What's in a Semantic Network?

James F. Allen
Alan M. Frisch

Computer Science Department
The University of Rochester
Rochester, NY 14627

Abstract

Ever since Woods's "What's in a Link" paper, there has been a growing concern for formalization in the study of knowledge representation. Several arguments have been made that frame representation languages and semantic-network languages are syntactic variants of the first-order predicate calculus (FOPC). The typical argument proceeds by showing how any given frame or network representation can be mapped to a logically isomorphic FOPC representation. For the past two years we have been studying the formalization of knowledge retrievers as well as the representation languages that they operate on. This paper presents a representation language in the notation of FOPC whose form facilitates the design of a semantic-network-like retriever.

1. Introduction

We are engaged in a long-term project to construct a system that can partake in extended English dialogues on some reasonably well specified range of topics. A major part of this effort so far has been the specification of a knowledge representation. Because of the wide range of issues that we are trying to capture, which includes the representation of plans, actions, time, and individuals' beliefs and intentions, it is crucial to work within a framework general enough to accommodate each issue. Thus, we began developing our representation within the first-order predicate calculus. So far, this has presented no problems, and we aim to continue within this framework until some problem forces us to do otherwise.

Given this framework, we need to be able to build reasonably efficient systems for use in the project. In particular, the knowledge representation must be able to support the natural language understanding task. This requires that certain forms of inference must be made. Within a general theorem-proving framework, however, those inferences desired would be lost within a wide range of undesired inferences. Thus we have spent considerable effort in constructing a specialized inference component that can support the language understanding task.

Before such a component could be built, we needed to identify what inferences were desired. Not surprisingly, much of the behavior we desire can be found within existing semantic network systems used for natural language understanding. Thus the question "What inferences do we need?" can be answered by answering the question "What's in a semantic network?"

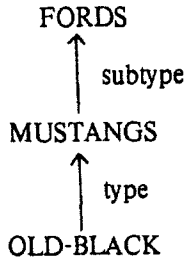
Ever since Woods's [1975] "What's in a Link" paper, there has been a growing concern for formalization in the study of knowledge representation. Several arguments have been made that frame representation languages and semantic-network languages are syntactic variants of the first-order predicate calculus (FOPC). The typical argument (e.g., [Hayes, 1979; Nilsson, 1980; Charniak, 1981a]) proceeds by showing how any given frame or network representation can be mapped to a *logically isomorphic* (i.e., logically equivalent when the mapping between the two notations is accounted for) FOPC representation. We emphasize the term "logically isomorphic" because these arguments have primarily dealt with the content (semantics) of the representations rather than their forms (syntax). Though these arguments are valid and scientifically important, they do not answer our question.

Semantic networks not only represent information but facilitate the retrieval of relevant facts. For instance, all the facts about the object JOHN are stored with a pointer directly to one node representing JOHN (e.g., see the papers in [Findler, 1979]). Another example concerns the inheritance of properties. Given a fact such as "All canaries are yellow," most network systems would automatically conclude that "Tweety is yellow," given that Tweety is a canary. This is typically implemented within the network matcher or retriever.

We have demonstrated elsewhere [Frisch and Allen, 1982] the utility of viewing a knowledge retriever as a specialized inference engine (theorem prover). A specialized inference engine is tailored to treat certain predicate, function, and constant symbols differently than others. This is done by building into the inference engine certain true sentences involving these symbols

and the control needed to handle with these sentences. The inference engine must also be able to recognize when it is able to use its specialized machinery. That is, its specialized knowledge must be coupled to the *form* of the situations that it can deal with.

For illustration, consider an instance of the ubiquitous type hierarchies of semantic networks:



By mapping the types *AUTOS* and *MUSTANGS* to be predicates which are true only of automobiles and mustangs respectively, the following two FOPC sentences are logically isomorphic to the network:

(1.1) $\forall$ x MUSTANGS(x) $\rightarrow$ FORDS(x)

(1.2) MUSTANGS(OLD-BLACK1)

However, these two sentences have not captured the form of the network, and furthermore, not doing so is problematic to the design of a retriever. The subtype and type links have been built into the network language because the network retriever has been built to handle them specially. That is, the retriever does not view a subtype link as an arbitrary implication such as (1.1) and it does not view a type link as an arbitrary atomic sentence such as (1.2).

In our representation language we capture the form as well as the content of the network. By introducing two predicates, *TYPE* and *SUBTYPE*, we capture the meaning of the type and subtype links. *TYPE(i,t)* is true iff the individual i is a member of the type (set of objects) t, and *SUBTYPE*(t_1,t_2) is true iff the type t_1 is a subtype (subset) of the type t_2. Thus, in our language, the following two sentences would be used to represent what was intended by the network:

(2.1) SUBTYPE(FORDS,MUSTANGS)

(2.2) TYPE(OLD-BLACK1,FORDS)

It is now easy to build a retriever that recognizes subtype and type assertions by matching predicate names. Contrast this to the case where the representation language used (1.1) and (1.2) and the retriever would have to recognize these as sentences to be handled in a special manner.

But what must the retriever know about the SUBTYPE and TYPE predicates in order that it can reason (make inferences) with them? There are two assertions, (A.1) and (A.2), such that {(1.1),(1.2)} is logically isomorphic to {(2.1),(2.2),(A.1),(A.2)}. (Note: throughout this paper, axioms that define the retriever's capabilities will be referred to as *built-in axioms* and specially labeled A.1, A.2, etc.)

(A.1) $\forall$ t_1,t_2,t_3 SUBTYPE(t_1,t_2) $\wedge$ SUBTYPE(t_2,t_3) $\rightarrow$ SUBTYPE(t_1,t_3)

(*SUBTYPE* is transitive.)

(A.2) $\forall$ o,t_1,t_2 TYPE(o,t_1) $\wedge$ SUBTYPE(t_1,t_2) $\rightarrow$ TYPE(o,t_2)

(Every member of a given type is a member of its supertypes.)

The retriever will also need to know how to control inferences with these axioms, but this issue is considered only briefly in this paper.

The design of a semantic-network language often continues by introducing new kinds of nodes and links into the language. This process may terminate with a fixed set of node and link types that are the knowledge-structuring primitives out of which all representations are built. Others have referred to these knowledge-structuring primitives as epistemological primitives [Brachman, 1979], structural relations [Shapiro, 1979], and system relations [Shapiro, 1971]. If a fixed set of knowledge-structuring primitives is used in the language, then a retriever can be built that knows how to deal with all of them.

The design of our representation language very much mimics this approach. Our knowledge-structuring primitives include a fixed set of predicate names and terms denoting three kinds of elements in the domain. We give meaning to these primitives by writing domain-independent axioms involving them. Thus far in this paper we have introduced two predicates (*TYPE* and *SUBTYPE*), two kinds of elements (individuals and types), and two axioms ((A.1) and (A.2)). We shall name types in uppercase and individuals in uppercase letters followed by at least one digit.

Considering the above analysis, a retrieval now is viewed as an attempt to prove some queried fact logically follows from the *base facts* (e.g., (2.1), (2.2)) and the built-in axioms (such as A.1 and A.2). For the purposes of this paper, we can consider all base facts to be atomic formulae (i.e., they contain no logical operators except negation). While compound formulae such as disjunctions can be represented, they are of little use to the semantic network retrieval facility, and so will

not be considered in this paper. We have implemented a retriever along these lines and it is currently being used in the Rochester Dialogue System [Allen, 1982].

2. The Basic Representation: Objects, Events, and Relations

An important property of a natural language system is that it often has only partial information about the individuals (objects, events, and relations) that are talked about. Unless one assumes that the original linguistic analysis can resolve all these uncertainties and ambiguities, one needs to be able to represent partial knowledge. Furthermore, the things talked about do not necessarily correspond to the world: objects are described that don't exist, and events are described that do not occur.

In order to be able to capture such issues we will need to include in the domain all conceivable individuals (cf. all conceivable concepts [Brachman, 1979]). We will then need predicates that describe how these concepts correspond to reality. The class of individuals in the world is subcategorized into three major classes: objects, events, and relations. We consider each in turn.

2.1 Objects

Objects include all conceivable physical objects as well as abstract objects such as ideas, numbers, etc. The most important knowledge about any object is its type. Mechanisms for capturing this were outlined above. Properties of objects are inherited from statements involving universal quantification over the members of a type. The fact that a physical object, *o*, actually exists in the world will be asserted as *IS-REAL(o)*.

2.2 Events

The problems inherent in representing events and actions are well described by Davidson [1967]. He proposes introducing events as elements in the domain and introducing predicates that modify an event description by adding a role (e.g., agent, object) or by modifying the manner in which the event occurred. The same approach has been used in virtually all semantic network- and frame-based systems [Charniak, 1981b], most of which use a case grammar [Fillmore, 1968] to influence the choice of role names. This approach also enables quantification over events and their components such as in the sentence, "For each event, the actor of the event causes that event." Thus, rather than representing the assertion that the ball fell by a sentence such as

(try-1) FALL(BALL1),

the more appropriate form is

(try-2) $\exists$ e TYPE(e,FALL-EVENTS) $\wedge$
OBJECT-ROLE(e,BALL1).

This formalism, however, does not allow us to make assertions about roles in general, or to assert that an object plays some role in an event. For example, there is no way to express "Role fillers are unique" or "There is an event in which John played a role." Because we do not restrict ourselves to binary relations, we can generalize our representation by introducing the predicate *ROLE* and making rolenames into individuals in the domain. *ROLE(o,r,v)* asserts that individual *o* has a role named *r* that is filled with individual *v*. To distinguish rolenames from types and individuals, we shall use italics for rolenames.

Finally, so that we can discuss events that did not occur (as opposed to saying that such an event doesn't exist), we need to add the predicate *OCCUR*. *OCCUR(e)* asserts that event e actually occurred. Thus, finally, the assertion that the ball fell is expressed as

(3) $\exists$ e TYPE(e,FALL-EVENTS) $\wedge$
ROLE(e,*OBJECT*,BALL1) $\wedge$
OCCUR(e).

Roles are associated with an event type by asserting that every individual of that type has the desired role. To assert that every event has an *OBJECT* role, we state

(4) $\forall$ e $\exists$ r TYPE(e,EVENTS)
$\rightarrow$ ROLE(e,*OBJECT*,r).

Given this formulation, we could now represent that "some event occurred involving John" by

(5) $\exists$ e,*rolename* TYPE(e,EVENTS) $\wedge$
ROLE(e,*rolename*,JOHN1) $\wedge$
OCCUR(e)

By querying fact (5) in our retriever, we can find all events involving John.

One of the most important aspects of roles is that they are functional, e.g., each event has exactly one object role, etc. Since this is important in designing an efficient retriever, it is introduced as a built-in axiom:

(A.3) $\forall$ r,o,v1,v2 ROLE(o,r,v1) $\wedge$ ROLE(o,r,v2)
$\rightarrow$ (v1 = v2).

2.3 Relations

The final major type that needs discussing is the class of relations. The same problems that arise in representing events arise in representing relations. For

instance, often the analysis of a simple noun-noun phrase such as "the book cook" initially may be only understood to the extent that some relationship holds between "book" and "cook." If we want to represent this, we need to be able to partially describe relations. This problem is addressed in semantic networks by describing relations along the same lines as events.

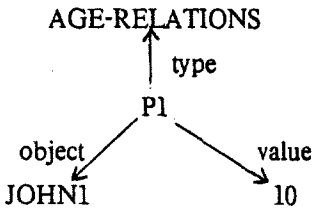
For example, rather than expressing "John is 10" as

(6) AGE-OF(JOHN1,10)

we use the *TYPE* and *ROLE* predicates introduced above to get

(7) ∃ p TYPE(p,AGE-RELATIONS) ∧
ROLE(p,*OBJECT*,JOHN1) ∧
ROLE(p,*VALUE*,10).

This, of course, mirrors a semantic network such as



As with events, describing a relation should not entail that the relation holds. If this were the case, it would be difficult to represent non-atomic sentences such as a disjunction, since in describing one of the disjuncts, we would be asserting that the disjunct holds. We assert that a relation, *r*, is true with *HOLDS(r)*. Thus the assertion that "John is 10" would involve (7) conjoined with *HOLDS(p)*, i.e.,

(8) ∃ p TYPE(p,AGE-RELATIONS) ∧
ROLE(p,*OBJECT*,JOHN1) ∧
ROLE(p,*VALUE*,10) ∧
HOLDS(p)

The assertion "John is not 10" is not the negation of (8), but is (7) conjoined with *~HOLDS(p)*, i.e.,

(9) ∃ p TYPE(p,AGE-RELATIONS) ∧
ROLE(p,*OBJECT*,JOHN1) ∧
ROLE(p,*VALUE*,10) ∧
~HOLDS(p).

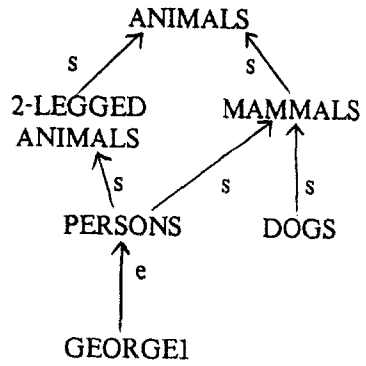
We could also handle negation by introducing the type NOT-RELATIONS, which takes one relation that is filled by another relation. To assert the above, we would construct an individual N1, of type NOT-RELATIONS, with its role filled with p, and assert that N1 holds. We see no advantage to this approach, however, since negation "moves through" the HOLDS predicate. In other words, the relation "not p" holding is equivalent to the relation "p" not holding. Disjunction and conjunction are treated in a similar manner.

3. Making Types Work for You

The system described so far, though simple, is close to providing us with one of the most characteristic inferences made by semantic networks, namely inheritance. For example, we might have the following sort of information in our network:

(10) SUBTYPE(MAMMALS,ANIMALS)
(11) SUBTYPE(2-LEGGED-ANIMALS,ANIMALS)
(12) SUBTYPE(PERSONS,MAMMALS)
(13) SUBTYPE(PERSONS,2-LEGGED-ANIMALS)
(14) SUBTYPE(DOGS,MAMMALS)
(15) TYPE(GEORGE1,PERSONS)

In a notation like in [Hendrix, 1979], these facts would be represented as:



In addition, let us assume we know that all instances of 2-LEGGED-ANIMALS have two legs and that all instances of MAMMALS are warm-blooded:

(16) ∀ x TYPE(x,2-LEGGED-ANIMALS)
→ HAS-2-LEGS(x)

(17) ∀ y TYPE(y,MAMMALS)
→ WARM-BLOODED(y)

These would be captured in the Hendrix formalism using his delineation mechanism.

Note that relations such as "WARM-BLOODED" and "HAS-2-LEGS" should themselves be described as relations with roles, but that is not necessary for this example. Given these facts, and axioms (A.1) to (A.3), we can prove that "George has two legs" by using axiom (A.2) on (13) and (15) to conclude

(18) TYPE(GEORGE1,2-LEGGED-ANIMALS)

and then using (18) with (16) to conclude

(19) HAS-2-LEGS(GEORGE1).

In order to build a retriever that can perform these inferences automatically, we must be able to distinguish facts like (16) and (17) from arbitrary facts involving implications, for we cannot allow arbitrary chaining and retain efficiency. This could be done by checking for implications where the antecedent is composed entirely of type restrictions, but this is difficult to specify. The route we take follows the same technique described above when we introduced the TYPE and SUBTYPE predicates. We introduce new notation into the language that explicitly captures these cases. The new form is simply a version of the typed FOPC, where variables may be restricted by the type they range over. Thus, (16) and (17) become

(20) ∀ x:2-LEGGED-ANIMALS HAS-2-LEGS(x)

(21) ∀ y:MAMMALS WARM-BLOODED(y),

The retriever now can be implemented as a typed theorem prover that operates only on atomic base facts (now including (20) and (21)) and axioms (A.1) to (A.3).

We now can deduce that GEORGE1 has two legs and that he is warm-blooded. Note that objects can be of many different types as well as types being subtypes of different types. Thus, we could have done the above without the type PERSONS, by making GEORGE1 of type 2-LEGGED-ANIMALS and MAMMALS.

4. Making Roles Work for You

In the previous section we saw how properties could be inherited. This inheritance applies to role assertions as well. For example, given a type EVENTS that has an *OBJECT* role. i.e.,

(22) SUBTYPE(EVENTS,INDIVIDUALS)

(23) ∀ x:EVENTS
∃ y:PHYS-OBJS ROLE(x,*OBJECT*,y).

Then if ACTIONS are a subtype of events, i.e.,

(24) SUBTYPE(ACTIONS,EVENTS),

it follows from (A.2), (23), and (24) that for every action there is something that fills its *OBJECT* role, i.e.,

(25) ∀ x:ACTIONS
∃ y:PHYS-OBJS ROLE(x,*OBJECT*,y).

Note that the definition of the type ACTIONS could further specify the type of the values of its *OBJECT* role, but it could not contradict fact (25). Thus

(26) ∀ x:ACTIONS
∃ y:PERSONS ROLE(x,*OBJECT*,y),

further restricts the value of the *OBJECT* role for all individuals of type ACTIONS to be of type PERSONS.

Another common technique used in semantic network systems is to introduce more specific types of a given type by specifying one (or more) of the role values. For instance, one might introduce a subtype of ACTION called ACTION-BY-JACK, i.e.,

(27) SUBTYPE(ACTION-BY-JACK,ACTIONS)

(28) ∀ abj:ACTION-BY-JACK
ROLE(abj,ACTOR,JACK).

Then we could encode the general fact that all actions by Jack are violent by something like

(29) ∀ abj:ACTION-BY-JACK
VIOLENT(abj).

This is possible in our logic, but there is a more flexible and convenient way of capturing such information. Fact (29), given (27) and (28), is equivalent to

(30) ∀ a:ACTIONS
(ROLE a ACTOR JACK)
→ VIOLENT(a).

If we can put this into a form that is recognizable to the retriever, then we could assert such facts directly without having to introduce arbitrary new types.

The extension we make this time is from what we called a type logic to a role logic. This allows quantified variables to be restricted by role values as well as type. Thus, in this new notation, (30) would be expressed as

(31) ∀ a:ACTIONS [ACTOR JACK]
VIOLENT(a).

In general, a formula of the form

$$\forall\ a{:}T\ [R_1V_1]...[R_nV_n]\ Pa$$

is equivalent to

$$\forall\ a\ (TYPE(a,T) \wedge ROLE(a,R_1,V_1) \wedge \ldots \wedge ROLE(a,R_n,V_n)) \rightarrow Pa.$$

Correspondingly, an existentially quantified formula such as

$$\exists\ a{:}T\ [R_1V_1]...[R_nV_n]\ Pa$$

is equivalent to

$$\exists\ a\ TYPE(a,T) \wedge ROLE(a,R_1,V_1) \wedge \ldots \wedge ROLE(a,R_n,V_n) \wedge Pa.$$

The retriever recognizes these new forms and fully reasons about the role restrictions. It is important to remember that each of these notation changes is an extension onto the original simple language. Everything that could be stated previously can still be stated. The new notation, besides often being more concise and convenient, is necessary only if the semantic network retrieval facilities are desired.

Note also that we can now define the inverse of (28), and state that all actions with actor JACK are necessarily of type ACTION-BY-JACK. This can be expressed as

(32) $\forall$ a:ACTIONS [ACTOR JACK] TYPE(a,ACTION-BY-JACK).

5. Equality

One of the crucial facilities needed by natural language systems is the ability to reason about whether individuals are equal. This issue is often finessed in semantic networks by assuming that each node represents a different individual, or that every type in the type hierarchy is disjoint. This assumption has been called E-saturation by [Reiter, 1980]. A natural language understanding system using such a representation must decide on the referent of each description as the meaning representation is constructed, since if it creates a new individual as the referent, that individual will then be distinct from all previously known individuals. Since in actual discourse the referent of a description is not always recognized until a few sentences later, this approach lacks generality.

One approach to this problem is to introduce full reasoning about equality into the representation, but this rapidly produces a combinatorially prohibitive search space. Thus other more specialized techniques are desired. We shall consider mechanisms for proving inequality first, and then methods for proving equality.

Hendrix [1979] introduced some mechanisms that enable inequality to be proven. In his system, there are two forms of subtype links, and two forms of instance links. This can be viewed in our system as follows: the SUBTYPE and TYPE predicates discussed above make no commitment regarding equality. However, a new relation, DSUBTYPE(t_1,t_2), asserts that t_1 is a SUBTYPE of t_2, and also that the elements of t_1 are distinct from all other elements of other DSUBTYPES of t_2. This is captured by the axioms

(A.4) $\forall\ t,t_1,t_2,i_1,i_2$ (DSUBTYPE(t_1,t) $\wedge$ DSUBTYPE(t_2,t) $\wedge$ TYPE(i_1,t_1) $\wedge$ TYPE(i_2,t_2) $\wedge$ ~IDENTICAL(t_1,t_2)) $\rightarrow$ ($i_1 \neq i_2$)

(A.5) $\forall\ t_1$,t DSUBTYPE(t_1,t) $\rightarrow$ SUBTYPE(t_1,t)

We cannot express (A.4) in the current logic because the predicate IDENTICAL operates on the syntactic form of its arguments rather than their referents. Two terms are IDENTICAL only if they are lexically the same. To do this formally, we have to be able to refer to the syntactic form of terms. This can be done by introducing quotation into the logic along the lines of [Perlis, 1981], but is not important for the point of this paper.

A similar trick is done with elements of a single type. The predicate DTYPE(i,t) asserts that i is an instance of type t, and also is distinct from any other instances of t where the DTYPE holds. Thus we need

(A.6) $\forall\ i_1,i_2$,t (DTYPE(i_1,t) $\wedge$ DTYPE(i_2,t) $\wedge$ ~IDENTICAL(i_1,i_2)) $\rightarrow$ ($i_1 \neq i_2$)

(A.7) $\forall$i,t DTYPE(i,t) $\rightarrow$ TYPE(i,t)

Another extremely useful categorization of objects is the partitioning of a type into a set of subtypes, i.e., each element of the type is a member of exactly one subtype. This can be defined in a similar manner as above.

Turning to methods for proving equality, [Tarjan, 1975] describes an efficient method for computing relations that form an equivalence class. This is adapted to support full equality reasoning on ground terms. Of course it cannot effectively handle conditional assertions of equality, but it covers many of the typical cases.

Another technique for proving equality exploits knowledge about types. Many types are such that their instances are completely defined by their roles. For such a type T, if two instances I1 and I2 of T agree on all their respective roles then they are equal. If I1 and I2 have a role where their values are not equal, then I1 and I2 are not equal. If we finally add the assumption that every instance of T can be characterized by its set of role values, then we can enumerate the instances of type T using a function (say t) that has an argument for each role value.

For example, consider the type AGE-RELS of age properties, which takes two roles, an *OBJECT* and a *VALUE*. Thus, the property P1 that captures the assertion "John is 10" would be described as follows:

(33) TYPE(P1,AGE-RELS) ∧
ROLE(P1,*OBJECT*,JOHN1) ∧
ROLE(P1,*VALUE*,10).

The type AGE-RELS satisfies the above properties, so any individual of type AGE-RELS with *OBJECT* role JOHN1 and *VALUE* role 10 is equal to P1. The retriever encodes such knowledge in a preprocessing stage that assigns each individual of type AGE-RELS to a canonical name. The canonical name for P1 would simply be "age-rels(JOHN1,10)".

Once a representation has equality, it can capture some of the distinctions made by perspectives in KRL. The same object viewed from two different perspectives is captured by two nodes, each with its own type, roles, and relations, that are asserted to be equal.

Note that one cannot expect more sophisticated reasoning about equality than the above from the retriever itself. Identifying two objects as equal is typically not a logical inference. Rather, it is a plausible inference by some specialized program such as the reference component of a natural language system which has to identify noun phrases. While the facts represented here would assist such a component in identifying possible referencts for a noun phrase given its description, it is unlikely that they would logically imply what the referent is.

6. Associations and Partitions

Semantic networks are useful because they structure information so that it is easy to retrieve relevant facts, or facts about certain objects. Objects are represented only once in the network, and thus there is one place where one can find all relations involving that object (by following back over incoming ROLE arcs). While we need to be able to capture such an ability in our system, we should note that this is often not a very useful ability, for much of one's knowledge about an object will not be attached to that object but will be acquired from the inheritance hierarchy. In a spreading activation type of framework, a considerable amount of irrelevant network will be searched before some fact high up in the type hierarchy is found. In addition, it is very seldom that one wants to be able to access all facts involving an object; it is much more likely that a subset of relations is relevant.

If desired, such associative links between objects can be simulated in our system. One could find all properties of an object o1 (including those by inheritance) by retrieving all bindings of x in the query

$$\exists x,r\ \mathrm{ROLE}(x,r,o1).$$

The ease of access provided by the links in a semantic network is effectively simulated simply by using a hashing scheme on the structure of all ROLE predicates. While the ability to hash on structures to find facts is crucial to an efficient implementation, the details are not central to our point here.

Another important form of indexing is found in Hendrix where his partition mechanism is used to provide a focus of attention for inference processes [Grosz, 1977]. This is just one of the uses of partitions. Another, which we did not need, provided a facility for scoping facts within logical operators, similar to the use of parentheses in FOPC. Such a focus mechanism appears in our system as an extra argument on the main predicates (e.g., HOLDS, OCCURS, etc.).

Since contexts are introduced as a new class of objects in the language, we can quantify over them and otherwise talk about them. In particular, we can organize contexts into a lattice-like structure (corresponding to Hendrix's vistas for partitions) by introducing a transitive relation SUBCONTEXT.

(A.8) $\forall$ c,c1,c2 SUBCONTEXT(c,c1) ∧
SUBCONTEXT(c1,c2)
→ SUBCONTEXT(c,c2)

To relate contexts to the HOLDS predicate, a proposition p holds in a context c only if it is known to hold in c explicitly, or it holds in a super context of c.

(A.9) $\forall$ p,t,c,c' SUBCONTEXT(c,c') ∧
HOLDS(p,c')
→ HOLDS(p,c).

As with the SUBTYPE relation, this axiom would defy an efficient implementation if the contexts were not organized in a finite lattice structure. Of course, we need axioms similar to (A.9) for the OCCURS and IS-REAL predicates.

7. Discussion

We have argued that the appropriate way to design knowledge representations is to identify those inferences that one wishes to facilitate. Once these are identified, one can then design a specialized limited inference mechanism that can operate on a data base of first order

facts. In this fashion, one obtains a highly expressive representation language (namely FOPC), as well as a well-defined and extendable retriever.

We have demonstrated this approach by outlining a portion of the representation used in ARGOT, the Rochester Dialogue System [Allen, 1982]. We are currently extending the context mechanism to handle time, belief contexts (based on a syntactic theory of belief [Haas, 1982]), simple hypothetical reasoning, and a representation of plans. Because the matcher is defined by a set of axioms, it is relatively simple to add new axioms that handle new features.

For example, we are currently incorporating a model of temporal knowledge based on time intervals [Allen, 1981a]. This is done by allowing any object, event, or relation to be qualified by a time interval as follows: for any untimed concept x, and any time interval t, there is a timed concept consisting of x viewed during t which is expressed by the term

(t-concept x t).

This concept is of type (TIMED Tx), where Tx is the type of x. Thus we require a type hierarchy of timed concepts that mirrors the hierarchy of untimed concepts.

Once this is done, we need to introduce new built-in axioms that extend the retriever. For instance, we define a predicate,

DURING(a,b),

that is true only if interval a is wholly contained in interval b. Now, if we want the retriever to automatically infer that if relation R holds during an interval t, then it holds in all subintervals of t, we need the following built-in axioms. First, DURING is transitive:

(A.10) $\forall$ a,b,c DURING(a,b) $\wedge$ DURING(b,c) $\rightarrow$ DURING(a,c)

Second, if P holds in interval t, it holds in all subintervals of t.

(A.11) $\forall$ p,t,t',c HOLDS(t-concept(p,t),c) $\wedge$ DURING(t',t) $\rightarrow$ HOLDS(t-concept(p,t'),c).

Thus we have extended our representation to handle simple timed concepts with only a minimal amount of analysis.

Unfortunately, we have not had the space to describe how to take the specification of the retriever (namely axioms (A.1) - (A.11)) and build an actual inference program out of it. A technique for building such a limited inference mechanism by moving to a meta-logic is described in [Frisch and Allen, 1982].

One of the more interesting consequences of this approach is that it has led to identifying various difference modes of retrieval that are necessary to support a natural language comprehension task. We have considered so far only one mode of retrieval, which we call *provability mode*. In this mode, the query must be shown to logically follow from the built-in axioms and the facts in the knowledge base. While this is the primary mode of interaction, others are also important.

In *consistency mode*, the query is checked to see if it is logically consistent with the facts in the knowledge base with respect to the limited inference mechanism. While consistency in general is undecidable, with respect to the limited inference mechanism it is computationally feasible. Note that, since the retriever is defined by a set of axioms rather than a program, consistency mode is easy to define.

Another important mode is *compatibility mode*, which is very useful for determining the referents of description. A query in compatibility mode succeeds if there is a set of equality and inequality assertions that can be assumed so that the query would succeed in provability mode. For instance, suppose someone refers to an event in which John hit someone with a hat. We would like to retrieve possible events that could be equal to this. Retrievals in compatibility mode are inherently expensive and so must be controlled using a context mechanism such as in [Grosz, 1977]. We are currently attempting to formalize this mode using Reiter's non-monotonic logic for default reasoning.

We have implemented a version of this system in HORNE [Allen and Frisch, 1981], a LISP embedded logic programming language. In conjunction with this representation is a language which provides many abbreviations and facilities for system users. For instance, users can specify what context and times they are working with respect to, and then omit this information from their interactions with the system. Also, using the abbreviation conventions, the user can describe a relation and events without explicitly asserting the TYPE and ROLE assertions. Currently the system provides the inheritance hierarchy, simple equality reasoning, contexts, and temporal reasoning with the DURING hierarchy.

Acknowledgments

This research was supported in part by the National Science Foundation under Grant IST-80-12418, and in part by the Office of Naval Research under Grant N00014-80-C-0197.

References

Allen, J.F., "ARGOT: A system overview," TR 101, Computer Science Dept., U. Rochester, 1982.

Allen, J.F., "An interval-based representation of temporal knowledge," *Proc.*, 7th IJCAI, Vancouver, B.C., August 1981a.

Allen, J.F., "What's necessary to hide?: Reasoning about action verbs," *Proc.*, 19th ACL, Stanford U., 1981b.

Allen, J.F. and A.M. Frisch, "HORNE user's manual," Computer Science Dept., U. Rochester, 1981.

Bobrow, D.G. and T. Winograd, "An overview of KRL, a knowledge representation language," *Cognitive Science 1*, 3-46, 1977.

Brachman, R.J., "On the epistemological status of semantic networks," in N.V. Findler, 1979.

Charniak, E., "A common representation for problem-solving and language-comprehension information," *Artificial Intelligence 16*, 3, 225-255, July 1981a.

Charniak, E., "The case-slot identity theory," *Cognitive Science 5*, 3, 1981b.

Davidson, D., "The logical form of action sentences," in N. Rescher (Ed). *The Logic of Decision and Action.* Pittsburgh, PA: U. Pittsburgh Press, 1967.

Fillmore, C.J., "The case for case," in E. Bach and R. Harmes (Eds). *Universals in Linguistic Theory.* New York: Holt, Rinehart and Winston, 1968.

Findler, N.V. (Ed). *Associative Networks: Representation and Use of Knowledge by Computers.* New York: Academic Press, 1979.

Frisch, A.M. and J.F. Allen, "Knowledge retrieval as limited inference," *Proc.*, 6th Conf. on Automated Deduction, New York, June 1982.

Grosz, B.J., "The representation and use of focus in dialogue understanding," TN 151, SRI, July 1977.

Haas, A., "Mental states and mental actions in planning," Ph.D. thesis, Computer Science Dept., U. Rochester, 1982.

Hayes, P.J., "The logic of frames," in D. Metzing (Ed). *Frame Conceptions and Text Understanding.* Walter de Gruyter & Co., 1979.

Hendrix, G.G., "Encoding knowledge in partitioned networks," in N.V. Findler, 1979.

Kowalski, R.A. *Logic for Problem Solving.* New York: North Holland, 1979.

Levesque, H. and J. Mylopolous, "A procedural semantics for semantic networks," in N.V. Findler, 1979.

Nilsson, N.J. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Co., 1980.

Perlis, D., "Language, computation, and reality," Ph.D. thesis, Computer Science Dept., U. Rochester, 1981.

Reiter, R., "A logic for default reasoning," *Artificial Intelligence 13*, 1-2, 81-132, April 1980.

Shapiro, S. C., "The SNePS semantic network processing system," in N.V. Findler, 1979.

Shapiro, S. C., "A net structure for semantic information storage, deduction and retrieval," *Proc.*, IJCAI, 1971.

Tarjan, R.E., "Efficiency of a good but not linear set union algorithm," *JACM 22*, 2, April 1975.

Woods, W. A., "What's in a link: Foundations for semantic networks," in D.G. Bobrow and A.M. Collins (Eds). *Representation and Understanding.* New York: Academic Press, 1975.