$ab_{no\_swimming\_near\_city} \leftarrow in\_BC \wedge \sim ab_{BC\_beaches}$.
Given only the preceding rules, an agent infers *away_from_beach*. If it is then told *on_beach*, it can no longer infer *away_from_beach*, but it can now infer *beach_access* and *swim_at_beach*. If it is also told *enclosed_bay* and *big_city*, it can no longer infer *swim_at_beach*. However, if it is then told *in_BC*, it can then infer *swim_at_beach*.
By having defaults of what is normal, a user can interact with the system by telling it what is abnormal, which allows for economy in communication. The user does not have to state the obvious.
One way to think about non-monotonic reasoning is in terms of **arguments**. The rules can be used as components of arguments, in which the negated abnormality gives a way to undermine arguments. Note that, in the language presented, only positive arguments exist that can be undermined. In more general theories, there can be positive and negative arguments that attack each other.

Implementation Issues

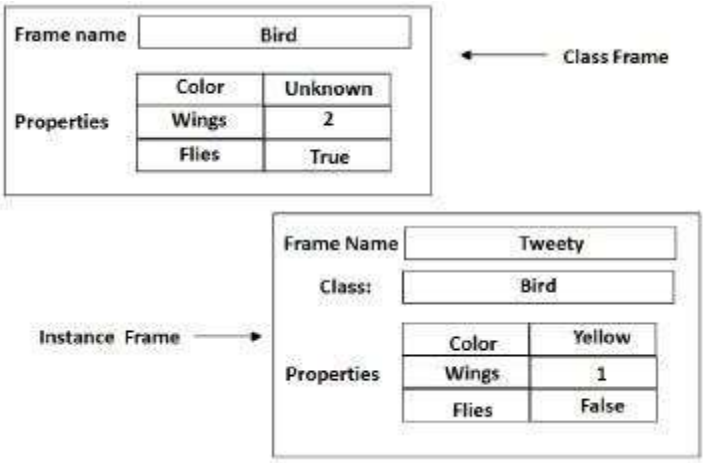
Weak Slot and Filler Structures

Evolution Frames

- As seen in the previous example, there are certain problems which are difficult to solve with Semantic Nets.
- Although there is no clear distinction between a semantic net and frame system, more structured the system is, more likely it is to be termed as a frame system.
- A frame is a collection of attributes (called slots) and associated values that describe some entities in the world. Sometimes a frame describes an entity in some absolute sense;
- Sometimes it represents the entity from a particular point of view only.
- A single frame taken alone is rarely useful; we build frame systems out of collections of frames that connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame.

Frames as Sets and Instances

- The set theory is a good basis for understanding frame systems.
- Each frame represents either a class (a set) or an instance (an element of class)
- Both ***isa*** and ***instance*** relations have inverse attributes, which we call subclasses & all instances.
- As a class represents a set, there are 2 kinds of attributes that can be associated with it.
    1. Its own attributes &
    2. Attributes that are to be inherited by each element of the set.



Frames as Sets and Instances

- Sometimes, the difference between a set and an individual instance may not be clear.
- Example: Team India is an instance of the class of Cricket Teams and can also think of as the set of players.
- Now the problem is if we present Team India as a subclass of Cricket teams, then Indian players automatically become part of all the teams, which is not true.
- So, we can make Team India a subclass of class called Cricket Players.
- To do this we need to differentiate between regular classes and meta-classes.
- Regular Classes are those whose elements are individual entities whereas Meta-classes are those special classes whose elements are themselves, classes.
- The most basic meta-class is the class *CLASS*.
- It represents the set of all classes.
- All classes are instances of it, either directly or through one of its subclasses.
- The class *CLASS* introduces the attribute cardinality, which is to inherited by all instances of CLASS. Cardinality stands for the number.

Other ways of Relating Classes to Each Other

- We have discussed that a class1 can be a subset of class2.
- If Class2 is a meta-class then Class1 can be an instance of Class2.
- Another way is the ***mutually-disjoint-with*** relationship, which relates a class to one or more other classes that guaranteed to have no elements in common with it.
- Another one is, ***is-covered-by*** which relates a class to a set of subclasses, the union of which is equal to it.
- If a class is-covered-by a set S of mutually disjoint classes, then S called a partition of the class.

Slots as Full-Fledged Objects (Frames)

Till now we have used attributes as slots, but now we will represent attributes explicitly and describe their properties.

Some of the properties we would like to be able to represent and use in reasoning include,

- The class to which the attribute can attach.
- Constraints on either the type or the value of the attribute.
- A default value for the attribute. Rules for inheriting values for the attribute.
- To be able to represent these attributes of attributes, we need to describe attributes (slots) as frames.

- These frames will organize into an ***isa*** hierarchy, just as any other frames, and that hierarchy can then used to support inheritance of values for attributes of slots.
- Now let us formalize what is a slot. A slot here is a relation.
- It maps from elements of its domain (the classes for which it makes sense) to elements of its range (its possible values).
- A relation is a set of ordered pairs.
- Thus it makes sense to say that relation R1 is a subset of another relation R2.
- In that case, R1 is a specialization of R2. Since a slot is a set, the set of all slots, which we will call SLOT, is a meta-class.
- Its instances are slots, which may have sub-slots.

Frame Example

In this example, the frames Person, Adult-Male, ML-Baseball-Player (corresponding to major league baseball players), Pitcher, and ML-Baseball-Team (for major league baseball team) are all classes.

Person	
isa :	*Mammal*
cardinality :	6,000,000,000
** handed :*	*Right*
Adult-Male	
isa :	*Person*
cardinality :	2,000,000,000
** height :*	5-10
ML-Baseball-Player	
isa :	*Adult-Male*
cardinality :	624
**height :*	6-1
** bats :*	equal to handed
** batting-average :*	.252
** team :*	
** uniform-color :*	
Fielder	
isa :	*ML-Baseball-Player*
cardinality :	376
**batting-average :*	.262
Pee-Wee-Reese	
instance :	*Fielder*
height :	5-10
bats :	*Right*
batting-average :	.309
team :	*Brooklyn-Dodgers*
uniform-color :	*Blue*
ML-Baseball-Team	
isa:	*Team*
cardinality :	26
** team-size :*	24
** manager :*	
Brooklyn-Dodgers	
instance :	*ML-Baseball-Team*
team-size :	24
manager :	*Leo-Durocher*
players :	*{Pee-Wee-Reese,...}*

- The frames Pee-Wee-Reese and Brooklyn-Dodgers are instances.
- The ***isa*** relation that we have been using without a precise definition is, in fact, the subset relation. The set of adult males is a subset of the set of people.
- The set of major league baseball players subset of the set of adult males, and so forth.
- Our instance relation corresponds to the relation element-of Pee Wee Reese is an element of the set of fielders.
- Thus he is also an element of all of the supersets of fielders, including major league baseball players and people. The transitivity of *isa* follows directly from the transitivity of the subset relation.

- 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 associate with it.
- Some attributes are about the set itself, and some attributes are to 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 ML-Baseball-Player, we have shown only two properties of it as a set: It a subset of the set of adult males. And it has cardinality 624.
- We have listed five properties that all major league baseball players have (height, bats, batting average, team, and uniform-color), and we have specified default values for the first three of them.
- By providing both kinds of slots, we allow both classes to define a set of objects and to describe a prototypical object of the set.
- Frames are useful for representing objects that are typical of stereotypical situations.
- The situation like the structure of complex physical objects, visual scenes, etc.
- A commonsense knowledge can represent using default values if no other value exists. Commonsense is generally used in the absence of specific knowledge.

Semantic Nets

- Inheritance property can represent using **isa** and **instance**
- Monotonic Inheritance can perform substantially more efficiently with such structures than with pure logic, and non-monotonic inheritance is also easily supported.
- The reason that makes Inheritance easy is that the knowledge in slot and filler systems is structured as a set of entities and their attributes.

These structures turn out to be useful as,

- It indexes assertions by the entities they describe. As a result, retrieving the value for an attribute of an entity is fast.
- Moreover, It makes easy to describe properties of relations. To do this in a purely logical system requires higher-order mechanisms.
- It is a form of object-oriented programming and has the advantages that such systems normally include modularity and ease of viewing by people.

Here we would describe two views of this kind of structure – Semantic Nets & Frames.

Semantic Nets

- There are different approaches to knowledge representation include semantic net, frames, and script.
- The semantic net describes both objects and events.
- In a semantic net, information represented as a set of nodes connected to each other by a set of labeled arcs, which represents relationships among the nodes.
- It is a directed graph consisting of vertices which represent concepts and edges which represent semantic relations between the concepts.
- It is also known as associative net due to the association of one node with other.
- The main idea is that the meaning of the concept comes from the ways in which it connected to other concepts.
- We can use inheritance to derive additional relations.

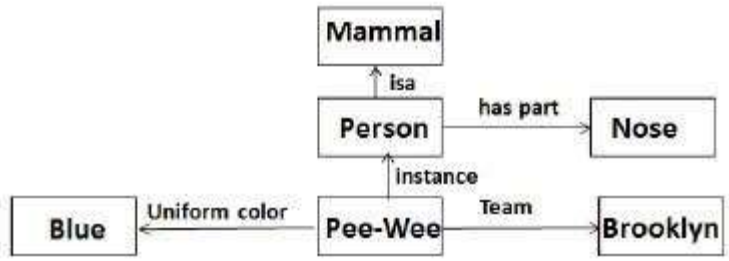


Figure: A Semantic Network

Intersection Search Semantic Nets

- We try to find relationships among objects by spreading activation out from each of two nodes. And seeing where the activation meets.
- Using this we can answer the questions like, what is the relation between India and Blue.
- It takes advantage of the entity-based organization of knowledge that slot and filler representation provides.

Representing Non-binary Predicates Semantic Nets

- Simple binary predicates like isa(Person, Mammal) can represent easily by semantic nets but other non-binary predicates can also represent by using general-purpose predicates such as ***isa*** and ***instance***.
- Three or even more place predicates can also convert to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe a relationship to this new object.

Conceptual Dependency

Introduction to Strong Slot and Filler Structures

- The main problem with semantic networks and frames is that they lack formality; there is no specific guideline on how to use the representations.
- In frame when things change, we need to modify all frames that are relevant – this can be time-consuming.
- Strong slot and filler structures typically represent links between objects according to more rigid rules, specific notions of what types of object and relations between them are provided and represent knowledge about common situations.
- Moreover, We have types of strong slot and filler structures:
    1. Conceptual Dependency (CD)
    2. Scripts
    3. Cyc

Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

- To help in the drawing of the inference from sentences.
- To be independent of the words used in the original input.
- That is to say: For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.

Moreover, It has used by many programs that portend to understand English (MARGIE, SAM, PAM).

Conceptual Dependency (CD) provides:

- A structure into which nodes representing information can be placed.
- Also, A specific set of primitives.
- A given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

- The agent and the objects represented.
- Moreover, The actions are built up from a set of primitive acts which can modify by tense.

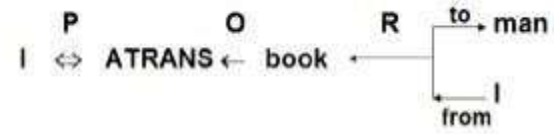
CD is based on events and actions. Every event (if applicable) has:

- an ACTOR o an ACTION performed by the Actor
- Also, an OBJECT that the action performs on
- A DIRECTION in which that action is oriented

These are represented as slots and fillers. In English sentences, many of these attributes left out.

A Simple Conceptual Dependency Representation

For the sentences, "I have a book to the man" CD representation is as follows:



Where the symbols have the following meaning.

- Arrows indicate directions of dependency.
- Moreover, The double arrow indicates the two-way link between actor and action.
- O — for the object case relation
- R – for the recipient case relation
- P – for past tense
- D – destination

Primitive Acts of Conceptual Dependency Theory

ATRANS

- Transfer of an abstract relationship (i.e. give)

PTRANS

- Transfer of the physical location of an object (e.g., go)

PROPEL

- Also, Application of physical force to an object (e.g. push)

MOVE

- Moreover, Movement of a body part by its owner (e.g. kick)

GRASP

- Grasping of an object by an action (e.g. throw)

INGEST

- Ingesting of an object by an animal (e.g. eat)

EXPEL

- Expulsion of something from the body of an animal (e.g. cry)

MTRANS

- Transfer of mental information (e.g. tell)

MBUILD

- Building new information out of old (e.g decide)

SPEAK

- Producing of sounds (e.g. say)

ATTEND

- Focusing of a sense organ toward a stimulus (e.g. listen)

There are four conceptual categories. These are,

ACT

- Actions {one of the CD primitives}

PP

- Also, Objects {picture producers}

AA

- Modifiers of actions {action aiders}

PA

- Modifiers of PP's {picture aiders}

Advantages of Conceptual Dependency

- Using these primitives involves fewer inference rules.
- So, Many inference rules already represented in CD structure.
- Moreover, The holes in the initial structure help to focus on the points still to established.

Disadvantages of Conceptual Dependency

- Knowledge must decompose into fairly low-level primitives.
- Impossible or difficult to find the correct set of primitives.
- Also, A lot of inference may still require.
- Representations can be complex even for relatively simple actions.
- Consider: Dave bet Frank five pounds that Wales would win the Rugby World Cup.
- Moreover, Complex representations require a lot of storage.

Scripts

Scripts Strong Slot

- A script is a structure that prescribes a set of circumstances which could be expected to follow on from one another.
- It is similar to a thought sequence or a chain of situations which could be anticipated.
- It could be considered to consist of a number of slots or frames but with more specialized roles.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- Causal relationships between events exist.
- Entry conditions exist which allow an event to take place
- Prerequisites exist for events taking place. E.g. when a student progresses through a degree scheme or when a purchaser buys a house.

Script Components

Each script contains the following main components.

- Entry Conditions: Must be satisfied before events in the script can occur.
- Results: Conditions that will be true after events in script occur.
- Props: Slots representing objects involved in the events.
- Roles: Persons involved in the events.
- Track: the Specific variation on the more general pattern in the script. Different tracks may share many components of the same script but not all.

- Scenes: The sequence of events that occur. Events represented in conceptual dependency form.

Advantages and Disadvantages of Script

Advantages

- Capable of predicting implicit events
- Single coherent interpretation may be build up from a collection of observations.

Disadvantage

- More specific (inflexible) and less general than frames.
- Not suitable to represent all kinds of knowledge.

To deal with inflexibility, smaller modules called memory organization packets (MOP) can combine in a way that appropriates for the situation.

Script Example

Script : Play in theater	Various Scenes
Track: Play in Theater **Props:** • Tickets • Seat • Play **Roles:** • Person (who wants to see a play) – P • Ticket distributor – TD • Ticket checker – TC **Entry Conditions:** • P wants to see a play • P has a money **Results:** • P saw a play • P has less money • P is happy (optional if he liked the play)	***Scene 1: Going to theater*** • P PTRANS P into theater • P ATTEND eyes to ticket counter ***Scene 2: Buying ticket*** • P PTRANS P to ticket counter • P MTRANS (need a ticket) to TD • TD ATRANS ticket to P ***Scene 3: Going inside hall of theater and sitting on a seat*** • P PTRANS P into Hall of theater • TC ATTEND eyes on ticket POSS_by P • TC MTRANS (showed seat) to P • P PTRANS P to seat • P MOVES P to sitting position ***Scene 4: Watching a play*** • P ATTEND eyes on play • P MBUILD (good moments) from play ***Scene5: Exiting*** • P PTRANS P out of Hall and theater

- It must activate based on its significance.
- If the topic important, then the script should open.
- If a topic just mentioned, then a pointer to that script could hold.
- For example, given "John enjoyed the play in theater", a script "Play in Theater" suggested above invoke.
- All implicit questions can answer correctly.

Here the significance of this script is high.

- Did John go to the theater?
- Also, Did he buy the ticket?
- Did he have money?

If we have a sentence like "John went to the theater to pick his daughter", then invoking this script will lead to many wrong answers.

- Here significance of the script theater is less.