Unit 8 Structured Knowledge Representation – I: Weak Slot-and-Filler Structures

Structure:

8.1 Introduction

In the last unit you have learnt the inference rules for propositional logic. You also learnt the steps to transform a sentence into Clausal Form and resolution principle, its algorithm and strategies. In this unit, we will discuss the structured knowledge representation. The representations considered up to this point focused primarily on expressiveness, validity, consistency, inference methods and related topics. Little consideration was given to the way in which the knowledge was structured and how it might be viewed by designers, or to the type of data structures that should be used internally. Neither was much consideration given to effective methods of organizing the knowledge structures in memory. In this unit we will address such problem by discussing Semantic Networks, Partitioned Semantic Networks and Frames.

Objectives:

After studying this unit, you should be able to:

- explain semantic networks
- define Reification
- explain partitioned semantic networks
- define frames
- list the advantages and disadvantages of frames.

8.2 Semantic Nets

Semantic networks are an alternative to predicate logic as a form of knowledge representation. The idea is that we can store our knowledge in the form of a graph, with nodes representing objects in the world, and arcs representing relationships between those objects.

The major idea is that:

- The meaning of a concept comes from its relationship to other concepts, and that,
- The information is stored by interconnecting nodes with labeled arcs.

For example, the figure 8.1 is intended to represent the data:
Tom is a cat.
Tom caught a bird.
Tom is owned by John.
Tom is ginger in colour.
Cats like cream.
The cat sat on the mat.
A cat is a mammal.
A bird is an animal.
All mammals are animals.
Mammals have fur.

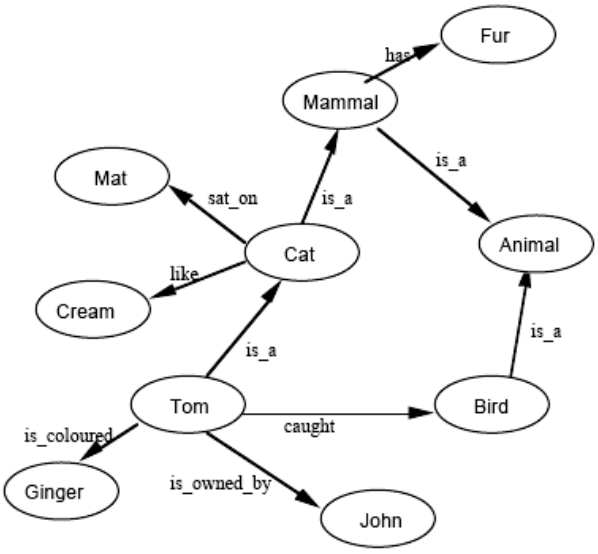

Figure 8.1: An example of Semantic Networks

It is argued that this form of representation is closer to the way humans structure knowledge by building mental links between things than the predicate logic we considered earlier. Note in particular how all the information about a particular object is concentrated on the node representing that object, rather than scattered around several clauses in logic. There is, however, some confusion here which stems from the imprecise nature of semantic nets. A particular problem is that we haven't distinguished between nodes representing classes of things, and nodes representing individual objects. So, for example, the node labeled Cat represents both the single (nameless) cat who sat on the mat, and the whole class of cats to which Tom belongs, which are mammals and which like cream. The is_a link has two different meanings – it can mean that one object is an individual item from a class, for example Tom is a member of the class of cats, or that one class is a subset of another, for example, the class of cats is a subset of the class of mammals. This confusion does not occur in logic, where the use of quantifiers, names and predicates makes it clear what we mean so:

Tom is a cat is represented by Cat(Tom)

The cat sat on the mat is represented by □ ∃x ∃□y(Cat(x) ∧Mat(y) ∧ SatOn(x,y))

A cat is a mammal is represented by ∀□x(Cat(X) →Mammal(x))

We can clean up the representation by distinguishing between nodes representing individual or instances, and nodes representing *classes*. The is_a link will only be used to show an individual belonging to a class. The link representing one class being a subset of another will be labeled a_kind_of, or ako for short. The names instance and subclass are often used in the place of is_a and ako, but we will use these terms with a slightly different meaning in the section on Frames below.

Note also the modification which causes the link labeled is_owned_by to be reversed in direction. This is in order to avoid links representing passive relationships. In general a passive sentence can be replaced by an active one, so "Tom is owned by John" becomes "John owns Tom". In general the rule which converts passive to active in English converts sentences of the form "X is Yed by Z" to "Z Ys X". This is just an example (though often used

for illustration) of the much more general principle of looking beyond the immediate *surface structure* of a sentence to find its *deep structure*.

The revised semantic net is shown in figure 8.2:

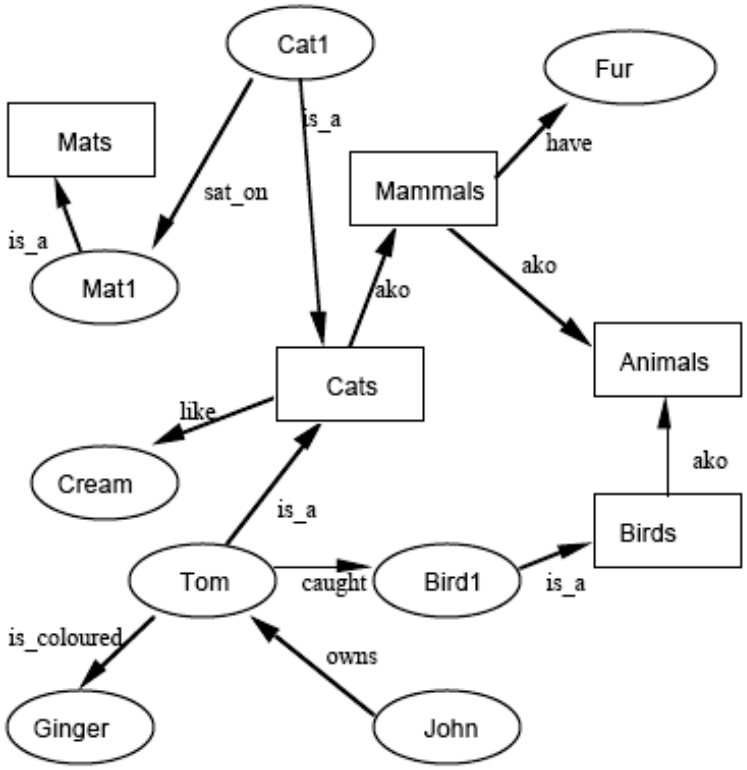

Figure 8.2: A Revised Semantic Networks

Note that where we had an unnamed member of some class, we have had to introduce a node with an invented name to represent a particular member of the class. This is a process similar to the Skolemisation we considered previously as a way of dealing with existential quantifiers. For example, “Tom caught a bird” would be represented in logic by x (bird(x) ^ caught(Tom,x)), which would be Skolemised by replacing the x with a Skolem constant; the same thing was done above where bird1 was the name given to the individual bird that Tom caught. There are still plenty of issues to be resolved if we really want to represent what is meant by the English phrases, or to be really clear about what the semantic net means, but we are getting towards a notation that can be used practically (one example of a thing we have skated over is how to deal with mass nouns like

"fur" or "cream" which refer to things that come in amounts rather than individual objects).

A direct Prolog representation can be used, with classes represented by predicates, thus:

```
cat(tom).
cat(cat1).
mat(mat1).
sat_on(cat1,mat1).
bird(bird1).
caught(tom,bird1).
like(X,cream) :– cat(X).
mammal(X) :– cat(X).
has(X,fur) :– mammal(X).
animal(X) :– mammal(X).
animal(X) :– bird(X).
owns(john,tom).
is_coloured(tom,ginger).
```

So, in general, an is_a link between a class c and an individual m is represented by the fact c(m). An a_kind_of link between a subclass c and a superclass s is represented by s(X) :- c(X). If a property p with further arguments a_1, … ,a_n is held by all members of a class c, it is represented by p(X,a1,…,an) :- c(X). If a property p with further arguments a_1, … ,a_n is specified as held by an individual m, rather than a class to which m belongs, it is represented by p(m,a_1,…,a_n).

The physical attributes of a person can be represented as in figure. 8.3.

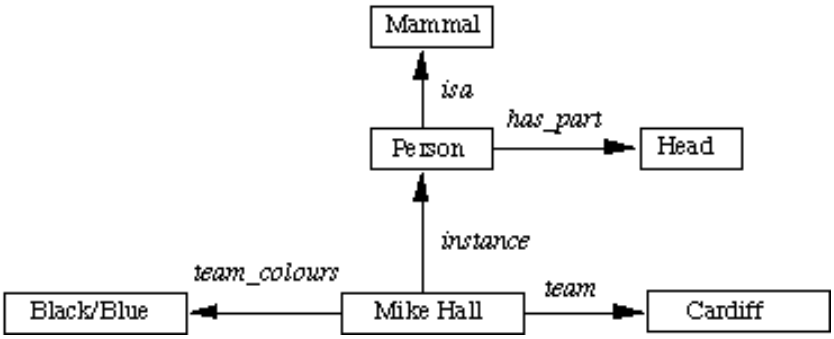

Figure 8.3: A Semantic Network for the attributes of a person

These values can also be represented in logic as: *isa(person, mammal), instance(Mike-Hall, person) team(Mike-Hall, Cardiff)*

As a more complex example consider the sentence: *John gave Mary the book*(refer figure 8.4). Here we have several aspects of an event.

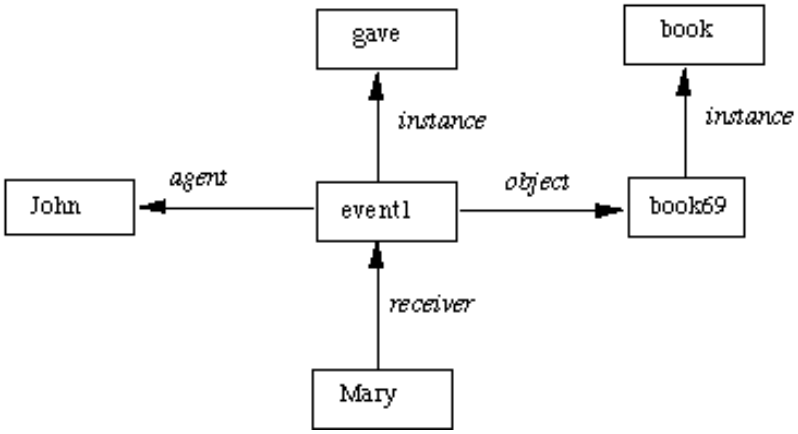

Figure 8.4: A Semantic Network for a Sentence

Inheritance

This Prolog equivalent captures an important property of semantic nets, that they may be used for a form of inference known as *inheritance*. The idea of this is that if an object belongs to a class (indicated by an is_a link) it *inherits* all the properties of that class. So, for example as we have a likes link between cats and cream, meaning “all cats like cream”, we can infer that any object which has an is_a link to cats will like cream. So both Tom and Cat1 like cream. However, the is_coloured link is between Tom and ginger, not between cats and ginger, indicating that being ginger is a property of Tom as an individual, and not of all cats. We cannot say that Cat1 is ginger, for example; if we wanted to we would have to put another is_coloured link between Cat1 and ginger.

Inheritance also applies across the a_kind_of links. For example, any property of mammals or animals will automatically be a property of cats. So we can infer, for example, that Tom has fur, since Tom is a cat, a cat is a kind of mammal, and mammals have fur. If, for example, we had another subclass of mammals, say dogs, and we had, say, Fido is_a dog, Fido

would inherit the property has fur from mammals, but not the property likes cream, which is specific to cats. This situation is shown in the figure 8.5:

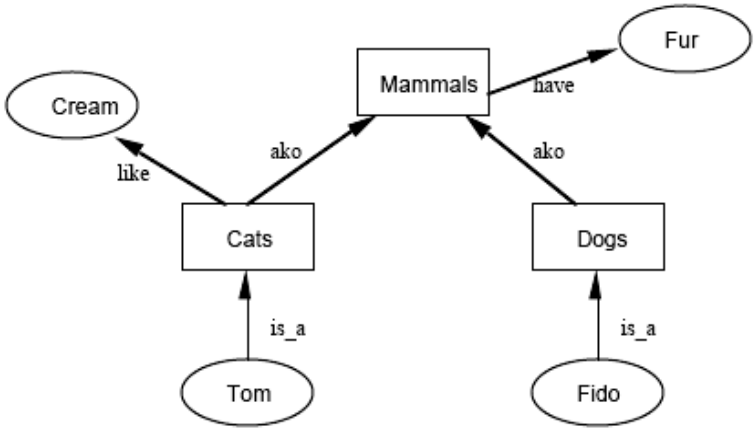

Figure 8.5: Semantic Networks showing Inheritance

Inheritance also provides a means of dealing with *default reasoning, e.g.* we could represent:

- Emu are birds.
- Typically birds fly and have wings.
- Emus run.

in the Semantic net shown in figure 8.6.

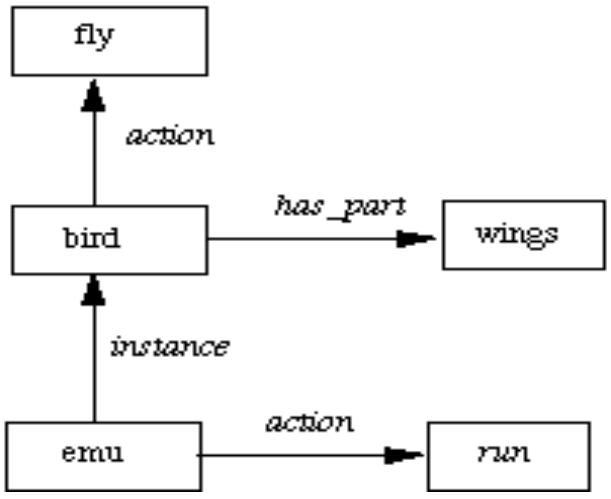

Figure 8.6: A Semantic Network for a Default Reasoning

In making certain inferences we will also need to *distinguish between the link that defines a new entity and holds its value and the other kind of link that relates two existing entities*. Consider the example shown in figure 8.7 where the height of two people is depicted and we also wish to compare them.

We need extra nodes for the concept as well as its value.

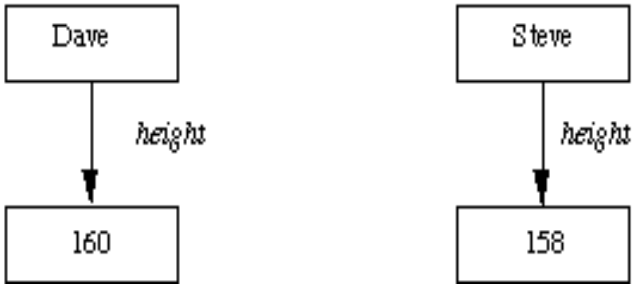

Figure 8.7: Two heights

Special procedures are needed to process these nodes, but without this distinction the analysis would be very limited. The figure 8.8 gives the comparison of two heights.

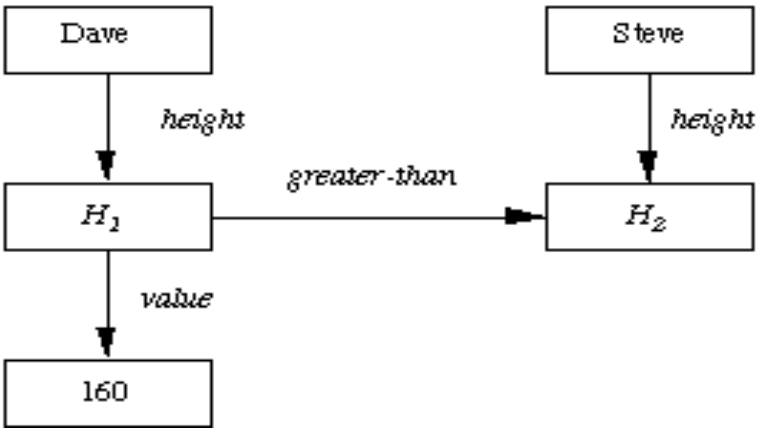

Figure 8.8: Comparison of two heights

Reification

An alternative form of representation considers the semantic network directly as a graph. We could represent each edge in the semantic net graph by a fact whose predicate name is the label on the edge. The nodes

in this graph, whether they represent individuals or classes are represented by arguments to the facts representing edges. This gives the following representation for our initial graph:

```
is_a(mat1,mats).
is_a(cat1,cats).
is_a(tom,cats).
is_a(bird1,birds).
caught(tom,bird1).
ako(cats,mammals).
ako(mammals,animals).
ako(birds,animals).
like(cats,cream).
owns(john,tom).
sat_on(cat1,mat1).
is_coloured(tom,ginger).
have(mammals,fur).
```

Alternatively, the graph could be built using the cells or pointers of an imperative language. There are also special purpose knowledge representation languages which provide a notation which translates directly to this sort of graph. This process of turning a predicate into an object in a knowledge representation system is known as *reification*. So, for example, the constant symbol cats represents the set of all cats, which we can treat as just another object.

Self Assessment Questions

1. _____ networks are an alternative to predicate logic as a form of knowledge representation.
2. _____ applies across the a_kind_of links.
3. What is the process of turning a predicate into an object in a knowledge representation system called?

   _____

8.3 Partitioned Semantic Networks

Partitioned Semantic Networks is an extension to Semantic nets that overcome a few problems or extend their expression of knowledge.

Partitioned Semantic Networks allow for:

- propositions to be made without commitment to truth.
- expressions to be quantified.

Basic idea: *Break network into **spaces** which consist of groups of nodes and arcs and regard each **space** as a node.*

Consider the following: *Andrew believes that the earth is flat.* We can encode the proposition *the earth is flat* in a *space* and within it have nodes and arcs that represent the fact (Refer figure 8.9). We can then have nodes and arcs to link this *space* to the rest of the network to represent Andrew's belief.

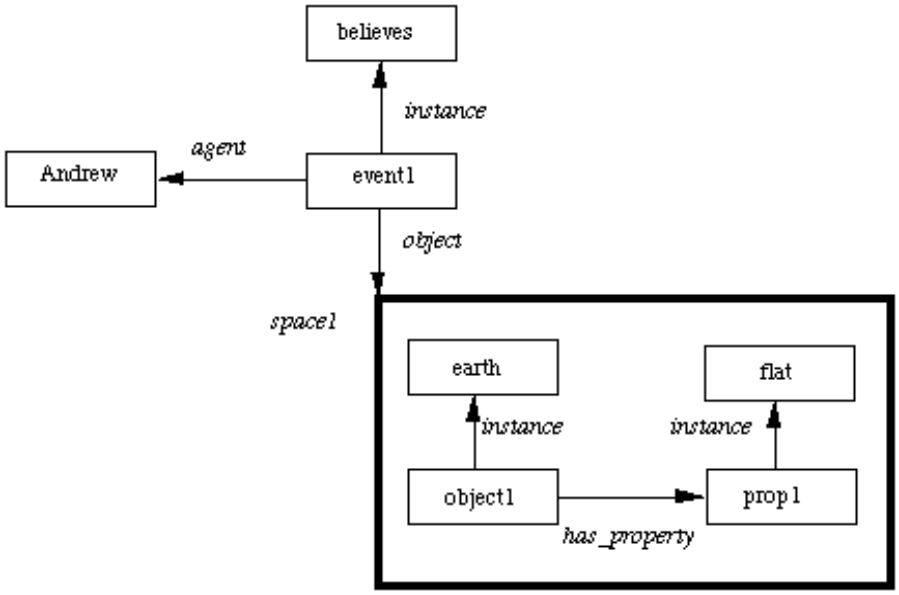

Figure 8.9: Partitioned network

Now consider the quantified expression: *Every parent loves his/her child.* To represent this we:

- Create a *general statement*, GS, special class.
- Make node g an instance of GS.
- Every element will have at least 2 attributes:
  - a *form* that states which relation is being asserted.

- one or more *for all* ($\forall$) or *exists* ($\exists$) connections -- these represent universally quantifiable variables in such statements *e.g. x*, *y* in $\forall$ x *parent(x)* $\rightarrow \exists$ y: *child(y)* $\wedge$ *loves(x,y)*

Here we have to construct two *spaces one* for each *x*,*y*. We can express $\exists$ variables as *existentially qualified* variables and express the event of *love* having an agent *p* and receiver *b* for every parent *p* which could simplify the network.

Also If we change the sentence to *Every parent loves child* then the node of the object being acted on (*the child*) lies outside the form of the general statement. Thus it is not viewed as an existentially qualified variable whose value may depend on the agent. So we could construct a partitioned network as in Figure 8.10.

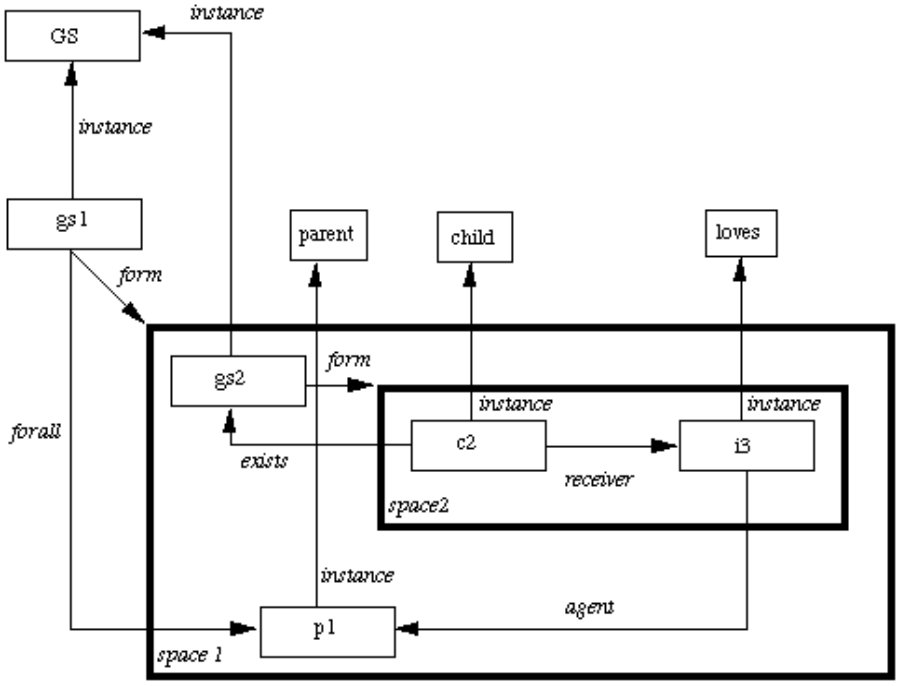

Figure 8.10: Partitioned network

8.4 Frames

Frames can also be regarded as an extension to Semantic nets. Indeed it is not clear where the distinction between a semantic net and a frame ends. Semantic nets were initially used to represent labeled connections between

objects. As tasks became more complex the representation needs to be more structured. The more structured the system, it becomes more beneficial to use frames.

A *frame* is a collection of attributes or slots and associated values that describe some real world entity. Frames on their own are not particularly helpful but frame systems are a powerful way of encoding information to support reasoning. Set theory provides a good basis for understanding frame systems.

A basic idea of frames is that people make use of stereotyped information about typical features of objects, images, and situations; such information is assumed to be structured in large units representing the stereotypes, and these units are what are referred to as “frames”.

Frames (or something similar) are important because they allow deep understanding of new situations about which only minimal information is directly available. They represent our understanding of regularities in the universe that allow intelligent action based on minimal clues.

Each frame represents:

- a class (set), or
- an instance (an element of a class).

Typical features of Frames

A frame can represent an *individual* object or a *class* of similar objects.

Instead of properties, a frame has *slots*. A slot is like a property, but can contain more kinds of information (sometimes called *facets* of the slot):

- The value of the slot; default value in case no value is present.
- A procedure that can be run to compute the value (an *if-needed procedure*).
- Procedures to be run when a value is put into the slot or removed (*if-added* and *if-removed* procedures). These can be used to implement *demons*.
- Data type information; constraints on possible slot fillers.
- Documentation.

Advantages of Frames

1) A frame collects information about an object in a single place in an organized fashion. (Cf. Logic, which represents information about an object by many small predicates scattered throughout the database.)
2) By relating slots to other kinds of frames, a frame can represent *typical* structures involving an object; these can be very important for reasoning based on limited information.
3) Frames provide a way of associating *knowledge* with objects (via the slot procedures).
4) Frames may be a relatively efficient way of implementing AI applications (direct procedure invocation versus search in a logic system).
5) Frames allow data that are stored and computed to be treated in a uniform manner. (e.g., AGE might be stored, or might be computed from BIRTHDAY.)
6) *Object-oriented programming* has much in common with frames.

Disadvantages of Frames

Some things that can be represented in logic cannot be represented well or at all in frames:

1) Slot fillers must be “real” data.
2) It is not possible to quantify over slots. For example, there is no way to represent “Some student made 100 on the exam”.
3) It is necessary to repeat the same information to make it usable from different viewpoints, since methods are associated with slots or particular object types.

When to use Frames

Frames are good for applications in which the structure of the data and the available information are relatively fixed. The procedures (*methods*) associated with slots provide a good way to associate knowledge with a class of objects. Frames are *not* particularly good for applications in which deep deductions are made from a few principles (as in theorem proving).

Consideration of the use of cases suggests how we can tighten up on the semantic net notation to give something which is more consistent, known as the *frame* notation. In the place of an arbitrary number of arcs leading from a node there are a fixed number of slots representing attributes of an object. Every object is a member or *instance* of a class, which it may be thought of

as linking to an is_a link as we saw before. The class indicates the number of slots that an object has, and the name of each slot. In the case of a giving object, for instance, the class of giving objects will indicate that it has at least three slots: the donor, the recipient and the gift. There may be further slots indicated as necessary in the class, such as ones to give the time and location of the action. The time slot may be considered a formalization of the tense of the verb in a sentence. The idea of inheritance is used, with some slots being filled at class level, and some at instance level. Where a slot is filled at class level the idea is that this represents attributes which are common to all members of that class. Where it is filled at instance level, it indicates that the value of that attribute varies among members of that class. Slots may be filled with values or with pointers to other objects. This is best illustrated by an example.

Example of Frame (1):

In this example, we have a general class of birds, and all birds have attributes-flying, feathered and color. The attributes flying and feathered are Boolean values and are fixed to true at this level, which means that for all birds the attribute flying is true and the attribute feathered is true. The attribute color, though defined at this level is not filled, which means that though all birds have a color, their color varies. Two subclasses of birds, pet_canaries and ravens are defined. Both have the color slot filled in, pet_canaries with yellow, ravens with black. The class pet_canaries has an additional slot, owner, meaning that all pet canaries have an owner, though it is not filled at this level since it is obviously not the case that all pet canaries have the same owner.

We can therefore say that any instance of the class pet_canary has attributes color yellow, feathered true, flying true, and owner, the last of these varying among instances. Any instance of class raven has colour black, feathered true, flying true, but no attribute owner. The two instances of pet_canary shown, Tweety and Cheepy have owners John and Mary who are separate instances of the class person, for simplicity no attributes have been given for class person. The instance of pet_canary Cheepy has an attribute which is restricted to itself, vet (since not all pet canaries have their own vet), which is a link to another person instance, but in this case we have subclass of person, vet. The frame diagram for this is shown in figure 8.11.

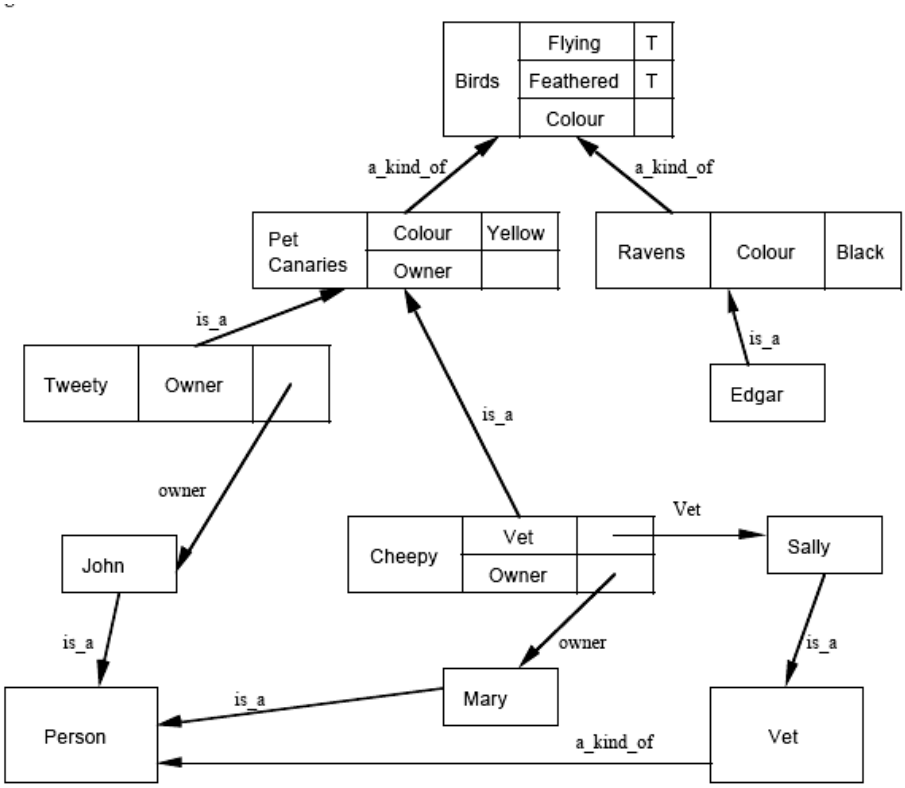

Figure 8.11: A frame system

We can define a general set of rules for making inferences on this sort of frame system. We can say that an object is an instance of a class if it is a member of that class, or if it is a member of a class which is a subclass of that class. A class is a subclass of another class if it is a kind of that class, or if it is a kind of some other class which is a subclass of that class. An object has a particular attribute if it has that attribute itself, or if it is an instance of a class that has that attribute.

In Prolog:

```
aninstance(Obj,Class) :– is_a(Obj,Class).
aninstance(Obj,Class) :– is_a(Obj,Class1), subclass(Class1,Class).
subclass(Class1,Class2) :– a_kind_of(Class1,Class2).
subclass(Class1,Class2) :– a_kind_of(Class1,Class3),
subclass(Class3,Class2).
```

We can then say that an object has a property with a particular value if the object itself has an attribute slot with that value, or it is an instance of a class which has an attribute slot with that value, in Prolog:

```
value(Obj,Property,Value) :– attribute(Obj,Property,Value).
value(Obj,Property,Value) :– aninstance(Obj,Class),
attribute(Class,Property,Value).
```

The diagram above is represented by the Prolog facts:

```
attribute(birds,flying,true).
attribute(birds,feathered,true).
attribute(pet_canaries,colour,yellow).
attribute(ravens,colour,black).
attribute(tweety,owner,john).
attribute(cheepy,owner,mary).
attribute(cheepy,vet,sally).
a_kind_of(pet_canaries,birds).
a_kind_of(ravens,birds).
a_kind_of(vet,person).
is_a(edgar,ravens).
is_a(tweety,pet_canaries).
is_a(cheepy,pet_canaries).
is_a(sally,vet).
is_a(john,person).
is_a(mary,person).
```

Note in particular how we have used reification leading to a representation of classes like birds, pet_canaries and so on by object constants, rather than by predicates as would be the case if we represented this situation in straightforward predicate logic. The term *superclass* may also be used, with X being a superclass of Y whenever Y is a subclass of X.

Using the Prolog representation, we can ask various queries about the situation represented by the frame system, for example if we made the Prolog query:

```
| ?- value(tweety,colour,V).
```

we would get the response:

```
V = yellow ?
```

while

| ?- value(john,feathered,V).
gives the response
no
indicating that feathered is not an attribute of John. Note that the no indicates that this is something which is not recorded in the system. If we wanted to actually store the information that persons are not feathered we would have to add:

attribute(person,feathered,true).
then the response would have been:
V = false ?

The only thing that has not been captured in this Prolog representation is the way that an attribute can be defined at one level and filled in lower down, like the color attribute of birds.

Example of Frame (2):

As discussed earlier, each frame represents either a class (a set) or an instance (an element of a class). To look at it again with a different example, consider a frame system as shown in figure 8.12 given below. In this example, the frames *Person, Adult-Male, Mohan-Bagan-Hockey-Player,* and *Mohan_Bagan_Hockey_Team* are all classes. The frames *Dhanraj-Pillai* and *Calcutta_Tigers* are instances. The *isa* (or *is_a*) relation is in fact the *subset* relation. The set of adult males is a subset of the set of people. The set of Mohan Bagan hockey players is a subset of the set of adult males and so forth. Dhanraj Pillai is an element of the set of fielders. Thus he is also an element of all of the supersets of fielders, including Mohan Bagan hockey players and people. Both the *isa* and *instance* relations have inverse attributes, which we call *subclasses* and *all-instances.*

Because a class represents a set, there are two kinds of attributes that can be associated with it. There are attributes about the set itself, and there are attributes that are to be inherited by each element of the set. We indicate the difference between these two by prefixing the latter with an asterisk (*). For example, consider the class *Mohan-Bagan-Hockey-Player*. Two properties of it are shown as a set: It is a subset of the set of adult males and it has cardinality 458. We have listed five properties that all Mohan bagan hockey players have (height, weight, position, team, and team-color).

By providing both kind of slots, we allow a class both to define a set of objects and to describe a prototypical object of the set.

Person

isa:	*Mammal*
Cardinality:	*5,000,0000,0000*
** handed:*	*Right*

Adult-Male

isa:	*Person*
Cardinality:	*3,000,0000,0000*
** height:*	*5-8*

Mohan-Bagan-Hockey-Player

isa:	*Adult-Male*
cardinality:	*458*
** height:*	*5-9*
** weight:*	*74*
** position:*	*Center*
** team:*	
** team-Color:*	

Fielders

isa:	*Hockey-Player*
cardinality:	*378*
** goal-average:*	*.25*

Danraj-Pillai

instance:	*Back*
height:	*8-0*
position:	*Centre*
goal-average:	*.98*
team:	*Mohan-Bagan*
uniform-colors:	*Black/Blue*

Mohan-Bagan-Team

isa:	*Team*
Cardinality:	*34*
** team-size:*	*11*
** coach:*	

Calcutta_Tigers

instance:	*Mohan-Bagan-Team*
team-size:	*11*
manager:	*KPS-Gill*
players:	*{Dhanraj-Pillai, Ashok-Jayaswal, …}*

Figure 8.12 A simple frame system

Note :

- The *isa* relation is in fact the subset relation.
- The *instance* relation is in fact *element of*.
- The *isa* attribute possesses a transitivity property.
- Both *isa* and *instance* have inverses which are called subclasses or all instances.
- There are attributes that are associated with the class or set such as cardinality and on the other hand there are attributes that are possessed by each member of the class or set.

Distinction between Sets and Instances

It is important that this distinction is clearly understood.

The team *Calcutta-Tigers,* which we have described as an instance of the class of *Mohan-Bagan-Hockey-Teams,* can be thought of as a set of players.

If *Calcutta_Tigers* were a *class* then

- its instances would be individual players
- it could not be a subclass of *Mohan-Bagan-Hockey-Team* which is not what we want to say.

We have to put it somewhere else in the *isa* hierarchy. For example, we can make it a subclass of *Mohan Bagan hockey players.* Then its elements, the players, are also elements of *Mohan-Bagan-Hockey-Player*, *Adult-Male and Person.* That is acceptable.

BUT there is a problem here:

- A class is a set and its elements have properties.
- We wish to use inheritance to bestow values on its members.
- But there are properties that the set or class itself has, such as the manager of a team.

Hence we need to view a class as two things simultaneously: a subset (*isa*) of a larger class that also contains its elements and an instance (*instance*) of a class of sets, from which it inherits its set-level properties. We seem to have a *Solution*: **MetaClasses**

A metaclass is a special class whose elements are themselves classes. It is useful to distinguish between regular classes, whose elements are individual entities, and metaclass, which are special classes whose elements are themselves classes. A class is now an element of (*instance*) some class (or classes) as well as a subclass (*isa*) of one or more classes. A class inherits properties from the class of which it is an instance, just as any instance does. In addition, a class inheritable properties down from its superclasses to its instances.

Figure 8.13, given below, shows how we could represent teams as classes using this distinction. The most basic metaclass is the class *Class.* It represents the set of all classes. All classes are instance of it, either directly or through one of its subclasses. In the example, *Team* is a subclass (subset) of *Class and Mohan-Bagan-Hocky-Team* is a subclass of *Team.* The class *Class* introduces the attribute *cardinality,* which is to be inherited by all instances of *Class* (includeing itself). This makes sense since all instances of *Class* are sets and all sets have a cardinality.

Team represents a subset of the set of all sets, namely those whose elements are sets of players on a team. It inherits the property of having a cardinality from *Class. Team* introduces the attribute *team-size*, which all its elements possess. Notice that *team-size* is like cardinality in that it measures the size of a set. But it applies to something different*; cardinality* applies to sets of sets and is inherited by all elements of *Class.* The slot *team-size* applies to the elements of those sets that happen to be teams. Those elements are sets of individuals.

Mohan-Bagan-Hockey-Team is also an instance of *Class*, since it is a set. It inherits the property of having a cardinality from the set of which it is an instance, namely *Class*. But it is a subset of *Team*. All of its instances will have the property of having a *team-size* since they are also instances of the superclass *Team.* We have added at this level the additional fact that the default team size is 24, so all instances of *Mohan-Bagan-Hockey-Team* will inherit that as well. In addition, we have added the inheritable slot *manager*.

Calcutta_Tigers is an instance of *Mohan-Bagan-Hockey-Team.* It is not an instance of *Class* because its elements are individuals, not sets. *Calcutta_Tigers* is a subclass of *Mohan-Bagan-Hockey-Player* since all of its elements are also elements of that set. Since it is an instance of *Mohan-Bagan-Hockey-Team*, it inherits the properties *team-size* and *manager,* as well as their default values. It specifies a new attribute *uniform-color*, which is to be inherited by all of its instances (who will be individual players).

Finally, *Dhanraj-Pillai* is an instance of *Calcutta-Tigers*. That makes him also, by transitivity up *isa* links, an instance of *Mohan-Bagan-Hockey-Player*. But in our earlier example we also used the class *Fielder,* to which we attached the fact that fielders have above-avarage betting averages. To allow that here, we simply make Dhanraj an instance of *Fielder* as well. He will thus inherit properties from both *Calcutta-Tigers* and from *Fielder,* as well as from the classes above these. We need to guarantee that when multiple inheritance occurs, as it does here, then it works correctly.

Now consider our *Mohan-Bagan-Hockey-Team* as:

Class

instance:	*Class*
isa:	*Class*
**cardinality:*	

Team

instance:	*Class*
isa:	*Class*
cardinality:	{the number of teams that exist}
** team size:*	{each team has a size}

Mohan-Bagan-Hockey-Team

instance:	*Class*
isa:	*Team*
cardinality:	28 {the number of hockey teams that exists}
** team size:*	24 {default 24 players on a team}
** manager:*	

Calcutta_Tigers

instance:	*Mohan-Bagan-Hockey-Team*
isa:	*Mohan-Bagan-Hockey-Player*

team-size: *24*
manager: *KPS Gill*
** uniform color:* *Blue*

Danraj-Pillai
instance: *Calcutta-Tigers*
instance : *Fielder*
uniform color: *Blue*
goal-average: *.98*

The basic metaclass is *Class*, and this allows us to

- define classes which are instances of other classes, and (thus)
- inherit properties from this class.

Inheritance of default values occurs when one element or class is an instance of a class.

Self Assessment Questions

4. _____ Semantic Networks is an extension to Semantic nets that overcome a few problems or extend their expression of knowledge.
5. What is the for a collection of attributes or slots and associated values that describe some real world entity.
6. An object has a particular attribute if it has that attribute itself, or if it is an instance of a class that has that attribute.(State True or False)
7. The _____ indicates the number of slots that an object has, and the name of each slot.
8. A _____ is a special class whose elements are themselves classes.

8.5 Summary

In this unit we discussed Semantic Networks, Partitioned Semantic Networks and Frames.

- Semantic networks are an alternative to predicate logic as a form of knowledge representation. This process of turning a predicate into an object in a knowledge representation system is known as *reification*.
- Partitioned Semantic Networks is an extension to Semantic nets that overcome a few problems or extend their expression of knowledge.

- Frames can be regarded as an extension to Semantic nets. Frames are good for applications in which the structure of the data and the available information are relatively fixed.

8.6 Terminal Questions

1. What do you mean by semantic networks? Explain with an example.
2. Explain Partitioned semantic networks with an example.
3. What do you mean by frame? What are the advantages and disadvantages of frames?
4. List the features of frames.

8.7 Answers

Self Assessment Questions

1. Semantic
2. Inheritance
3. reification
4. Partitioned
5. frame
6. True
7. class
8. metaclass

Terminal Questions

1. Semantic networks are an alternative to predicate logic as a form of knowledge representation. (Refer section 8.2 for detail)
2. Partitioned Semantic Networks is an extension to Semantic nets that overcome a few problems or extend their expression of knowledge. (Refer section 8.3 for detail)
3. A frame *is* a collection of attributes or slots and associated values that describe some real world entity. (Refer section 8.4 for details regarding its meaning, advantages and disadvantages)
4. A frame has *slots*. A slot is like a property, but can contain more kinds of information (sometimes called *facets* of the slot):
   - The value of the slot; default value in case no value is present.
   - A procedure that can be run to compute the value (an *if-needed procedure*). (Refer section 8.4 for detail)