

## 2. Semantic Networks

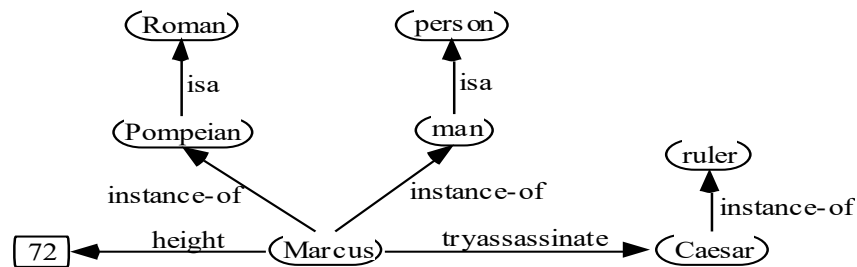
A semantic network represents knowledge in the form of a graph in which the nodes represent objects, situations, or events, and the arcs represent the relationships between them. Labels on the arcs specify what type of relationship exists

### Example 1

Represent the following sentences into a semantic network

1. Marcus was a man
2. Marcus was a Pompeian
3. All Pompeians were Romans
4. Caesar was a ruler
8. Marcus tried to assassinate Caesar
9. All men are persons
10. The height of Marcus was 72

### Solution



### Types of Nodes

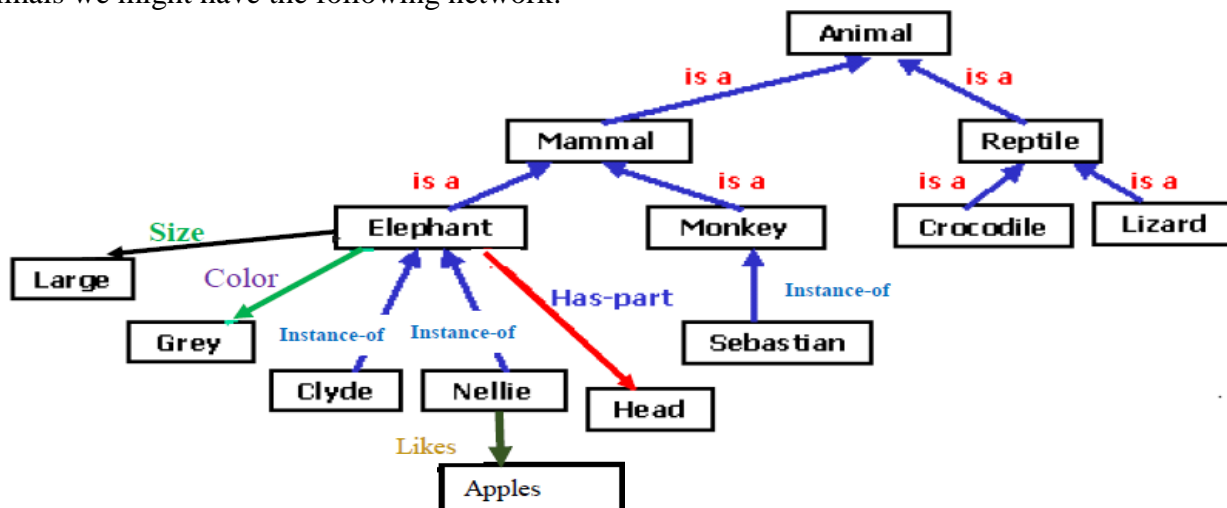
- **Generic node:** This refers to a general node. e.g. Roman, Pompeian, etc
- **Individual Nodes / Instance Nodes:** These are nodes that explicitly state that they are specific instances of a generic node

### Relationships

The most important relations between concepts are:

- Subclass – between classes and subclasses i.e. the relationship "is-a" asserts that a class, ("man"), is a subclass of another class ("person") (denotes inheritance).
- Instance – between a particular object (Marcus, Caesar) and its parent class i.e. the relationship "instance-of" asserts that an individual is an instance (element) of a class.

However, any other relations are allowed, such as *has-part*, *colour*, etc. So, to represent some knowledge about animals we might have the following network:



This network represents the fact that mammals and reptiles are animals, an elephant is a large grey mammal, Clyde and Nellie are both elephants, and that Nellie likes apples. The subclass relations define a *class hierarchy*.

The subclass and instance relations may be used to derive new information which is not explicitly represented. We should be able to conclude that Clyde and Nellie both have a head, and are large and grey. They *inherit* information from their parent classes. Semantic networks normally allow efficient inheritance-based inferences using special purpose algorithms.

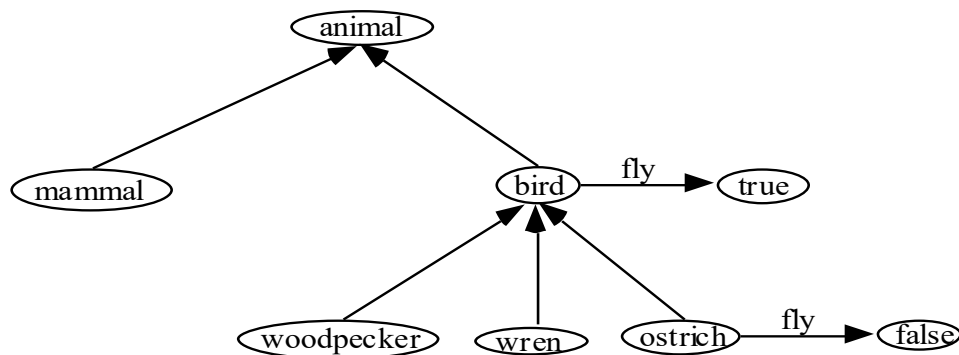
Notice that Semantic nets are fine at representing relationships between two objects - but what if we want to represent a relation between three or more objects? Say we want to represent the fact that "John gives Mary the book" This might be represented in logic as *gives(John, Mary, book2)* where book2 represents the particular book we are talking about. However, in semantic networks we have to view the fact as representing a set of binary relationships between a "giving" event and some objects.

### Default inheritance

There are some domains of knowledge in which exceptions to general rules exist. E.g. it is usually useful to assume that all birds can fly. However certain birds such as the ostrich and the kiwi cannot fly. In such a case, it is reasonable to use a representation scheme in which properties associated with concepts in a hierarchy are assumed to be true of all subclasses, unless specifically overridden by a denial or modification associated with the subclass.

### Example

In the figure below the fact that a woodpecker flies is inherited from bird. On the other hand, the ostrich does not inherit this property from bird because it is explicitly represented that the ostrich does not fly. This overrides the default which is further up in the tree.



To summarize, nets allow us to simply represent knowledge about an object that can be expressed as binary relations. Subclass and instance relations allow us to use *inheritance* to infer new facts/relations from the explicitly represented one.

### Exercises

1. Represent the following sentences into a semantic network:

Palco is a calico.

Herb is a tuna.

Charlie is a tuna.

All tunas are fishes.

All calicos are cats.

All cats like to eat all kinds of fishes.

2. Represent the following sentences into a semantic network:

Circus elephants are elephants

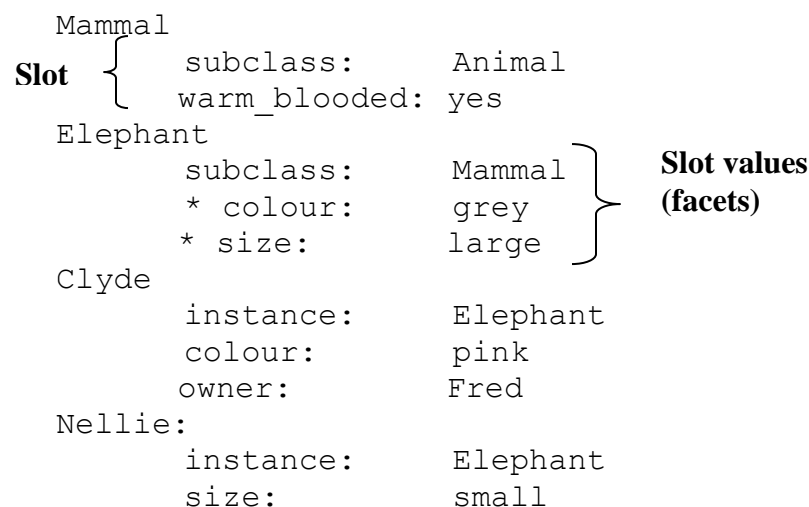
Elephants have head  
 Elephants have trunks  
 Heads have mouths  
 Elephants are animals  
 Animals have hearts  
 Circus elephants are performers  
 Performers have costumes  
 Costumes are cloths  
 Horatio is a circus elephant

### 3. Frames

Frames are a variant of nets that are one of the most popular ways of representing non-procedural knowledge in an expert system. In a frame, all the information relevant to a particular concept is stored in a single complex entity, called a frame. Superficially, frames look pretty much like record data structures. However frames, at the very least, support inheritance. They are often used to capture knowledge about *typical* objects or events, such as a typical bird, or a typical restaurant meal.

#### Example frame representation

We could represent some knowledge about elephants in frames as follows:



A particular frame (such as Elephant) has a number of *attributes* or *slots* such as colour and size where these slots may be filled with particular values, such as grey. "\*" is used to indicate those attributes that are only true of a *typical* member of the class, and not necessarily every member.

In the above frame system, it is possible to infer that Nellie was small, grey and warm blooded. Clyde is large, pink and warm blooded and owned by Fred. Objects and classes inherit the properties of their parent classes UNLESS they have an individual property value that conflicts with the inherited one.

The following diagram shows relationship between semantic nets and frames

Semantic Networks	Frames
Nodes	Objects
Links	Slots
Node at the other end of the link	Slot value

### Example

Animals eat, breath and have skin. Birds are special kind of animal which have wings, feathers and normally can fly. Crows are black and can fly. Penguins are black and white and ostriches are brown. Penguins and ostriches are birds but they cannot fly. Tom, a human being, owns a pet crow called 'black-head'.

Animals:

```
can_eat: true
can_breath: true
have_skin: true
```

Birds:

```
sub_class: Animals
have_wings: true
have_feathers: true
* can_fly: true
```

Crows:

```
sub_class: Birds
colour: black
```

Penguins:

```
sub_class: Birds
colour: black_and_white
can_fly: false
```

Ostriches:

```
sub_class: Birds
colour: brown
can_fly: false
```

tom:

```
instance: human_being
owns: black_head
```

black\_head:

```
instance: crow
```

## 4. Production systems (Rule-Based Systems)

Instead of representing knowledge in a relatively declarative, static way (as a bunch of things that are true), rule-based system represent knowledge in terms of a bunch of rules that tell you what you should do or what you could conclude in different situations. A rule-based system consists of a bunch of IF-THEN *rules*, a bunch of *facts*, and some *interpreter* controlling the application of the rules, given the facts.

There are two broad kinds of rule system: *forward chaining* systems, and *backward chaining* systems. In a forward chaining system you start with the initial facts, and keep using the rules to draw new conclusions (or take certain actions) given those facts. In a backward chaining system you start with some hypothesis (or goal) you are trying to prove, and keep looking for rules that would allow you to conclude that hypothesis, perhaps setting new subgoals to prove as you go. Forward chaining systems are primarily data-driven, while backward chaining systems are goal-driven.

### General format

*If the condition C is satisfied, then the action A is appropriate*

### Example

- i) *If it is raining, then you should open the umbrella.*
- ii) *IF the 'traffic light' is green*  
*THEN the action is go*  
*IF the 'traffic light' is red*  
*THEN the action is stop*

The action will be carried out if the condition is true and the condition and action can both be composed of several parts.

#### A Typical Rule(A rule from the MYCIN expert system)

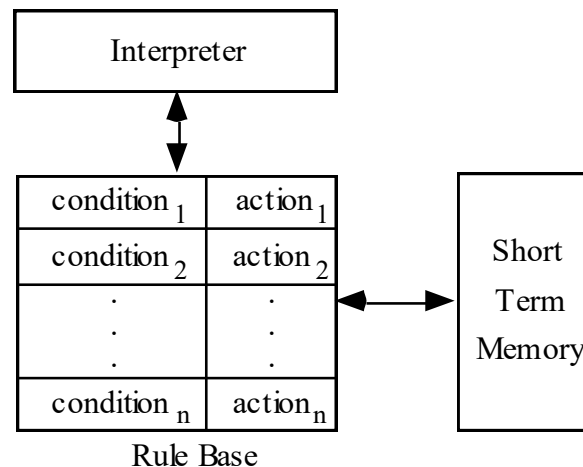
*IF the strain of the organism is gramneg  
AND the morphology of the organism is rod  
AND the aerobicity of the organism is aerobic  
THEN there is strong suggestive evidence that the class of the  
organism is enterobacteriaceae*

#### **Architecture of a Production System**

A production system consists of three parts:

- A short-term memory of facts.
- A rule base composed of a set of production rules of the form "antecedent --> consequent". The antecedents comprise conditions that characterize the short-term memory and the consequents comprise actions that manipulate the short-term memory. If the antecedent of a production rule is satisfied by the short-term memory then the production may be executed (fired). The consequent of the production rule will change the content of the short-term memory so that other rules will have their conditions satisfied.
- An interpreter that represents a mechanism to examine the short-term memory and to determine which rules to fire.

The following diagram represents the architecture of a production system.



#### **Types of production rules:**

- Situation-action rules:* - The antecedent is a logical combination of assertions about the data in the short-term memory and the consequent is a collection of actions that change the memory.
- Inference rules:* - Both the antecedents and the consequents are assertions about the data in the short-term memory.

The production systems based on situation-action rules can be run only in forward chaining and are therefore called antecedent-driven systems.

The production systems based on inference rules may be run in either forward chaining or backward chaining, since both sides of a production consist of the same constructs.

#### **Execution in a Production System**

The execution of a production system can be defined as a series of recognize-act cycles. It involves the following:

- Match:** - determines the rules that can be fired. The set of rules that can be fired is called the *conflict set*.

- ii) **Conflict resolution:-** It involves selecting one rule from the conflict set
- iii) **Apply the rule:-** A rule is executed (fired). Since the effect of executing the actions may change the short-term memory, different rules may fire on successive cycles.

### Conflict Resolution Strategies:

Several strategies may be used to choose the rule to fire from the conflict set:

1. **Rule order:** -The first encountered rule that matches the short-term memory.
2. **Priority:** - The highest priority rule (assuming that each rule has an associated priority).
3. **Specificity:** - The most specific rule, that is, the one with the most detailed condition part that matches the current short term memory.
4. **Recent:** - The rule that refers to the element most recently added to the short-term memory.
5. **New rule:** - Occurs in a rule-binding instantiation for a rule that has not occurred previously;

### Note

A rule that would not modify the content of the short-term memory should not be fired (e.g. a rule that would infer a fact that is already in the short term memory).

### Exercises

Consider the following a production system characterized by

- Initial short term memory: C5, C1, C3
- Production rules:
  - C1 & C2 --> C4
  - C3 --> C2
  - C1 & C3 --> C6
  - C4 --> C6
  - C5 --> C1

Show a possible sequence of two recognize-act cycles. Which will be the new content of the short-term memory after these two cycles?

### Solution

1. *Match:* determines the rules that can be fired

<ul style="list-style-type: none"> <li>C3 --&gt; C2</li> <li>C1 &amp; C3 --&gt; C6</li> <li>C5 --&gt; C1</li> </ul>	}	conflict set.
---	---	---------------

2. *Conflict resolution:* select one rule from the conflict set (use rule order)

C3 --> C2

3. *Apply the rule:*

- new short term memory: C5, C1, C3, C2

1. *Match:* determines the rules that can be fired

- C1 & C2 --> C4
- C3 --> C2
- C1 & C3 --> C6
- C5 --> C1

2. *Conflict resolution:* select one rule from the conflict set

C1 & C2 --> C4

3. *Apply the rule:*

- new short term memory: C5, C1, C3, C2, C4

Other Knowledge Representation Schemes are: scripts, lists, decision tables, decision trees and object attribute value (OAV) triplet

### **Hybrid Representation**

- No single representation will be suitable for representing every problem domain i.e. every representation schema has its own advantages and disadvantages. Therefore a hybrid representation schema may be better than any single schema.
- A.I. programming tools containing rules, frames and logic are now available