## UNIT 4

## KNOWLEDGE REPRESENTATION AND REASONING

Knowledge Representation – Ontological Engineering- Categories and Objects – Events – Mental Events and Mental Objects – Reasoning Systems for Categories – Reasoning with Default Information.

### 4.1 KNOWLEDGE REPRESENTATION ONTOLOGICAL ENGINEERING

Concepts such as Events, Time, Physical Objects, and Beliefs— that occur in many different domains. Representing these abstract concepts is sometimes called ontological engineering.

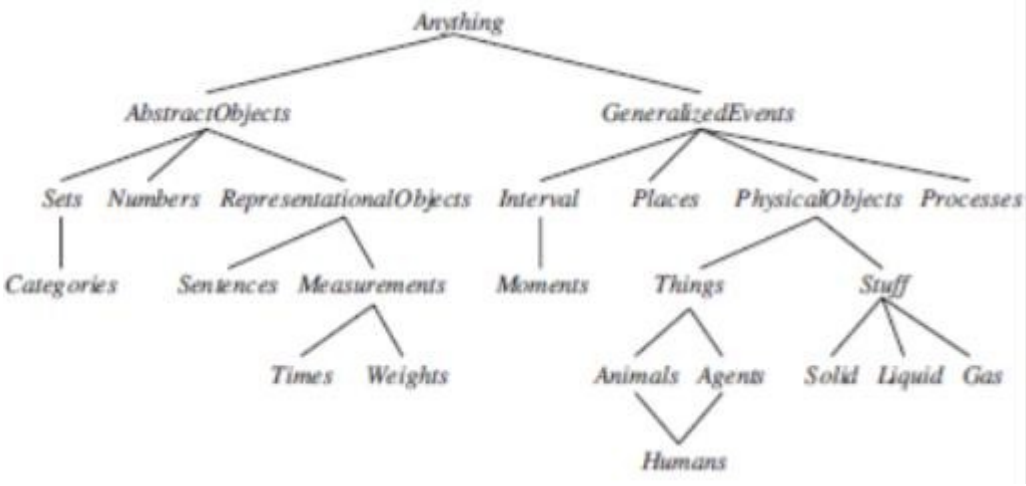


**Figure 4.1 The upper ontology of the world, showing the topics to be covered later in the chapter. Each link indicates that the lower concept is a specialization of the upper one. Specializations are not necessarily disjoint; a human is both an animal and an agent, for example.**

The general framework of concepts is called an upper ontology because of the convention of drawing graphs with the general concepts at the top and the more specific concepts below them, as in Figure

### 4.2 Categories and Objects

The organization of objects into **categories** is a vital part of knowledge representation. Although interaction with the world takes place at the level of individual objects, much reasoning takes place at the level of categories.

For example, a shopper would normally have the goal of buying a **basketball,** rather than a particular basketball such as BB9 There are two choices for representing categories in first-order logic: predicates and objects. That is, we can use the predicate ***Basketball (b),*** or we

can reify1 the category as an object, Basketballs.

We could then say ***Member(b, Basketballs*** ), which we will abbreviate as ***b*** ∈ ***Basketballs***, to say that b is a member of the category of basketballs. We say ***Subset(Basketballs, Balls),*** abbreviated as ***Basketballs*** ⊂ ***Balls***, to say that Basketballs is a subcategory of Balls. Categories serve to organize and simplify the knowledge base through inheritance. If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and ***Apples*** is a subclass of Fruit, then we can infer that every apple is edible. We say that the individual apples inherit the property of edibility, in this case from their membership in the Food category. First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members. Here are some types of facts, with examples of each:

- An object is a member of a category.

***BB9*** ∈ ***Basketballs***

- A category is a subclass of another category. ***Basketballs*** ⊂ ***Balls***
- All members of a category have some properties.

***(x*** ∈ ***Basketballs)*** ⇒ ***Spherical (x)***

- Members of a category can be recognized by some properties. Orange(x) ∧ Round (x) ∧ Diameter(x)=9.5 ∧ x∈ Balls ⇒ x∈ Basketballs
- A category as a whole has some properties.

**Dogs** ∈ **Domesticated Species**

Notice that because Dogs is a category and is a member of Domesticated Species, the latter must be a category of categories. Categories can also be defined by providing necessary and sufficient conditions for membership. For example, a bachelor is an unmarried adult male:

***x*** ∈ ***Bachelors*** ⇔ ***Unmarried(x)*** ∧ ***x*** ∈ ***Adults*** ∧ ***x*** ∈ ***Males***

**Physical Composition**

We use the general PartOf relation to say that one thing is part of another. Objects can be grouped into part of hierarchies, reminiscent of the Subset hierarchy:

***PartOf (Bucharest, Romania)***
***PartOf (Romania, EasternEurope)***
***PartOf(EasternEurope, Europe)***
***PartOf (Europe, Earth)***
***The PartOf relation is transitive and reflexive; that is,***
***PartOf (x, y)*** ∧***PartOf (y, z)*** ⇒***PartOf (x, z)***
***PartOf (x, x)***
Therefore, we can conclude PartOf (Bucharest, Earth).

For example, if the apples are Apple1, Apple2, and Apple3, then

***BunchOf ({Apple1,Apple2,Apple3})***

denotes the composite object with the three apples as parts (not elements). We can define ***BunchOf in terms of the PartOf relation. Obviously, each element of s is part of***

***BunchOf (s): ∀x x∈ s ⇒PartOf (x, BunchOf (s)) Furthermore, BunchOf (s)*** is the smallest object satisfying this condition. In other words, BunchOf (s) must be part of any object that has all the elements of s as parts:

***∀y [∀x x∈ s ⇒PartOf (x, y)] ⇒PartOf (BunchOf (s), y)***

Measurements

In both scientific and commonsense theories of the world, objects have height, mass, cost, and so on. The values that we assign for these properties are called measures. ***Length(L1)=Inches(1.5)=Centimeters(3.81)***

Conversion between units is done by equating multiples of one unit to another: ***Centimeters(2.54 ×d)=Inches(d)***

Similar axioms can be written for pounds and kilograms, seconds and days, and dollars and cents. Measures can be used to describe objects as follows:

***Diameter (Basketball12)=Inches(9.5)***

***ListPrice(Basketball12)=$(19)***

***d∈ Days ⇒ Duration(d)=Hours(24)***

**Time Intervals**

Event calculus opens us up to the possibility of talking about time, and time intervals. We will consider two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:

***Partition({Moments,ExtendedIntervals}, Intervals )***

***i∈Moments⇔Duration(i)=Seconds(0)***

The functions Begin and End pick out the earliest and latest moments in an interval, and the function Time delivers the point on the time scale for a moment.

The function Duration gives the difference between the end time and the start time.

***Interval (i) ⇒Duration(i)=(Time(End(i)) − Time(Begin(i)))***

***Time(Begin(AD1900))=Seconds(0)***

***Time(Begin(AD2001))=Seconds(3187324800)***

***Time(End(AD2001))=Seconds(3218860800)***

***Duration(AD2001)=Seconds(31536000)***

Two intervals Meet if the end time of the first equals the start time of the second. The complete set of interval relations, as proposed by Allen (1983), is shown graphically in Figure 12.2 and logically below:

***Meet(i,j) ⇔ End(i)=Begin(j)***

***Before(i,j) ⇔ End(i) < Begin(j)***

***After (j,i) ⇔ Before(i, j)***

***During(i,j) ⇔ Begin(j) < Begin(i) < End(i) < End(j)***

***Overlap(i,j) ⇔ Begin(i) < Begin(j) < End(i) < End(j)***

***Begins(i,j) ⇔ Begin(i) = Begin(j)***

***Finishes(i,j) ⇔ End(i) = End(j)***

***Equals(i,j) ⇔ Begin(i) = Begin(j) ∧ End(i) = End(j)***

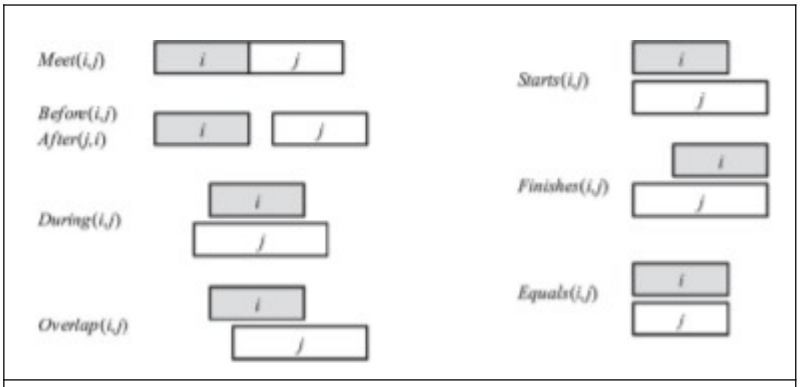


**Figure 4.2 Predicates on time intervals**

## 4.3 EVENTS

Event calculus reifies fluents and events. The fluent **At(Shankar, Berkeley)** is an object that refers to the fact of Shankar being in Berkeley, but does not by itself say anything about whether it is true. To assert that a fluent is actually true at some point in time we use the predicate T, as in **T(At(Shankar, Berkeley),** t). Events are described as instances of event categories. The event E1 of Shankar flying from San Francisco to Washington, D.C. is described as **E1 ∈Flyings∧ Flyer (E1, Shankar ) ∧ Origin(E1, SF) ∧ Destination (E1,DC)** we can define an alternative three-argument version of the category of flying events and say ***E1 ∈Flyings(Shankar, SF,DC)*** We then use ***Happens(E1, i)*** to say that the event E1 took place over the time interval i, and we say the same thing in functional form with Extent(***E1)=i.*** We represent time intervals by a (start, end) pair of times; that is, i = (t1, t2) is the time interval that starts at t1 and ends at t2. The complete set of predicates for one version of the event calculus is T(f, t) Fluent f is true at time t Happens(e, i) Event e happens over the time interval i Initiates(e, f, t) Event e causes fluent f to start to hold at time t Terminates(e, f, t) Event e causes fluent f to cease to hold at time t Clipped(f, i) Fluent f ceases to be true at some point during time interval i Restored (f, i) Fluent f becomes true sometime during time interval i We

assume a distinguished event, Start, that describes the initial state by saying which fluents are initiated or terminated at the start time. We define T by saying that a fluent holds at a point in time if the fluent was initiated by an event at some time in the past and was not made false (clipped) by an intervening event. A fluent does not hold if it was terminated by an event and not made true (restored) by another event. Formally, the axioms are:

***Happens(e, (t1, t2)) ∧Initiates(e, f, t1) ∧ ¬ Clipped(f, (t1, t)) ∧ t1 < t ⇒T(f, t)Happens(e, (t1, t2)) ∧ Terminates(e, f, t1)∧¬Restored (f, (t1, t)) ∧ t1 < t ⇒¬T(f, t)***

***where Clipped and Restored are defined by Clipped(f, (t1, t2)) ⇔∃ e, t, t3 Happens(e, (t, t3))∧ t1 ≤ t < t2 ∧ Terminates(e, f, t) Restored (f, (t1, t2)) ⇔∃ e, t, t3 Happens(e, (t, t3)) ∧ t1 ≤ t < t2 ∧ Initiates(e, f, t)***

## 4.4 MENTAL EVENTS AND MENTAL OBJECTS

What we need is a model of the mental objects that are in someone's head (or something's knowledge base) and of the mental processes that manipulate those mental objects. The model does not have to be detailed. We do not have to be able to predict how many milliseconds it will take for a particular agent to make a deduction. We will be happy just to be able to conclude that mother knows whether or not she is sitting.

We begin with the propositional attitudes that an agent can have toward mental objects: attitudes such as Believes, Knows, Wants, Intends, and Informs. The difficulty is that these attitudes do not behave like "normal" predicates.

For example, suppose we try to assert that Lois knows that Superman can fly: **Knows (Lois, CanFly(Superman))** One minor issue with this is that we normally think of CanFly(Superman) as a sentence, but here it appears as a term. That issue can be patched up just be reifying **CanFly(Superman);** making it a fluent. A more serious problem isthat, if it is true that Superman is Clark Kent, then we must conclude that Lois knows that Clark can fly: **(Superman = Clark) ∧Knows(Lois, CanFly(Superman)) |= Knows(Lois, CanFly (Clark))** Modal logic is designed to address this problem. Regular logic is concerned with a single modality, the modality of truth, allowing us to express "P is true." Modal logic includes special modal operators that take sentences (rather than terms) as arguments.

For example, "A knows P" is represented with the notation KAP, where K is the modal operator for knowledge. It takes two arguments, an agent (written as the subscript) and a sentence. The syntax of modal logic is the same as first-order logic, except that sentences can also be formed with modal operators. In first-order logic a model contains a set of objects and an interpretation that maps each name to the appropriate object, relation, or function. In modal logic we want to be able to consider both the possibility that Superman's secret identity is Clark and that it isn't. Therefore, we will need a more complicated model, one that consists of a collection of possible worlds rather than just one true world. The worlds are connected in a graph by accessibility relations, one relation for each modal operator. We say that world w1 is accessible from world w0 with respect to the modal operator KA if everything in w1 is

consistent with what A knows in w0, and we write this as **Acc(KA,w0,w1).** In diagrams such as Figure 12.4 we show accessibility as an arrow between possible worlds. In general, a knowledge atom KAP is true in world w if and only if P is true in every world accessible from w. The truth of more complex sentences is derived by recursive application of this rule and the normal rules of first-order logic. That means that modal logic can be used to reason about nested knowledge sentences: what one agent knows about another agent's knowledge. For example, we can say that, even though Lois doesn't know whether Superman's secret identity is Clark Kent, she does know that Clark knows: **KLois [KClark Identity(Superman, Clark ) ∨KClark¬Identity(Superman, Clark )]** Figure 3.15 shows some possible worlds for this domain, with accessibility relations for Lois and Superman

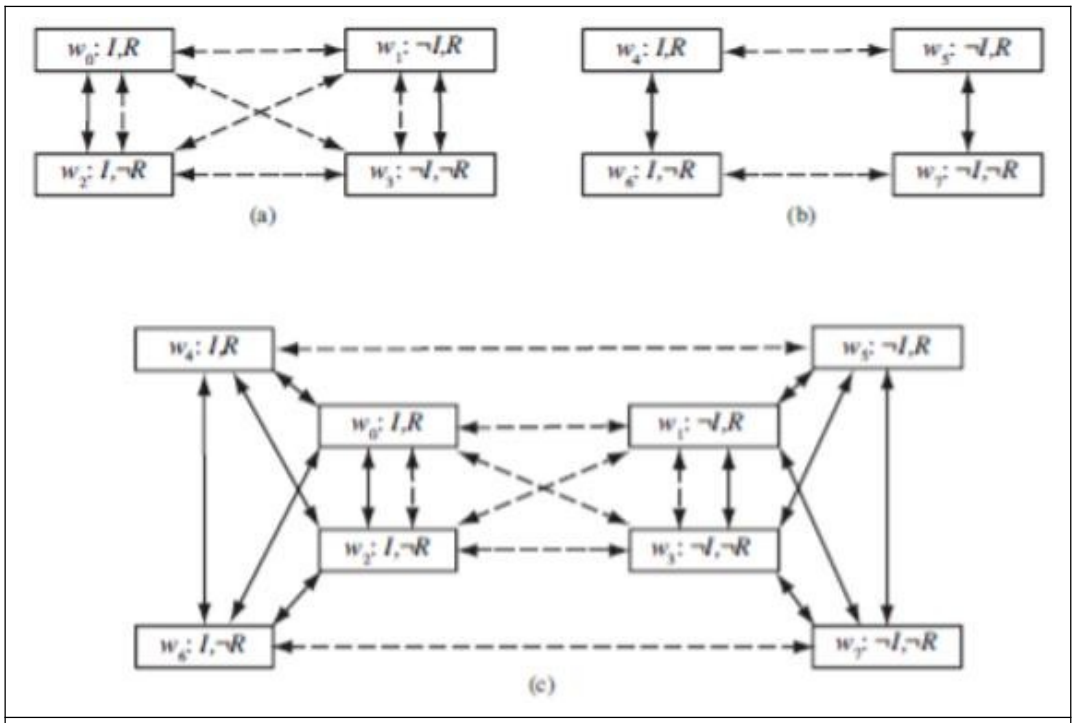


**Figure 4.4**

In the TOP-LEFT diagram, it is common knowledge that Superman knows his own identity, and neither he nor Lois has seen the weather report. So in w0 the worlds w0 and w2 are accessible to Superman; maybe rain is predicted, maybe not. For Lois all four worlds are accessible from each other; she doesn't know anything about the report or if Clark is Superman. But she does know that Superman knows whether he is Clark, because in every world that is accessible to Lois, either Superman knows I, or he knows ¬I. Lois does not know which is the case, but either way she knows Superman knows. In the TOP-RIGHT diagram it is common knowledge that Lois has seen the weather report. So in w4 she knows rain is predicted and in w6 she knows rain is not predicted. Superman does not know the report, but he knows that Lois knows, because in every world that is accessible to him, either she knows R or she knows ¬ R. In the BOTTOM diagram we represent the scenario where it is common knowledge that Superman knows his identity, and Lois might or might not have seen the weather report. We

represent this by combining the two top scenarios, and adding arrows to show that Superman does not know which scenario actually holds. Lois does know, so we don't need to add any arrows for her. In w0 Superman still knows I but not R, and now he does not know whether Lois knows R. From what Superman knows, he might be in w0 or w2, in which case Lois does not know whether R is true, or he could be in w4, in which case she knows R, or w6, in which case she knows ¬R.

## 4.5 REASONING SYSTEMS FOR CATEGORIES

This section describes systems specially designed for organizing and reasoning with categories. There are two closely related families of systems: semantic networks provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership; and description logics provide a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

### 4.5.1 SEMANTIC NETWORKS

There are many variants of semantic networks, but all are capable of representing individual objects, categories of objects, and relations among objects. A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links. For example, Figure 12.5 has a Member Of link between Mary and Female Persons, corresponding to the logical assertion Mary ∈FemalePersons ; similarly, the SisterOf link between Mary and John corresponds to the assertion SisterOf (Mary, John). We can connect categories using SubsetOf links, and so on. We know that persons have female persons as mothers, so can we draw a HasMother link from Persons to FemalePersons? The answer is no, because HasMother is a relation between a person and his or her mother, and categories do not have mothers. For this reason, we have used a special notation—the double-boxed link—in Figure 12.5. This link asserts that ∀x x∈ Persons ⇒ [∀ y HasMother (x, y) ⇒ y ∈FemalePersons ] We might also want to assert that persons have two legs—that is, ∀x x∈ Persons ⇒ Legs(x, 2) The semantic network notation makes it convenient to perform inheritance reasoning. For example, by virtue of being a person, Mary inherits the property of having two legs. Thus, to find out how many legs Mary has, the inheritance algorithm followsthe MemberOf link from Mary to the category she belongs to, and then follows SubsetOf links up the hierarchy until it finds a category for which there is a boxed Legs link—in this case, the Persons category.

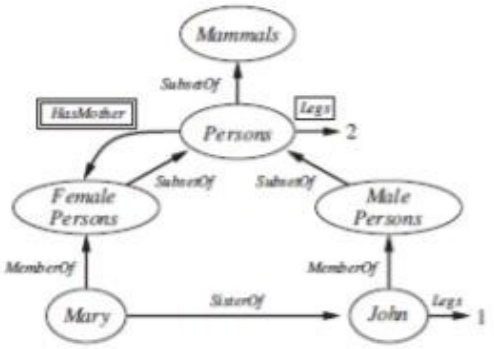


**Figure 4.5 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.**

Inheritance becomes complicated when an object can belong to more than one category or when a category can be a subset of more than one other category; this is called multiple inheritance. The drawback of semantic network notation, compared to first-order logic: the fact that links between bubbles represent only binary relations. For example, the sentence Fly(Shankar, NewYork, NewDelhi, Yesterday) cannot be asserted directly in a semantic network. Nonetheless, we can obtain the effect of n-ary assertions by reifying the proposition itself as an event belonging to an appropriate event category. Figure 4.6 shows the semantic network structure for this particular event. Notice that the restriction to binary relations forces the creation of a rich ontology of reified concepts.

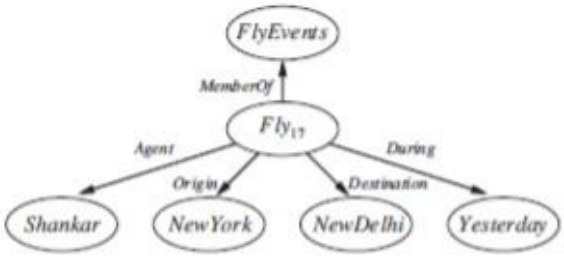


**Figure 4.6 A fragment of a semantic network showing the representation of the logical assertion *Fly(Shankar, NewYork, NewDelhi, Yesterday)***

One of the most important aspects of semantic networks is their ability to represent default values for categories. Examining Figure4.5 carefully, notice that John has one leg, despite the fact that he is a person and all persons have two legs. In a strictly logical KB, this would be a contradiction, but in a semantic network, the assertion that all persons have two legs has only default status; that is, a person is assumed to have two legs unless this is contradicted by more specific information