_w), Pos (_s)
IsFan(fan(_w, _s)) :- YesNo(_w), Spin (_s)
IsResistor(resistor(_w, _s)) :- YesNo(_w), Temp (_s)

YesNo(YES) :-
 YesNo(NO) :-
Plug(plugged) :-
Plug (not-plugged) :-
Pos(0) :-
Pos (1) :-
Pos (2) :-
Spin(spinning) :-
Spin (not-spinning) :-
Temp(hot) :-
Temp (cool) :-

CableSwitchFan(cable(YES, plugged), switch(YES, 1),
            fan (YES, spinning)) :-
CableSwitchFan(cable(YES, plugged), switch (YES, 2),
            fan (YES, spinning)) :-
CableSwitchFan(_c, switch(YES, 0), fan(_w, not-spinning)) :-
CableSwitchResistor(_c, switch(YES, 0), resistor(_w, cool)) :-
CableSwitchResistor(_c, switch(YES, 1), resistor(_w, cool)) :-
CableSwitchResistor(cable (YES, plugged), switch(YES, 2),
resistor(YES, hot)) :-
```

If we ask the following question:

```
:- Dryer(cable(_cw, plugged), switch(_sw, 2), fan(_fw,
spinning), resistor(_rw, cool))
```

the Prolog interpreter will not be able to reach a conclusion because it is unable to perform a case analysis on the possible values for the attribute "working" of the resistor. This is due to the fact that negative information cannot be represented and used in the Horn clause logic of Prolog.

Further examples the use of Omega in the construction of expert systems are reported in [Attardi et al. 1985a, Attardi et al. 1985b].

An implementation of the Omega system as described in this paper has been done on a Symbolics Lisp Machine, in the framework of ESPRIT project P440.

VI. Conclusions

We have argued in favor of a logic with structuring capabilities and about problem oriented deduction strategies in order to manage the complexity, as opposed to other approaches where the expressiveness of language is reduced or where formal properties of the system are not provided.

Omega is a formal calculus of descriptions, which offers such structuring capabilities. We discussed how, exploiting such features of the logic, knowledge can suitably be arranged in a network so that algorithms can be devised to perform taxonomic reasoning. Deduction strategies can be defined at the metalevel in order to be tailored to the problem.

VII. Aknowledgements

Carl Hewitt has been the leading force in the early stages of the design of Omega. Andrea Corradini, Stefano Diomedi and Maurizio De Cecco of the ESPRIT team at DELPHI have contributed to the implementation of the ideas presented in this paper. Catiuscia Palamidessi e Simone Martini helped to clarify some theoretical issues.

VIII. References

[1] Aït-Kaci, H. and R. Nasr (1985) LOGIN: A Logic Programming Language with Built-in Inheritance, MCC Technical Report number AI-068-85.

[2] Attardi, G., Corradini, A., De Cecco, M., Diomedi, S., Simi, M. (1985a) The Omega Knowledge Base Development Environment. in "Esprit '85: a Status Report on Continuing Work", North-Holland.

[3] Attardi, G., Corradini, A., De Cecco, M., Simi, M., (1985b) Building Expert Systems with Omega. Technical Report ESP/85/2.

[4] Attardi, G. and M. Simi (1981a) Consistency and Completeness of Omega, a Logic for Knowledge Representation. IJCAI. Vancouver.

[5] Attardi, G. and M. Simi (1981b) Semantics of Inheritance and Attributions in the Description System Omega. AI Memo 642, M.I.T.

[6] Attardi, G. and M. Simi (1984) Metalanguage and Reasoning across Viewpoints. *Proc. of Sixth European Conference on Artificial Intelligence*, Pisa.

[7] Brachman, R. J., (1985) "I lied about the trees" or, Defaults and Definitions in Knowledge Representation, *AI Magazine*, Vol.6, N.3.

[8] Brachman, R.J., Gilbert, V.P., Levesque, H.J., (1985) An Essential Hybrid Reasoning System: Knowledge and Symbol Levels Accounts of Krypton. *Proc. of 9th IJCAI*, Los Angeles, 1985.

[9] Brachman, R.J., Levesque, H.J., (1984) The Tractability of Subsumption in Frame Based Description Languages, in *Proc. of AAAI-84*, Austin, TX, August.

[10] Brachman, R.J., Schmolze (1985) An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2), April-June.

[11] deKleer. J., Doyle, J., Rich, C., Steele, G.L., Sussman, G.J., (1978) AMORD: a Deductive Procedure System. MIT AI-Memo 435.
[12] Fahlman, S., (1979) NETL: A System for Representing and Using Real World Knowledge. MIT Press.
[13] Fikes, R. E. and T. P. Kehler, (1985) The Role of Frame-Based Representation in Reasoning. *CACM* Vol. 28, No. 9.
[14] Kalish and Montague (1964) Logic: Techniques of Formal Reasoning. Harcourt, Brace and World.
[15] Kornfeld, W. (1982). Using parallel processing for problem solving. M.I.T. Ph.D. Thesis.
[16] Henkin, L., Completeness in the Theory of types, *The Journal of Symbolics Logic*, Vol. 15, No. 2, 1950.
[17] Hewitt, C., Attardi, G., Simi M. (1980) Knowledge Embedding in the Description System Omega. *Proc. of First AAAI Conference*, Stanford.
[18] Martin, W., (1979) Description and Specialization of Concepts. In Patrick Winston and Richard Brown, Eds. Artificial Intelligence, MIT Press, Cambridge.
[19] Mylopoulos, J., and Wong, H., (1980) Some Features of the Taxis Data Model, *Proc. of the Conference on Very Large Data Bases*, IEEE Computer Society, pp. 399-410.
[20] Steels L., (1979) Reasoning Modeled as a Society of Communicating Experts, AI Lab Technical Report 542, MIT.
[21] Stefik, M., Bobrow, D.G., Mittal, S., Conway, L. (1983) Knowledge Programming in LOOPS: Report on an Experimental Course, *The AI Magazine*, Vol. 4, No. 3, Fall, pp 3-14.
[22] Weyhrauch, R. (1980) "Prolegomena to a Theory of Formal Reasoning", *Artificial Intelligence* 13, 1-2, 133-170.