Knowledge Embedding in
the Description System Omega

Carl Hewitt, Giuseppe Attardi, and Maria Simi
M.I.T.

545 Technology Square
Cambridge, Mass 02139

Abstract
Omega is a description system for knowledge embedding which combines mechanisms of the predicate calculus, type systems, and pattern matching systems. It can express arbitrary predicates (achieving the power of the \(\omega\)-order quantificational calculus), type declarations in programming systems (Algo, Simula, etc.), pattern matching languages (Planner, Merlin, KRL, etc.). Omega gains much of its power by unifying these mechanisms in a single formalism.

In this paper we present an axiomatization of basic constructs in Omega which serves as an important component of the interface between implementors and users.

2 -- Overview

The syntax of Omega is a version of template English. For example we use the indefinite article in instance descriptions such as the one below:

\((a \text{Son})\)

Instance descriptions like the previous one in general describe a whole category of objects, like the category of sons, in this example.

Such description however can be made more specific, by prescribing particular attributes for the instance description. So for example,

\((a \text{Son (with father Paul) (with mother Mary)})\)

describes a son with father Paul and with mother Mary.

Omega differs from systems based on records with attached procedures (SIMULA and its descendants), generalized property lists (FRL, XRL, etc.), frames (Minsky), and units (KRL) in several important respects. One of the most important differences is that instance descriptions in Omega cannot be updated. This is a consequence of the monotonicity of knowledge accumulation in Omega. Change in Omega is modeled through the use of viewpoints [Barber 1980]. Another difference is that in Omega an instance description can have more than one attribution with the same relation. For example

\((a \text{Human (with child Jack) (with child Jill)})\)

describes a human with a child Jack and a child Jill.

Statements can be deduced because of transitivity of the inheritance relation. For example
John is (a Man)

can be deduced from the following statements

(John is (a Son))
((a Son) is (a Man))

In order to aid readability we will freely mix infix and prefix notations. For example the statement

(John is (a Man))

is completely equivalent to

(is John (a Man))

3 -- Inheritance

The inheritance relation in Omega differs somewhat from the usual ISA relation typically found in semantic networks. For example from

(John is (a Human))
((a Human) is (a Mammal))
(Human is (a Species))

we can deduce

(John is (a Mammal))

but cannot conclude that (John is (a Species)). However we can deduce that

(John is (a (a Species)))

which says that John is something which is a species.

We can often avoid the use of explicit universal quantifiers. For instance the following sentence in quantificational calculus

\( \forall x \text{ Man}(x) \Rightarrow \text{Mortal}(x) \)

can be expressed as

\( (\text{a Man} \text{ is } \text{a Mortal}) \)

The scope of such a variable is the whole statement. Thus the above statement is an abbreviation for

\( (\text{for-all } x ((=x \text{ is } (a \text{ Man})) \Rightarrow (=x \text{ is } (a \text{ Mortal})))) \)

Occasionally it is necessary to use a for-all in the interior of a statement. For example in the following statement expresses the Axiom of Extensionality which is one of the most fundamental principles in Omega:

\( ((\text{for-all } x (x = x_1) \Rightarrow (x = x_2))) \Rightarrow (x_1 = x_2) \)

In the above statement, the scope of \( x_1 \) and \( x_2 \) is the whole statement, while the scope of \( x \) is only the statement \( (x = x_1) \Rightarrow (x = x_2) \).

A form of existential quantification is implicit in the use of attributions. For instance

\( (\text{Pat is } (a \text{ Man } (\text{with father } (\text{an Irishman}))))) \)

says that there is an Irishman who is Pat's father.

Omega makes good use of the ability to place nontrivial descriptions on the left hand side of an inheritance relation. For example from the following statements

\( (\text{a Teacher } (\text{with subject } =s)) \text{ is } (\text{an Expert } (\text{with field } =s)) \)
\( (\text{John is } (\text{a Teacher } (\text{with subject Music}))) \)

we get the following by transitivity of inheritance:

\( (\text{John is } (\text{an Expert } (\text{with field Music}))) \)

Note that statements like the following

\( (\text{is } (\text{and } (\text{a WarmBloodedAnimal } (\text{a BearerOfLiveYoung})) \text{ (a Mammal}))) \)

are much more difficult to express in systems such as KRL and FRL which are based on generalized records and property lists.

If it happens that two descriptions inherit from each other, we will say they are the same. For example if

\( ((\text{a Woman} \text{ is } (a \text{ Human } (\text{with sex female})))) \)
\( ((\text{a Human } (\text{with sex female})) \text{ is } (\text{a Woman})) \)

then we can conclude

\( (\text{a Woman}) \text{ same } (\text{a Human } (\text{with sex female}))) \)

We can express the general principle being used here in Omega as follows.
4 -- Lattice Operators and Logical Operators

The domain of descriptions constitutes a complemented lattice, with respect to the inheritance ordering is, meet and join operations and and or, complementation operation not, and Nothing and Something as the bottom and top respectively of the lattice. Some axioms are required to express these relations. For example all descriptions inherit from Something.

\( (\text{is} \text{ is} \text{ Something}) \)

Furthermore Nothing is the complement of Something.

\( (\text{Nothing same (not Something)}) \)

The usual logical operators on statements are \( \land, \lor, \neg, \Rightarrow \) for conjunction, disjunction, negation, and implication respectively. The description operators not, and, or, etc. apply to all descriptions including statements. It is very important not to confuse the logical operators with the lattice operators in Omega. Note for example:

\( (\land \text{ true false) is false) } \)

\( (\land \text{ true false) is Nothing} \)

\( (\land \text{ true true) is true} \)

Unfortunately most "knowledge representation languages" have not carefully distinguished between lattice operators and logical operators leading to a great deal of confusion.

Note that a statement of the form \( (p \text{ is} \text{ true}) \) does not in general imply that \( (p \text{ same true}) \). For example

\( (\text{Nixon is (a UnindictedCoConspirator) is true}) \)

\( (((\text{a Price (with merchandise Tea)} (\text{with place China})) \text{ is S1) is true}) \)

does not imply

\( (\text{same}) \)

\( (\text{Nixon is (a UnindictedCoConspirator)} \)

\( (\text{a Price (with merchandise Tea) (with place China)}) ) \)

5 -- Basic Axioms

We will state some of the axioms for the description system. The axioms for a theory are usually stated in a metalanguage of the theory. However, since our language contains its metalanguage, we can here give the axioms as ordinary statements in the description system itself.

5.1 Extensionality

Inheritance obeys an Axiom of Extensionality which is one of the most fundamental axioms of Omega. Many important properties can be derived from extensionality which can be expressed in Omega as follows:

\( (\leftrightarrow (=\text{description}_1 \text{ is} \text{ description}_2) \)

\( (\text{for all} \text{ is} (=\text{description}_1)) \)

\( (\text{is} \text{ is} \text{ description}_2)) \)

Note that the meaning of the above statement would be drastically changed if we simply omitted the universal quantifier as follows:

\( (\leftrightarrow (=\text{description}_1 \text{ is} \text{ description}_2) \)

\( (\Rightarrow (=\text{description}_1)) \)

\( (\text{is} \text{ is} \text{ description}_2)) \)

The axiom extensionality illustrates the utility of explicitly incorporating quantification in the language in contrast to some programming languages which claim to be based on logic.

From this axiom alone we are able to derive most of the lattice-theoretic properties of descriptions. In particular we can deduce that is is a reflexive and transitive relation. The following

\( (=\text{description} \text{ is} \text{ description}) \)

expresses the reflexivity of inheritance whereas the following

\( (=> (=\text{description}_1 \text{ is} \text{ description}_2)) \)

\( (=\text{description}_2 \text{ is} \text{ description}_3)) \)

\( (=\text{description}_1 \text{ is} \text{ description}_3)) \)

expresses the transitivity of inheritance.

5.2 Commutativity

Commutativity says that the order in which attributions of a concept are written is irrelevant. We use the notation that an expression of the form \( <\ldots> \) is a sequence of 0 or more elements.
5.5 Monotonicity of Attributes

Monotonicity of attributes is a fundamental property of instance descriptions which is closely related to transitivity of inheritance.

\[
\Rightarrow
\]

\[
(a =\text{description}_1 \text{ is } \text{description}_2)
\]

is

\[
(a =\text{concept} \text{ (with } \text{attribute } \text{description}_1))
\]

\[
(a =\text{concept} \text{ (with } \text{attribute } \text{description}_2))
\]

For example if

\[
(Fred \text{ is } \text{an American})
\]

\[
(Bill \text{ is } \text{a Person (with father Fred))}
\]

then

\[
(Bill \text{ is } \text{a Person (with father } \text{an American}}))
\]

Note that the complementation in Omega is not monotonic. For example

\[
((a \text{ Bostonian}) \text{ is } (a \text{ NewEnglander}))
\]

does not imply that

\[
((\text{not } (a \text{ Bostonian}) \text{ is } (\text{not } (a \text{ NewEnglander}))
\]

5.6 Constraints

Constraints can be used to restrict the objects which will satisfy certain attributions. For example

\[
(a \text{ Human (withConstraint child } a \text{ Male}))
\]

describes humans who have only male children. The Axiom for Constraints is

\[
((a =C (\text{withConstraint } =R =d1) (\text{with } =R =d2)) \text{ is}
\]

\[
(a =c (\text{with } =R (\text{and } =d1 =d2)))
\]

If

\[
(Joan \text{ is } \text{a Human}
\]

\[
(with\text{Constraint child } a \text{ Male})
\]

\[
(with \text{child Jean})
\]
then

(John is (a Human (with child (and (a Male) Jean))))

Note that solely from the statement

(Amy is (a Human (with child (a Male)) (with child Jean)))

no important conclusions can be drawn in Omega. It is entirely possible that Jean is a female with a brother.

We have found the constrained attributions in Omega to be useful generalizations of the increasingly popular "constraint languages" which propagate values through a network of property lists.

6 -- Higher Order Capabilities

In this section we present examples which illustrate the power of the higher-order capabilities of Omega.

6.1 Transitive Relations

If

(3 is (an Integer (with larger 4)))
and
(4 is (an Integer (with larger 5))),
we can conclude by monotonicity that

(3 is (an Integer (with larger (an Integer (with larger 5))))).

From the above statement, we would like to be able to conclude that

(3 is (an Integer (with larger 5))).

This goal can be accomplished by the statement

(larger is (a Transitive-relation (with concept Integer)))

which says that larger is a transitive relation for the concept Integer.

The Axiom for Transitive Relations states that if R is a transitive relation for a concept C and x is an instance of C which is R-related to an instance of C which is R-related to m, then x is R-related to m.

(=> (size is (a Transitive-relation (with concept =C)))

(is

(a =C (with size =R (a =C (with =R m))))

(a =C (with =R m)))

The desired conclusion can be reached by using the above description with C bound to Integer, R bound to larger, and m bound to 5.

6.2 Projective Relations

If

(2 is (a Complex (with real-part (> 0))))
and
(2 is (a Complex (with real-part (an Integer))))
then by merging

it follows that

(z is (a Complex (with real-part (> 0)) (with real-part (an Integer)))).

However in order to be able to conclude that

(z is (a Complex (with real-part (and (> 0) (an Integer))))),

some additional information is needed. One very general way to provide this information is by

(real-part is (a Projective-relation (with concept Complex)))

and by the statement

(=R k is (a Projective-relation (with concept =C))

(is

(a =C (with =R =d))

(a =C (withConstraint =R =e))))

The desired conclusion is reached by using the above description with =C bound to Complex, =R bound to real-part, =description1 bound to (> 0), and =description2 bound to (an Integer).

6.3 Inversion

Inverting relations for efficiency of retrieval is a standard technique in data base organization. Inversion makes use of the converse of a relation with respect to a concept which satisfies the following Axiom for Converse:

(=R same

(a =R same

(a Converse

(With relation (a Converse

(With relation =R)

(With concept =C)))

(With concept =C))))

The Axiom of Inversion expresses how to invert inheritance relations for constrained instance descriptions:

(=R same

(=d1 is (a =C (withConstraint =R (an =d2))))

(=d2)

(is

(a =R with (a Converse

(With relation =R)

(With relation =C) =d1) is =d2)))
For example suppose

\[
((a \text{ Converse (with relation son) (with concept Person)})
\text{ same Parent})
\]

we can conclude

\[
(Sally \text{ is (a Person (withConstraint son (an American))))}
\]

if and only if

\[
((a \text{ Son (with parent Sally)) is (an American))
\]

We have inversion to be a useful generalization of the generalized selection mechanisms in Simula, SmallTalk, and KRL as well as the generalized getprop mechanism in FRL.

The interested reader might try to define the transitivity, projectivity, and converse relations in other "knowledge representation languages."

7 -- Conclusions

Omega encompasses the capabilities of both the \(\omega\)-order quantification calculus, type theory, and pattern matching languages in a unified way. We have illustrated how Omega is more powerful than First Order Logic by showing how it can directly express important properties of relations such as transitivity, projectivity, and converse that are not first order definable.

Omega is based on a small number of primitive concepts including inheritance, instantiation, attribution, viewpoint, logical operations (conjunction, disjunction, negation, quantification, etc.) and lattice operations (meet, join, complement, etc.) It makes use of \text{Inheritance} and \text{Attribution} between descriptions to build a \text{network} of descriptions in which knowledge can be embedded.

Omega is sufficiently powerful to be able to express its own rules of inference. In this way Omega represents a self-describing system in which a great deal of knowledge about itself can be embedded. Because of its expressive power, we have to be very careful in the axiom system for Omega in order to avoid Russell's paradox. Omega uses mechanisms which combines ideas from the Lambda Calculus and Intuitionistic Logic to avoid contradictions in the use of self reference.

We have found axiomatization to be a powerful technique in the development, design, and use of Omega. Axiomatization has enabled us to evolve the design of Omega by removing many bugs which have shown up as undesirable consequences of the axioms. The axiomatization has acted as a contract between the implementors and users of the system. The axioms provide a succinct specification of the rules of inference that can be invoked. The development of Omega has focused on the goals of conceptual simplicity and power. The axiomatization of Omega in itself is a measure of our progress in achieving these goals.

8 -- Related Work

The intellectual roots of our description system go back to von Neumann-Bernays-Goedel set theory [Goedel: 1940], the \(\omega\)-order quantificational calculus, and the lambda calculus. Its development has been influenced by the property lists of LISP, the pattern matching constructs in PLANNER-71 and its descendants QA-4, POPLER, CONNIVER, etc., the multiple descriptions and beta structures of MERLIN, the class mechanism of SIMULA, the frame theory of Minsky, the packagers of PLASMA, the stereotypes in [Hewitt: 1975], the tanglel hierarchies of NETL, the attribute grammars of Knuth, the type system of CLU, the descriptive mechanisms of KRL-0, the partitioned semantic networks of [Fikes and Hendrix: 1977], the conceptual representations of [Yonezawa: 1977], the class mechanism of SMALLTALK [Ingalls: 1978], the goblets of Knowledge Representation Semantics [Smith: 1978], the selector notation of BETA, the inheritance mechanism of OWL, the mathematical semantics of actors [Hewitt and Attardi: 1978], the type system in Edinburgh LCF, the XPRT system of Luc Steels, the constraints in [Borning: 1977, 1979 and Steele and Sussman: 1978].

9 -- Further Work

We have also developed an Omega Machine (which is not described in this paper) that formalizes the operational semantics of Omega.

Mike Brady has suggested that it might be possible to develop a denotational semantics for Omega along the lines of Scott's model of the lambda calculus. This development is one possible approach to establishing the consistency of Omega.
10 -- Acknowledgments

We are grateful to Dana Scott for pointing out a few axioms that were incorrectly stated in a preliminary version of this paper. Jerry Barber has been extremely helpful in aiding us in developing and debugging Omega. Brian Smith and Gene Ciccarelli helped us to clear up some important ambiguities. Conversations with Alan Borning, Scott Fahlman, William Martin, Allen Newell, Alan Perlis, Dana Scott, Brian Smith, and the participants in the "Message Passing Systems" seminar were extremely helpful in getting the description system nailed down. Richard Weyhrauch has raised our interests in meta-theories. His system FOL is one of the first to exploit the classical logical notion of metatheory in A.I. systems. Several discussions with Luc Steels have been the source of cross-fertilization between the ideas in our system and his XPARI system. Roger Duffey and Ken Forbus have served as extremely able teaching assistants in helping to develop this material for the Paradigms for Problem Solving Course at MIT. Comments by Peter Deutsch and Peter Szolovits have materially helped to improve the presentation.

Our logical rules of inference are a further development of a natural deduction system by Kalish and Montague. Some of the axioms for inheritance were inspired by Set Theory.

11 -- Bibliography


Church, A. "A Formulation of the Simple Theory of Types", 1941.


Kalish and Montague.


