Chapter 7
Networks-Based Representation

Abstract Network-based method is another approach for knowledge representation and reasoning. They have particularly the advantage that, using the network one can navigate through the knowledge represented, and can perform the inferences. This chapter presents the semantic networks, conceptual graphs, frames, and conceptual dependencies, as well as their syntax and semantics. The $\mathscr{DL}$ (description logic)—a modified predicate logic for real-world applications is treated in detail, with examples of its language—the concept language for inferencing. Conceptual dependency (CD) is a language-independent representation and reasoning framework, such that whatever may be the natural language used, as long as its meaning is the same, the CD will be the same. The script language for representation and reasoning along with its syntax, semantics, and reasoning for CD is presented, followed with chapter summary, and an exhaustive list of exercises.

Keywords Network-based representation · Semantic networks · Conceptual graph · Frames · Description Logic (DL) · Conceptual dependencies · Scripts

7.1 Introduction

Semantic Networks were developed with the goal of characterizing the knowledge and the reasoning of a system by means of network-shaped cognitive structures. The similar goals were later achieved through frame-based systems, which depend on the notion of a "frame" as a prototype, and have the capability to express the relationship between the frames. The frames and semantic networks are quite different from each other, but both have cognitive features, and have the capability that allows navigation in the structures. Due to this, both of them can be classified as network-based structures, where the network is used for representing individuals and the relationship between them.

Due to their human-oriented origins, the network-based systems are more appealing and effective from the practical point of view than the logical systems, that are based on predicate logic and its variants. However, these network-based systems were not accepted as a complete solution, due to their lack of semantic properties.

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_7

Due to that, every system behaved differently from the others, despite their almost identical-looking components, as well as identical-looking relationships. Hence, the need was felt to represent semantic characteristics in these structures. The semantic characteristics in network-based systems could be achieved by introducing the notion of hierarchical structures. Using hierarchical structures in semantics networks and frames, one could gain both in terms of ease of representation, and also in terms of efficiency of reasoning.

Learning Outcomes of this Chapter:

1. Identify all of the data, information, and knowledge elements and related organizations, for a computational science application. [Assessment]
2. Describe how to represent data and information for processing. [Familiarity]
3. Compare and contrast the most common models used for structured knowledge representation, highlighting their strengths and weaknesses. [Assessment]
4. Identify the components of non-monotonic reasoning and its usefulness as a representational mechanism for belief systems. [Familiarity]
5. Compare and contrast the basic techniques for qualitative representation. [Assessment]

7.2 Semantic Networks

A network-based representation provides means of structuring and exhibiting the structure of knowledge. In a network, the pieces of knowledge are clustered together into coherent semantic groups. It provides a natural way of mapping knowledge between the natural language and these networks. In addition, the network representation provides a pictorial representation of knowledge objects, their attributes, and relationship between them [2].

The basic difference between ontologies we studied earlier and the semantic networks is that, ontologies are hierarchies, which may have multiple inheritances, providing knowledge organization of world. Whereas, semantic networks are not necessarily be hierarchy and they follow the lattice structure for knowledge representation.

There is a need of built-in feature of natural language understanding in knowledge representation, so that it becomes possible to carry out the inference through this representation. However, when we make use of general theorem proving framework, like resolution, these actually desired inferences are lost due to a wide range of inferences carried out using resolution-refutation method (see Example 3.14, p. 74).

The Semantic networks not only represent information but facilitate the retrieval of relevant facts. For instance, all the facts about an object “Rajan” are stored with a pointer directly to one node representing Rajan. Another advantage of semantic networks is about the inheritance of properties. If a semantic network represents the knowledge: “All canaries are of yellow color”, and “Tweety is canary”, the network would be able to infer that “Tweety is of yellow color.” This inference is performed

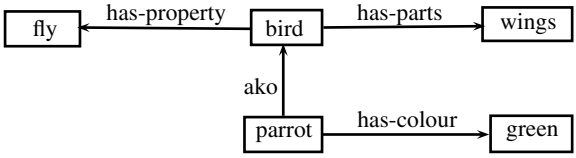

Fig. 7.1 A semantic network

through *network mathcer* or *retriever*. The most advanced system of inference has *Inference Engine*, which can perform specialized inferences tailored to treat certain functions, predicates, and constant symbols differently than others. This is achieved by building into the inference engine certain true sentences, which involve these symbols, and control is provided to handle these sentences. The inference engine is able to recognize the special conditions, on which it makes use of specialized machinery. It becomes possible by coupling the specialized knowledge to the form of situations that it can deal with.

The semantic networks also called the *Associative Networks*, model the semantics and words of the English language. In a system developed by their inventor Quillian, evidenced that meanings were found between the words by the path connecting them. These models carry an intuitive appeal: the related information is always clustered and bound together through relational links, and the knowledge required to perform a certain task is typically contained in a narrow domain or in the vicinity of the concerned task. This type or knowledge organization, in some way, resembles the way knowledge is stored and retrieved in human brain [11].

Semantic networks includes a lattice of concept types [6]. Originally they included a different correlated nets, which were based on 56 relations, like: subtypes, instances, case-relations, part-whole, kinship relations, and various types of attributes. A simple example of semantic network is shown in Fig. 7.1, where *ako*(a-kind-of), *has-parts*, *color*, and *has-property*, are binary relations.

There are several benefits of using Semantic Networks for representing knowledge:

- Real-world meanings (semantics) are clearly identifiable.
- Reflects the structure of the part of the world being represented in the knowledge structuring.
- The representation due to “is-a” and “is-partof” relations help in organizing the inheritance based hierarchies, which are useful for inheritance-based inferences.
- Accommodates a hierarchy to be useful for default reasoning (e.g., we can assume the height of an adult as 175 cm, but if the person is a basketball player, then we take it as 190 cm).
- The semantic networks are useful in representing events and natural language sentences, whose meanings can be very precise. However, the concept of semantic networks is very general. This causes a problem, unless we are clear about the syntax and semantics in each case.

The Semantic networks have been used for knowledge representation in various applications like, natural language understanding, information retrieval, deductive databases, learning systems, computer visions, and speech generation systems.

7.2.1 Syntax and Semantics of Semantics Networks

Unlike the predicate logic, there is no well accepted syntax and semantic for semantic networks. A syntax for any given system is determined based on the objects and relation primitives chosen and the rules used for connection of the objects. However, there are some primitives which are quite established. We can define a Semantic Network by specifying its fundamental components:

1. *Lexical part*:
   a. *Nodes* denote the objects.
   b. *Edges* or *links* denote the relations between objects.
   c. *Labels* denoting the particular objects and relations between them.
2. *Structural part*: The nodes and the edges connecting them form directed graphs, and the labels are placed on the edges, which represent the relation between nodes.
3. *Semantic part*: Meanings (semantics) are associated with the edges and node labels, whose details depend on the application domains.
4. *Procedural part*: The *constructors* are part of the procedural part, they allow for the creation of new edges (links) and nodes. The *destructors* allow the deletion of edges and nodes, the *writers* allow the creation and alteration of labels, and the *readers* can extract answers to questions. Clearly, there is plenty of flexibility in creating these representations.

The word-symbols used for the representation are those which represent object constants and n-ary relation constants. The network nodes usually represent nouns (objects) and the arcs represent the relationships between objects. The direction of the arrow is taken from the first to the second objects, as they represent in the relations. The Fig. 7.2 shows a *is-a* hierarchy representing a semantic network. In set theory terms, *is-a* corresponds to the *sub-set* relation '$\subseteq$', and an *instance* corresponds to the membership relation '$\in$' (an object class relation) [3].

The commonly used relations are: *Member-of*, *Subset-of*, *ako* (a-kind of), *has-parts*, *instance-of*, *agent*, *attributes*, *shaped-like*, etc. The 'is-a' relationship occurs quite often, like, in sentences: "Rajan is a Professor", "Bill is a student", "cat is a pet animal", "Tree is a plant", "German shepherd is a dog", etc. The 'is-a' relation is most often used to state that an object is of a certain type, or to state that an object is a *subtype* of another, or an object is an *instance* of a class.

Figure 7.2 shows some important features of semantic networks. The representation makes it easy to retrieve the properties of any object efficiently due to hierarchy of relations. These networks implement property of inheritance (a form of *inference*).

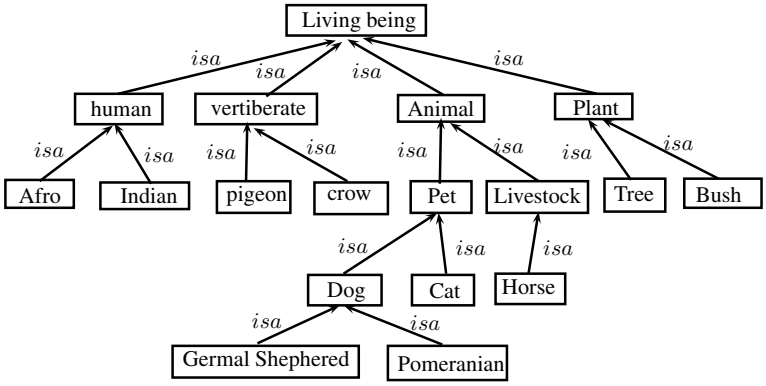

Fig. 7.2 Semantic network showing "Is-a" Hierarchy

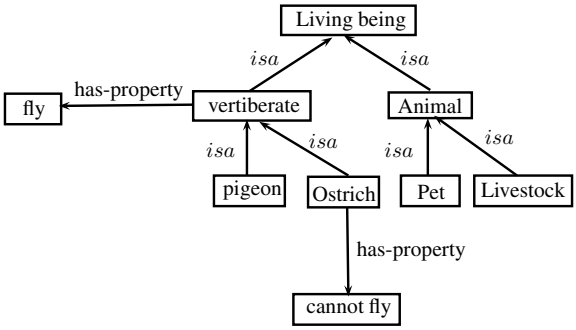

Fig. 7.3 A semantic network showing contradiction to an inherited property

The nodes which are members or subsets of other nodes may inherit the properties from their higher level ancestor nodes. For example, we can infer from Fig. 7.2 that dogs are animals and pigeons are vertebrates, and both of them are living beings. The property inherited like this is recognized as *default reasoning*. It is assumed that unless there is an information to the contradictory, it is reasonable to inherit the information from the ancestor nodes. In Fig. 7.3, Pigeon inherits the property of "can fly" from the vertebrates, while Ostrich has a locally installed attribute of "cannot fly", hence the property 'fly' will not be inherited by it.

The inference procedures for semantic networks can also be in parallel to those in propositional and predicate logic. For example, if a class A of objects have property P, and 'a' is a member of A, we can infer that 'a' has property P. The inferences in these networks can be defined as per those in predicate logic, making use of unification, chaining, modus ponens, and resolution, however, the reasoning is *default* type [13].

7.2.2 Human Knowledge Creation

The semantic networks are based on the *associationist* theory, which defines the meaning of an object in terms of a network of associations with other objects. When human perceives and reasons about an object, that perception is first mapped into a concept. This concept is part of our entire knowledge about the world and is connected through appropriate relationships to other concepts. These relationships form an understanding of the properties and behavior of objects such as *snow*. For example, through associations, we associate the snow with other concepts like cold, white, snowman, slippery, and ice. Our understanding of snow and truth of statements such as "snow is white" and "the snowman is white" manifests out of this network of associations.

There is experimental evidence that, in addition to associate concepts, humans also organize their knowledge hierarchically, as per that information is kept at the highest levels, to be inherited by other concepts. The evidence have shown that if the concepts in the networks were far off in the hierarchy, it took a relatively longer time by humans to understand this relation compared to the concepts which were in proximity to each other. For example, with reference to Fig. 7.2, the human response will be faster to infer that "dog is an animal", compared to the statement "Pomeranian is a living being". This time difference is argued due to the fact that humans store the information in a hierarchical way. The fastest recall was for the cases where the traits were specific to the object. For example, the negative response for "Can Ostrich fly?", will be faster than the positive response for "Can pigeons fly?". This is because in the reasoning for the first path: "Ostrich → cannot fly", is faster that for "pigeon → vertebrate → fly", due to path length difference [1].

7.2.3 Semantic Nets and Natural Language Processing

The inheritance based system allows to store the knowledge at the highest level of abstraction. This results to reduction in size of the knowledge base, as well as it helps in updating the inconsistencies. The graphs can explicitly represent the relations using arcs and nodes, which helps in formalizing the knowledge of semantic networks. These networks can be used to answer various queries related to the knowledge base stored, using inferences.

Much of the research in network representation has been done in the field of Natural Language Understanding (NLU). Often, the natural language understanding requires the understanding of the common sense, the ways in which the physical objects behave, the interactions that occur between humans, and the ways in which the human institutions are organized. A natural language understanding the program must understand the intentions, beliefs, hypothetical reasoning, plans, and goals embedded in the natural language text. Due to these, the language understanding has been the driving force for knowledge representation.

The NLU programs define the words of the English language in terms of other words, like the English dictionary does, rather than defining in terms of primitive words or axioms. Thus, to understand the meaning of a word we traverse a network of words until we understand the meaning of the required word.

7.2.4 Performance

The network structures used to provide intuitive and useful representations for modeling semantic knowledge. These models are required to fulfill the following four objectives:

1. Is is possible to organize the human semantic knowledge using the general structural principles which are characteristics of semantic networks?
2. Is it possible that the human performance of semantic processing can be emulated in terms of general processes operating on semantic networks?
3. Is it possible to emulate the human processes of semantic retrieval and search on general structures of semantic networks?
4. Can the semantic networks emulate the human processes of semantic acquisition and development of semantics in humans?

For all the above questions, the required answer is not only in terms of yes/no, but to what degree it is achievable. Also, the answer depends on the applications, which exploit the features of these networks, as well as there is yet lot to be found out.

The best example of a semantic network is *semantic web*, which enables the people to access the documents and services on the Internet. The interface to service is represented in web pages written in natural language, which must be understood and acted on by humans. The existing web is augmented by the semantic web with formalized knowledge and data, to be processed by computers.

7.3 Conceptual Graphs

Although there is no accepted standard for semantic network representations, but something which is very close to the goal is *Conceptual Graphs.* It is the portrayal of mental perception which consists of basic primitive concepts and relationships which exist between them. The conceptual graphs may be regarded as formal building blocks of Semantic networks. When they are linked together, they form a more complex and useful network [4].

Simmons [12] suggested primitives to represent standard relationships, by using the *case structure* of English verbs. In the verb oriented approach, links define the roles played by nouns and noun phrases inaction of the sentence. Case relationships includes: agent, object, instrument, location, and time [12].

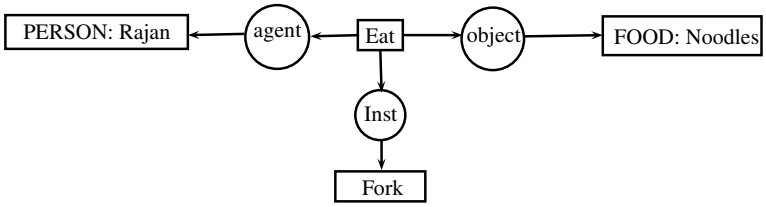

Fig. 7.4 Conceptual Graph-I

A sentence in the semantic network is represented with a *verb node*, and various case links to this node represent other participants in carrying out the action. The complete structure formed is called *case-frame*. While this sentence is parsed, the algorithm identifies the verb node, and retrieves the complete case-frame from the knowledge base. As a next step, the algorithm binds the values of an agent, object, etc., to appropriate nodes in the case-frame.

Example 7.1 Represent the sentence: "Rajan eats noodles with fork" as a conceptual graph.

The given sentence is represented by the conceptual graph shown in Fig. 7.4, and the corresponding predicate formula is given as Eq. 7.1.

$$\exists x \exists y(eat(x) \wedge person(Rajan) \wedge food(Noodles) \wedge fork(y) \wedge agent(x, Rajan) \wedge object(x, Noodles) \wedge inst(x, y)) \quad (7.1)$$

The concept we have represented is called *typed* or *sorted* version of logic, each of the four concepts (i.e., Rajan, noodles, eat, and fork) have type labels, that represent the type of entity the concept refers. The two concepts, Rajan and Noodles, which have the names, identify the referent, e.g., the concept [PERSON:Rajan] has type PERSON and referent Rajan. Three concepts have type labels: Agent, Instrument, Object. The Conceptual Graph (CG) as a whole indicates the semantic that a person "Rajan" is an agent of some instance of eating the food, noodles are an object, and the fork is an instrument. Eat and Fork has no fields to refer them by name, as these are generic concepts.

$$[PERSON : Rajan] \leftarrow (AGENT) \leftarrow [EAT] - \rightarrow (OBJECT) \rightarrow [FOOD : noodles] \leftarrow (INSTRUMENT) \leftarrow [FORK] \quad (7.2)$$

The concept symbols may represent entities, actions, properties, or events. For the Fig. 7.4, a linear conceptual graph form, which is easier to represent as text is given in the Eq. 7.2. □

We note that the representation of an English language sentence in the language of CG in Eq. 7.2, captures much of the deep structures of the natural language, such as the relationship between the verb and its subject (called the agent relation), and that between verb and object. When this sentence is parsed, the built-in relationship indicates that "Rajan" is a person, who is eating, and the fork is used for eating noodles (as object). These linguistic relationships are stored independent of the actual sentence, and are independent of the language of the sentence.

In 1976, John Sowa developed the concept of the conceptual graph, as an intermediate language, that mapped natural language questions and assertions to a relational database. The concepts were represented using symbols of rectangles, and the circles represented as conceptual relations. An arc that pointed to a circle, marked the first argument of the relation, and an arc pointing away from the circle marked the last argument. The relation is expressed in mathematical form as, $(relation(arg_1, arg_2, \ldots, arg_n)$. If there is only one argument, the arrow-head is omitted, while for relation with two or more arguments, the arrow-heads are replaced by integers, $1, 2, \ldots, n$.

Example 7.2 Represent the sentence "Rajan is going to Mumbai by bus", using a Conceptual Graph.

The Fig. 7.5 shows a CG for the sentence: "Rajan is going to Mumbai by bus."

In the CG all the four concepts, *Person*, *Go*, *Mumbai*, and *Bus*, have type label, for the type of the entity referred by the concept. Two concepts, Person and Destination have names. The verb "Go" is related to the remaining three concepts, by relations of agent, destination, and Instrument. The complete CG indicates that the person Rajan is an agent of some instance "Going", the city of Mumbai is the destination, and the bus is the instrument.

$$(\exists x)(\exists y)(Go(x)Person(Rajan)City(Mumbai)Bus(y) \\ Agent(x, Rajan)Dest(x, Mumbai)Inst(x, y)) \quad (7.3)$$

The Fig. 7.5 can be translated into the predicate formula (7.3). □

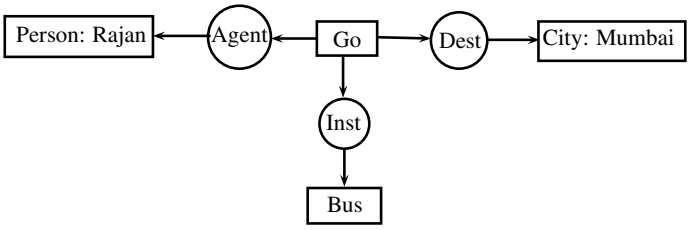

Fig. 7.5 Conceptual Graph-II

The only logical operators used in Fig. 7.5 are conjunction and the existential quantifier, as expressed in the Eq. 7.3, which are the most common operators in translations from natural languages, to first-order predicate logic [14].

7.4 Frames and Reasoning

The frames are structure-based knowledge representation technique, and are similar to semantic networks. The latter is based on the concept of human associative memory, but may simply be thought of as data structures of nodes—"concepts"—and links—"associations"—between them [3].

The concept of a frame was proposed in the 1970s by Minsky. As per Minsky, when we encounter a new situation, or there is substantial change in the present context, we select from memory a new structure, called *Frame*. This is a previously remembered framework, which is adapted to fit into the current set of things as required for the new situation, with necessary changes in the frame. The frame is a *data structure* to represent a stereotype situation, like a certain kind of living room, or a frame of a picnic, or of the classroom, etc. Each frame is attached with several kinds of information, where some information may also be about how the frame is to be used, while other information may be about what one may expect to be happening next, and some information may be about what is to be done if these expectations are not met, and so on [8].

Consider representing the sentence: "Car #12 is red"

Approach 1: *red(car12)*. With this representation, it is easy to ask "what is red", but we cannot ask "what is the color of car12?"
Approach 2: *color(car12, red)*. In this approach, it is easy to ask "What is red?" Also, we can ask, "What is the color of car12?" But, we cannot ask "What property of car12 has value red?"
Approach 3: *property(car12, color, red)*. With this, it is easy to ask all the above questions.

We call this representation as, object-property-value representation, and have format *property(Object, Property, Value)*. To get the object-centered representation, we merge many properties of the object of the same type into one structure, as follows:
property(Object, Property1, Value1)
property(Object, Property2, Value2)
…
property(Object, Property-n, Value-n)

The representation is called Frame, as shown in Fig. 7.6.

It is important to note that objects enable grouping of procedures for determining the properties of objects, their parts, and interaction with parts. The first step in structuring is to collect together all propositions concerning a particular object in a data structure, like records in PASCAL, property lists in LISP, or relations in a

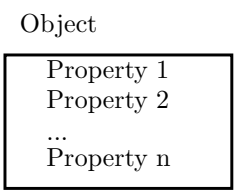

Fig. 7.6 A frame object

Object Property Values		
Elephant	is-a:	mammal
	color:	grey
	has:	trunk
	size:	huze
	habitate:	India/Africa

Fig. 7.7 A frame of "Elephant"

database. Figure 7.7 shows an example of a structured representation of facts by collecting together all the properties of an object in the data structure.

There are two types of frames: 1. The *Individual frames* represent a single object, e.g., a person, part of a trip; 2. Other type, the *Generic frames*, represent categories of objects, e.g., students. An example of a generic frame is, "Indian city", and individual frames are "Delhi", "Mumbai". An individual frame is a named list of *buckets*, also called *slots*. What goes in the bucket is called a *filler* of the slot.

```
(frame − name
< slot − name1 filler1 >
< slot − name2 filler2 > ...)
```

7.4.1 Inheritance Hierarchies

A frame is a network of nodes and relations, whose "top levels" are fixed, and represent the things that are always true concerning the supposed situation. The lower levels of this frame-based network comprise many terminals or "slots", which must be filled in by specific instances of the data. The conditions must also be specified for each terminal, under which the assignment be made. The assignments are usually smaller frames, called "sub-frames" (see Fig. 7.8). Simple conditions are indicated by markers, which might require a terminal assignment to be a person, an object of sufficient value, or a pointer to a sub-frame of a certain type. It is possible to specify relations among the things assigned to several terminals using more complex conditions.

The inheritance hierarchies serve for economic data conservation. Instead of storing all the properties of each object, all the objects are structured in a hierarchy, and only the individual properties are stored in the object itself, while the general properties are attached to the predecessors and inherited by all the successors.

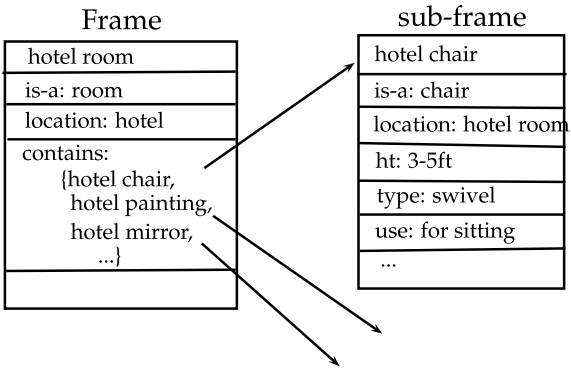

Fig. 7.8 A frame representation of hotel room

In object-centered representations, an object is a natural way to organize the knowledge about some physical objects, like "a desk has a surface-material, number of drawers, width, length, height, color, procedure for unlocking, etc." Some variations can be "no drawers, multi-level surface". Alternatively, an object may describe a situation, e.g., for a lecture-hall the complete set of situation is: hall, students, teacher, day, time, seating arrangement, lighting, grading, etc. Or, it can be about a trip with slot values as: origin (of trip), destination, procedures for buying ticket, transport, getting through customs, reserving a hotel room, locating a car rental, etc.

7.4.2 Slots Terminology

Every frame is identified by its individual name—the *frame name*; a frame consists of a set attributes associated with it (see Fig. 7.8). For example, in the frame "Person", slots may be: name, weight, height, and age; for the frame "Computer", the slots may be: model, processor, memory, and price. There is also a value attached to each attribute or slot. A frame-based approach provides a natural way of structured and compact knowledge representation. The knowledge is organized in slots that describe various attributes or properties of any object. This approach is appropriately suited for object-oriented programming of expert systems.

Following is the typical Information included in a Slot:

1. *Relationship*: A frame provides the relationship to the other frames. The frame *Hotel room* (Fig. 7.8) can be a member of other frame class *Room*, which in turn can belong to the class *Housing*, thus providing the relationship with these other room types.
2. *Slot value*: The value of a slot can be numeric, symbolic, or Boolean (True/False). For example, a slot identified as 'Person' is symbolic, with slot names as 'Age', and 'Height', both having float values. The slot's values can be dynamically assigned during a session with the expert system, or they can be static, or can be initialized in the beginning while the slot is created.

3. *Default value of slot*: A slot may contain default value when the true value is not available, and there is no evidence that the value chosen is in no way providing any contradiction. For example, in a frame named as *Car* when slot values are not provided, default values of the slots: *wheels-count* and *Engine-count* can be taken as 4 and 1, respectively.
4. *Slot value's range*: The range of a slot's value is useful in checking the bounds of the slot value—whether the provided value of a slot is within the prescribed limit? For example, a pressure range of a car tire may range 30–50 psi (pounds per inch).
5. *Slot Procedure*: A slot has a procedure attached to it; when this procedure is called it may read a value from the given slot, or it can update the value of the slot (write the value in it).
6. *Facets*: The facets provide an extension to slot value structure in a frame-based expert system. A facet provides extended knowledge about the attribute of a frame. It can be used for establishing the attributed value of a frame, it can control end-user queries, and can direct the inference engine as how to process the attributes.

7.4.3 Frame Languages

Frame-based languages are knowledge representation languages, where a frame designates a concept or a concrete entity. A concept is represented by a generic frame, called class, and concrete entity illustrates one or more classes. These classes are also represented by a specific frame, called *instance*. A frame, whether it is a class or instance, is a data structure composed of slots which carry the properties of the entity described by the frame or relations between frames.

The slots themselves are described using facets, which contribute to the description semantics of the slot. The facets are two types: 1. descriptive (passive), and 2. active (reflexes). The passive faces represent domain, vales, or defaults. The active facets (also called *daemons* or *procedural* facets) are concerned to actions. They start with keywords like, If-needed, If-created, which gets triggered after the value of a slot is manipulated. The slots are characterized either by vales introduced by the *value facet*, or default values, which reintroduced by the *default facets*.

The classes are organized as a hierarchy, and relations called structural links, that connect the frames. A commonly used link, ako (a-kind-of) connect the classes within the hierarchy, whereas "Is-a" link allows to connect instances to classes to which they belong. A class connected by a link ako to another class, called *mother-class*, inherits all the properties possessed by the mother-class. These properties are represented using slots. The presence of a value within a class slot means this value is true for this slot, as well as it is true for all the sub-classes and the instances of that class, where the value has been declared. Specifying a default value in a class means this value is generally true for that slot, also for the sub-classes, and the instances of that class where the default is declared. However, the default can be overridden by declaring an exception to the default, at any of the sub-class(es).

It is assumed that there is no multiple inheritance conflicts among these slots. The frames of the hierarchy communicate by means of messages. The frames make use of methods (processes), which gets executed when a message is received by the frame in which this message is declared or when the message is received by its sub-classes or instances.

A domain knowledge base builder uses the frame languages to describe the type of objects to be modeled by the system, and this representation can be provided efficiently by the frame languages. The object description of certain type may contain a prototype description of objects of that type. These objects are used for creating a default description of an object when its type becomes known in the model.

The frame-based languages are good for providing facilities for describing object attributes. For example, a frame representing a *car* might include descriptions for the length of the car, number of seats, size of the engine, engine's horse-power, etc. These attributes can be used to include partial descriptions of the attribute values, and are helpful in preserving the semantic integrity of a system's knowledge base by restricting the number and range of permitted attribute values. The frame languages do not provide a facility for describing the declarative behavior, however, they have facilities for attaching procedural information expressed in some other language, e.g., LISP. The procedural capability enables behavioral models of objects, and as an expert in an application domain. It also provides a powerful tool for object-oriented programming, where frames are treated as objects, and they respond to messages.

There are two standard forms for attaching the procedures: 1. *methods*, and 2. *active values*. In languages, like LISP and Prolog, the methods are used as procedures, that are attached to the frames, and respond to messages sent to the frames. Just like the attributes, the methods are also stored as values in slots, and they are recognized as responders to the messages. The messages sent to the frames carry the information about target message-responder slot, these messages contain the arguments needed by the methods stored in the slot. The "actives values" in the slots are either procedures or collection of production rules attached to the slot. These (procedure or rules) are invoked when the slot's values are accessed or new values are stored in the slots. Thus, these slots behave like "daemons", and monitor the change and usages of the values.

The active values can also be used to dynamically compute values on a "when-needed" basis. The methods and active values are so written that they apply to any member of a class of objects, and are included by the knowledge base designer in the class description as a part of the prototype description of a class member.

7.4.4 Case Study

Many systems make use of the approach discussed above to control the reasoning, where functions or rule classes behave like daemons, are attached to the slots of the frames. The attachments get invoked when the value of a slot changed. That way, they behave like sensors or alarms, or monitors. For example, an expert system is used for

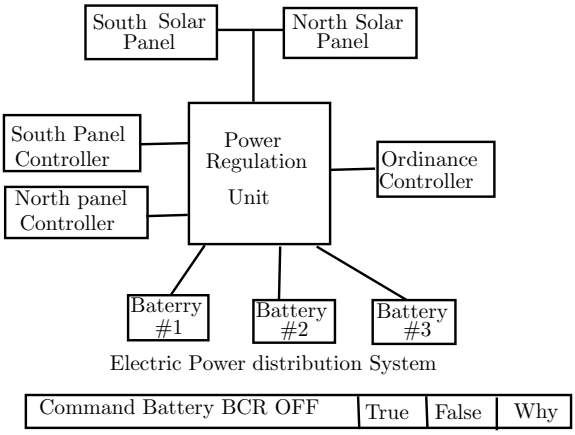

Fig. 7.9 A satellite diagnostic system

an intelligent alarm facility, and this calls a user supplied function only when a slot's value crosses the threshold. The user may establish an alarm by providing critical boundaries, and an alarm function. The system stores the boundaries and function as facets of the slots, and may attach generic active value for check of boundary crossing, whenever a value of the slot changes.

An example of the above phenomenon can be found in the knowledge system (see Fig. 7.9), which is a Satellite diagnostic system . The system is meant to serve as an intelligent facility for human operators to perform diagnostic and trouble-shooting of satellite malfunctions. Such systems can also be used as a simulator to train the operators and diagnostic experts. An important requirement of the diagnosis task is that it requires detailed analysis of the satellite system through a diverse set of domain experts. The software architecture of the prototype, which can be built using an expert system, will make use of daemons attached to the slots to respond to messages sent to objects. The prototypes of experts have the property that they can be instantiated, as well as deleted dynamically, as per the need, during the operation of the system.

Significant events can be controlled. This system makes elaborate and effective use of frames, including prototype expert frames, which also, like a daemon, can be instantiated and deleted dynamically as per the requirement, during operation of the system. The use of frame-based approach to build the system's model allowed the designers to organize the frames in such a way, that knowledge could easily be accessible and comprehensible to diagnostic experts, as well as the satellite operators [9].

A satellite diagnostic, e.g., STAR-PLAN, is designed with several requirements, that resulted in an architecture based on the integration of frames and production rules, with the following important features:

1. First, the system's knowledge is made accessible and comprehensible to the satellite operators as well as to diagnostic experts.
2. The system is incrementally and progressively built-up, as the descriptions of additional satellite modules become ready, and as the needed experts became

available. Accordingly, the system's knowledge is required to be partitioned into a limited number of experts, for example, knowledge about a particular type of malfunction, or about a particular module of the satellite.

3. In the case of STAR-PLAN, it was known to the designers that eventually, the system will become very large, and it would be operating in real-time. To meet the speed requirements, the system was so designed that only part of the system shall be awakened—the one which is required at a certain moment of time, and other shall be put to sleep.

Role of Frame Language

The designers of STAR-PLAN used frame language to build the taxonomy that described parts of a typical communication satellite. The daemons and methods were later associated with the prototype in the taxonomy. This maintained the relationship between various parts, and defined the behavior of each part, which resulted in a kind of object-oriented programming style, built for creating simulation behavior.

To model the diagnostic part, two separate taxonomies were constructed, such that each class in the first taxonomy represented experts who were assigned the task of "watching over" to some particular components of the satellite. The members of these classes were called *Guardians*. In the second taxonomy, each class represented experts responsible for responding to a particular type of problem that may occur in the satellite. The members of the class in this second taxonomy were called *Monitors*.

For each component of the satellite, guardians were created and initialized at the start of the satellite system. When an *initialize* message is sent to a guardian, it places an intelligent alarm in the system's model of the satellite. These alarms would wake-up their guardian by sending a message when a problem has occurred in the satellite. Hence, the guardian is active only when the satellite demands.

Alarms/Inferencing

The methods of *guardian* system respond to messages from the *daemons*. This is done by invoking a class of diagnostic rules that determine what kind of problem is occurring. For example, to find out all the possible consequences of an anomalous situation that might have tripped the alarms, the set of rules is applied in forward-chaining manner. In the process, all the rules are applied that have conditions matching some aspects of the anomalous situations, or it matches a conclusion of the already applied rule. This process of rules' application continuous until no match is left.

The total number of rules associated with this system are small in number, typically, 10–20, so that modularization provides a small expert system of closely related rules to be focused to develop a guardian.

As soon as a guardian comes to know about the occurrence of a problem, it creates a*monitor* and initializes it. This represents an expert for the problem. The monitor's task is to watch the problem right from its evolution, and to make the recommendations to the satellite operator. Once the initialization is done, the monitor may create its own daemons and put itself to sleep for a fixed duration. When the monitor wakes up itself or it is awakened by a message from one of its daemons, it invokes a set of rules to analyze the status of the satellite. If the monitor

is waking-up itself after a fixed amount of time, the rules will be invoked by a backward-chaining rule interpreter that tests a specific hypothesis (goal) about the problem. The backward-chaining system attempts to find a sequence of rule applications that should conclude the hypothesis. When such a sequence is found, the rules are applied so that the hypothesis is added into the knowledge base.

Based on the conclusions arrived-at making use of rules, the monitor will either put itself to sleep again or make recommendations to the operator. When the monitor concludes that the problem has been solved, it removes its daemons and then removes (deletes) itself, and frees the memory occupied.

Due to the creation and deletion of monitors in a dynamic way, the satellite problems by the STAR-PLAN system model are actually handled like by human operators and experts. The situation is like in real-world: when a problem is identified, a suitable expert is called in, which works with the team until the problem is resolved, and then the expert leaves! The monitor's rule sets are organized for problem specific knowledge base about the problems of the satellite. The module-based system and its organization structure also makes it easier for a (human) domain expert to create and debug the knowledge base of rules.

7.5 Description Logic

Approaches to knowledge representation are sometimes divided roughly into two categories: *logic-based* formalisms, which evolved out of the intuition, that predicate calculus could be used unambiguously to capture facts about the world; and other, non-logic-based representations. The latter was often developed by building on more *cognitive notions*—for example, network structures, and rule-based representations derived from experiments on *recall* from human memory and human execution of tasks like mathematical puzzle solving. Even though such approaches were often developed for specific representational chores, the resulting formalisms were usually expected to serve in general use [5].

Since first-order predicate logic (FOPL) provides very powerful and general machinery, logic-based approaches were more general purpose from the very start. In a logic-based approach, the representation language is usually a variant of the first-order predicate calculus, and reasoning amounts to verifying logical consequence. In the *non-logical* approaches, often based on the use of graphical interfaces, knowledge is represented by means of some ad hoc data structures, and reasoning is accomplished by similarly ad hoc procedures that manipulate the structures. Among these specialized representations we find semantic networks and frames. However, frames and semantic networks lack formal semantics. Description Logic ($\mathscr{DL}$) was first introduced into Knowledge Representation (KR) systems to overcome these deficiencies of semantic networks and frames. The $\mathscr{DL}$ makes it easier to describe definitions and properties of categories. The $\mathscr{DL}$ evolved from semantic networks to formalize the network representation while retaining the emphasis on taxonomic structures as an organizing principle.

A Description Logic models *concepts*, *roles* and *individuals*, and their relationships. The fundamental modeling concept of a $\mathscr{DL}$ is the axiom: "a logical statement relating roles and/or concepts". $\mathscr{DL}$ is a family of formal knowledge representation languages, which is more expressive than propositional logic and has more efficient decision properties than first-order predicate logic. It is used in formal reasoning on the concepts of an application domain (known as terminological knowledge). It is used for providing a logical formalism for ontologies and the Semantic Web. Modern ontology languages are based on $\mathscr{DL}$, such as OWL (Ontology Web Language).

7.5.1 Definitions and Sentence Structures

A $\mathscr{DL}$ describes the domain in term of the following:

- *Individuals*—are the things in the world that are being described. (For example a house, book, ram, john, rita, etc, all starting with lowercase letters).
- *Classes/Categories/Roles*—are sets of individuals. It is a *ako* (a kind of) concept. A class is a set of all real or potential things that would be in the class. For example, Hunter, Teenager, etc.
- *Properties/Relations*—are used to describe individuals. It is *ako* Roles or relational nouns, and used to describe objects that are parts or attributes or properties of other objects. Examples are: Child, Mother, Age, etc.

Two different sets of symbols—*logical symbols* (with a fixed meaning) and *non-logical symbols* (domain-dependent) are used in the Description Logic.

Following classes of *Logical symbols* are used in $\mathscr{DL}$:

Punctuation: (,), [,]
Positive integers
Concept-forming operators: $\forall$, $\exists$, *FILLs*, *AND*.
Connectives: $\sqsubseteq$, $\doteq$, $\rightarrow$, $\sqcup$, $\sqcap$.

Non-logical symbols:

- Constants: `john`, `rajanShaw` (camel casing, but starting with uncapitalized letter.
- Atomic concepts: `Person`, `FatherOfOnlyGirls`, `Hunter`, `Teenager` (camel casing, first letter capital).
- Roles: `:Height`, `:Age`, `:FatherOf`, `:Child`, `Mother` (same as concepts, but precede by colons).

Concepts

In terms of semantics, the concepts are given set-theoretic interpretation, where a concept is interpreted as a set of individuals, and roles are a set of pairs of individuals. An interpretation domain can be chosen arbitrarily, which can be infinite also. The infinite domain and the open-world assumptions are distinctive features of Description Logic.

7.5.2 Concept Language

Atomic concepts are thus interpreted as subsets of the interpretation domain, while the semantics of the other constructs is then specified by defining the set of individuals denoted by each construct. For example, the concept $C \sqcap D$ is the set of individuals obtained by intersecting the sets of individuals denoted by C and D, respectively. For example, *Female* $\sqcap$ *Teacher*. Similarly, the interpretation of $\forall R.C$ is the set of individuals that are in the relationship R with individuals belonging to the set denoted by the concept C. For example, $\forall$*ResidentsOfJodhpur*.*Students* represents all the students who are residents of Jodhpur.

There exists some ambiguity also, due to the natural language being the source, where many nouns can be used to refer as a category as well as relations. For example, `child` can be used as a *category*, i.e., a very young person, it can also be used to represent a relation, which stands for the inverse of `parent`.

An important feature of Description Logic is to define complex concepts in terms of simpler ones. This is achieved by means of *concept-forming operators*: $\exists$, $\forall$, *AND*, *FILLs*. A *complex concepts* is defined as [4, 10]:

Every atomic concept is a concept;
If R is a role and C is a concept, then $\forall R.C$ is a concept;
If R is a role and n is a positive integer, then $\exists n.R$ is a concept;
If R is a role and C is a constant, then *FILLs* $R.C$ is a concept; and
If $C_1 \ldots C_n$ are concepts, then *AND* $C_1, \ldots, C_n$ is a concept.

The symbol $\exists$ stands for the class of individuals in the domain that are related by relation R to at least n other individuals. The following can be created as complex concepts:

$\exists$ 1.*Child*: All the individuals (the class of the individuals) that have at least one child.

$\exists$ 2.*HasCar*: All the individuals that have at least two cars.

$\exists$ 6.*HasWheels*: All the individuals that have at least six wheels.

The *FILLs* $R.C$ stands for those individuals that are related (*R-related*) to the individual identified by C. For example, "All the individuals that have the car with plate *RJC12* is represented by *FILLs HasCar*.*RJC*12.

The $\forall$ $R.C$ stands for those individuals that are *R-related* only to individuals of class C. For example, $\forall$ *BeingInThisRoom*.*PHDStudents* represents "All the individuals that are in this room and are Ph.D. students."

The full syntax of a concept in $\mathscr{D}\mathscr{L}$ is:

$$
\begin{aligned}
Concept :- & Thing \mid conceptName \\
& \mid AND(concept_1, concept_2, ...) \\
& \mid \forall RoleName.concept \\
& \mid \leq ingtger.RoleName \\
& \mid \geq integer.RoleName \\
& \mid FillsRoleName.IndividualName \\
& \mid SameAs(Path, Path) \\
& \mid \exists IndividualName.concept \mid \top \mid \bot \\
Path :- & [RoleName, ...]
\end{aligned}
$$

The family of concept languages is called Attribute Language ($\mathscr{A}\mathscr{L}$) , which is minimal language that is of practical importance. Given that *Person* and *Female* are both atomic concepts, then $Person \sqcap Female$ and $Person \sqcap \neg Female$ are $\mathscr{A}\mathscr{L}$-concepts which intuitively describe, persons that are female, and those that are not female. Suppose that *hasChild* is atomic role, then $Person \sqcap \exists hasChild.\top$, and $Person \sqcap \forall\ hasChild.Female$ denote those persons that have a child, and all those whose children are female. Opposite to the *top*, there is a *bottom* concept ($\bot$), using which we can describe the persons without a child: $Person \sqcap \forall hasChild.\bot$.

Sentences

A knowledge base in a Description Logic is collection of sentences like,

If C_1 and C_2 are concepts, then $(C_1 \sqsubseteq C_2)$ is a sentence;

If C_1 and C_2 are concepts, then $(C_1 \doteq C_2)$ is a sentence;

If C is a constant and D is a concept, then $(C \rightarrow D)$ is a sentence;

For example,

$PhDStudent \sqsubseteq Student$, i.e., Every Ph.D. student is also a student (not vice versa).

$C_1 \doteq C_2$, i.e., concept C_1 is equivalent to concept C_2 , i.e. the individuals that satisfy C_1 are precisely those that satisfy C_2.

$PhDStudent \doteq AND(Student,\ Graduated,\ HasFunding)$, i.e., a Ph.D. student is a student that is already graduated, and that has some funding.

$C \rightarrow D$, i.e., the individual denoted by C satisfies the description expressed by concept d. For example, $rajan \rightarrow PostDoc$, i.e. "Rajan is a Post Doc."

When compared with FOPL, the FOPL focuses on sentences, and it does not help you with reasoning on complex categories. For example, we can say that X is a hunter by a 1-ary predicate $Hunter(X)$. Similarly, there is 1-ry predicate, we can say $Shooter(X)$. What if we want to say is that X is both a hunter and a shooter. In predicate logic, it is

$$Hunter(X) \wedge Shooter(X),$$

whereas in $\mathscr{DL}$ it is a 2-ary relation

$$Hunter\&Shooter(X).$$

or

$$AND(Hunter, Shooter).$$

In the $\mathscr{DL}$, intersection of concepts, which is denoted $C \sqcap D$, is used to restrict the set of individuals under consideration to those that belong to both C and D. In the syntax of $\mathscr{DL}$, concept expressions are variable-free. In fact, a concept expression denotes the set of all individuals satisfying the properties specified in the expression. Therefore, $C \sqcap D$ can be regarded as the first-order logic sentence, $C(x) \wedge D(x)$, where the variable ranges over all individuals in the interpretation domain and $C(x)$ is true for those individuals that belong to the concept C.

We can represent the concept of "persons that are not female" and the concept of "individuals that are female or male" by the expressions:

$$Person \sqcap \neg Female$$

and

$$Female \sqcap Male.$$

The key characteristic features of $\mathscr{DL}$ lies in the constructs that are helpful for creating relationships between concepts. The most common and elementary is the *value restriction*, written as $\forall R.C$, which means all the individuals that are having relationship R with the concept being described, belong to concept C. Similarly, $\exists R.C$ is a value restriction for some individuals.

Example 7.3 Represent the concepts of semantic network form in Fig. 7.10, using $\mathscr{DL}$.

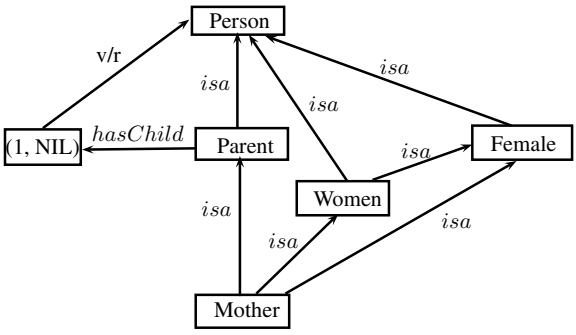

Fig. 7.10 Semantic network hierarchy

Assume that the atomic concepts are: `Female`, `Person`, and `Woman`. And, `hasChild`, `hasFemaleRelative` are atomic roles. They use the operators *intersection*, *union* and *complement* of concepts, interpreted as set operations.

The Description Logic has a characteristic feature as their ability to represent other kinds of relationships that can hold between concepts, this is beyond the IS-A relationships. For example, in Fig. 7.10, the concept of *Parent* has a property that is usually called a "role," and expressed by a link from the concept to a node for the role, labeled `hasChild`. The role has a "value restriction," denoted by the label v/r, which expresses a limitation on the range of types of objects that can fill that role. In addition, the node has a number of restrictions expressed as (1, NIL), where the first number is a lower bound on the number of children and the second element is the upper bound, and NIL denotes no restriction on the upper limit. Overall, the representation of the concept of *Parent* here can be read as "A parent is a person having at least one child, and all of his/her children are persons", thus the role link between `Parent` and `Person` in Fig. 7.10 can be expressed as a concept expression $\mathscr{DL}$ as,

$$\exists 1.\texttt{hasChild.Parent} \sqcap \forall \texttt{hasChild.}\textit{Person}.$$

Existential quantification and value restrictions are thus meant to characterize relationships between concepts. Such an expression therefore characterizes the concept of *Parent* as the set of individuals having at least one filler of the role `hasChild` belonging to the concept Person; moreover, every filler of the role `hasChild` must be a person.

Relationships are inherited from concepts to their subconcepts. For example, the concept `Mother`, i.e., a female parent, is a more specific descendant of both the concepts `Female` and `Parent`, and as a result inherits from `Parent` the link to `Person` through the role `hasChild`; in other words, `Mother` inherits the restriction on its `hasChild` role from `Parent`. The other relations are translated as:

$$\texttt{Woman} \doteq \texttt{Person} \sqcap \texttt{Female}.$$
$$\texttt{Mother} \doteq \texttt{Woman} \sqcap \texttt{Female} \sqcap \texttt{Parent}.$$
$$\forall \texttt{Female.Person}$$
$$\forall \texttt{Parent.Person}.$$

we observe that there may be implicit relationships between concepts. For example, if we define `Woman` as the concept of a *female person*, it is the case that every `Mother` is a `Woman`. It is the task of the knowledge representation system to find implicit relationships such as these (many are more complex than this one). Typically, such inferences have been characterized in terms of properties of the network. In this case, one might observe that both `Mother` and `Woman` are connected to both `Female` and `Person`, but the path from `Mother` to `Person` includes a node `Parent`,

which is more specific then Person, thus enabling us to conclude that `Mother` is more specific than `Person`. □

7.5.3 *Architecture for $\mathscr{DL}$ Knowledge Representation*

A knowledge representation system based on Description Logic provides facilities to set up knowledge base, to reason about their concepts, and manipulate them. The Fig. 7.11 shows the related blocks and their interactions for this purpose.

The KB comprises two components, the *TBOX* (terminology Box) introduces the terminology, i.e., the vocabulary of an application domain; and the *ABOX* (assertion box) contains assertions about the named individuals in terms of vocabulary.

The vocabulary consists of concepts, which denotes the individuals, and the roles which denote the binary relationship between the individuals. In addition to these, $\mathscr{DL}$ system allows the users to build a complex description of concepts and roles. The *TBOX* can be used to assign names to complex descriptions. The description language has model theoretic semantics. Consequently, the semantics in *ABOX* and *TBOX* are FOPL formulas or its extensions.

The $\mathscr{DL}$ system provides the services for reasoning using using KB, typically, to reason if the terminology is satisfiable. The reasoning process checks that the assertions are consistent. With subsumption testing, it is easy to organize the concepts of terminology in the hierarchy.

In any application, the KR system is embedded into a large environment. The other components interact system through queries to KB and by modifying it, i.e., by adding or retracting concepts, roles, and assertions.

The basic form of declaration in a *TBox* is a concept definition, that is, the definition of a new concept in terms of other previously defined concepts. For example, a woman can be defined as a female person by writing this declaration:

$$\texttt{Woman} \doteq \texttt{Person} \sqcap \texttt{Female}$$

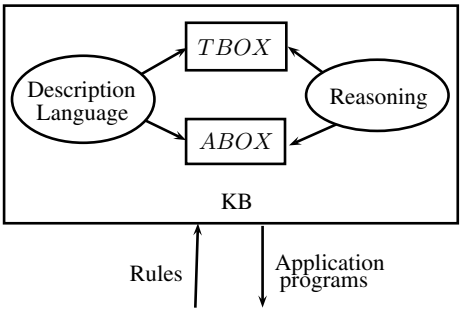

Fig. 7.11 $\mathscr{DL}$ based knowledge representation architecture

There are some important common assumptions usually made about $\mathscr{DL}$ terminologies:

- Only one definition for a concept name is allowed;
- Definitions are acyclic in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

The *ABox* comprises extended knowledge about the domain of interest, which are, assertions about individuals, called membership assertions. For example,

$$\texttt{Female} \sqcap \texttt{Person}(\texttt{sita})$$

show that the individual named as `sita` is a female person. Given this definition of woman, one can derive from this assertion that `sita` is an instance of the concept Woman. Similarly,

$$\texttt{hasChild}(\texttt{sita}, \texttt{luv})$$

indicates that `sita` has `luv` as a child. Assertions of the first category are also called *concept assertions*, while of the second is called *role assertions*.

7.5.4 Value Restrictions

Let us now turn our attention to role restrictions by looking first at the quantified role restrictions and, subsequently, at what we call "number restrictions." Most languages provide (full) existential quantification and value restriction that allows one to describe, for example, the concept of "individuals having a female child" as $\exists\texttt{hasChild.Female}$, and to describe the concept of "individuals all of whose children are female" by the concept expression $\forall\texttt{hasChild.Female}$. In order to distinguish the function of each concept in the relationship, the individual object that corresponds to the second argument of the role viewed as a binary predicate is called a role filler. In the above expressions, which describe the properties of `Parents` having female children, individual objects belonging to the concept `Female` are the fillers of the role `hasChild`.

Another important kind of role restriction is given by number restrictions, which restrict the cardinality of the sets of fillers of roles. For instance, the concept

$$(\geq 3\ \texttt{hasChild}) \sqcap (\leq 2\ \texttt{hasFemaleRelative})$$

represents the concept of "individuals having at least three children and at most two female relatives." Number restrictions are sometimes viewed as a distinguishing feature of Description Logics, although one can find some similar constructs in some database modeling languages (notably Entity-Relationship models).

Beyond the constructs to form concept expressions, Description Logics provide constructs for roles, which can, for example, establish role hierarchies. However,

the use of role expressions is generally limited to expressing relationships between concepts.

Intersection of roles is an example of a role-forming construct. Intuitively, $\texttt{hasChild} \sqcap \texttt{hasFemaleRelative}$ yields the role "has-daughter," so that the concept expression

$$\texttt{Woman} \sqcap \leq 2\ (\texttt{hasChild} \sqcap \texttt{hasFemaleRelative})$$

denotes the concept of "a woman having at most 2 daughters".

Example 7.4 Represent the following statement in Description Logic: A cheese pizza is defined as a pizza having toping and having a pizza base. The topping is a cheese topping, while the base is a pizzabase. A cheese topping is a topping.

$$\begin{aligned} \texttt{CheesePizza} &= \texttt{Pizza} \\ &\sqcap (\exists \texttt{hasTopping.CheeseTopping}) \\ &\sqcap (\exists \texttt{hasPizzabase.PizzaBase}). \\ \texttt{cheeseTopping} &\sqsubseteq \texttt{Topping}. \end{aligned}$$

□

7.5.5 Reasoning and Inferences

The basic inference on concept expressions in Description Logics is *subsumption*, typically written as $C \sqsubseteq D$ (read as "C is subsumed by D"). Sometimes, this axiom type is also referred to as *is-a* relationship, inspired by the often chosen wording for this type of statement (e.g. "a cat is a mammal" would be a typical verbalization of $\texttt{Cat} \sqsubseteq \texttt{Mammal}$). Determining subsumption is the problem of checking whether the concept denoted by D (the *subsumer*) is considered more general than the one denoted by C (the *subsumee*). In other words, subsumption checks whether the first concept always denotes a subset of the set denoted by the second one.

The principle inferences of $\mathscr{DL}$ are *subsumtion*—checking if one category is a subset of another category, and classification checking, whether an object belongs to a category.

For example, one might be interested in knowing whether $\texttt{Woman} \sqsubseteq \texttt{Mother}$. In order to verify this kind of relationship, one has, in general, to take into account the relationships defined in the terminology in Fig. 7.10.

Given a knowledge base expressed as a set S of sentences:

"Does a constant c satisfies concept d?"
"Is a concept c subsumed by a concept d?"

Answering to these questions amount to compute the entailment. For example, representation for "A Ph.D. student is, a student that already graduated, and that has some funding." is:

$$\texttt{PhDStudent} \doteq \texttt{AND(Student, Graduated, HasFunding).}$$

As another example, to say that "Bachelors are unmarried adult males", we write in $\mathscr{DL}$ as

$$\texttt{Bachelor} \doteq \texttt{Unmarried} \sqcap \texttt{Adult} \sqcap \texttt{Male}$$

The most important aspect of $\mathscr{DL}$ is its emphasis on tractability of inference. A problem instance is solved by designing it and then asking if it is subsumed by one of several possible solution categories. The complexity of $\mathscr{DL}$ is far simpler than FOPL. The $\mathscr{DL}$ usually also lacks the negation and disjunction operators.

The main application domains of description logic are: software engineering, configuration of large software, digital libraries, Web-based information systems, Planning, and Data Mining.

7.6 Conceptual Dependencies

The Conceptual Dependency (CD) framework is a simplified linguistic system, aimed to provide a computational theory of simulated performance. In Conceptual Dependency terms, the linguistic process is a mapping into and out of some mental representation. This mental representation consists of concepts related to each other by various meaning-contingent dependency links. Each concept in the interlingual network may be associated with some word that is its realization on a sentential level.

A representation using conceptual dependency is a linked network, which characterizes the conceptualization inherently present in a piece of language, with reference to a real-world scenarios. A simple rule for representing concepts as dependent on other concepts is, to check whether the dependent concept further explains its governor, and this concept cannot make sense without its governor. For example, in the sentence,

"The big man steals the red book from the girl.",

the analysis of the sentence is as follows: The article, 'The' stands for connecting sentences in paragraphs, i.e., 'The' also specifies that 'man' might have been used previously. The adjective, 'Big' refers to the concept 'big', which cannot stand independently. The concept 'man', however, can stand alone, but in the above sentence it is conceptually modified by 'big', and in the network, it is realized as *governor* with its dependent. The verb 'steals' is an *action*, which is dependent on the concept of doing the acting. A conceptualization (a statement about a conceptual actor) cannot be complete without a concept acting (or an attribute statement). Thus, to complete a two-way dependency, a link must exist between the 'man' and the 'steal'.

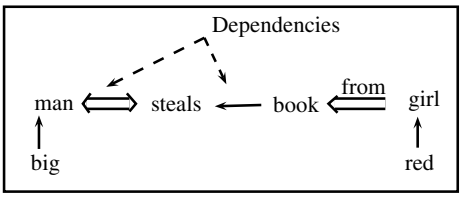

Fig. 7.12 CD for "The big man steals the red book from the girl"

Which indicates that they are dependent on each other, and also govern each other as shown in Fig. 7.12.

It is mandatory that every conceptualization has a two-way dependency link. In the Fig. 7.12, the concept 'Book' governs the concept 'red' attributively (color is an attribute), and the whole entity is placed as objectively dependent on 'steals'. The construction "from the girl" is realized as being dependent on the action through the conceptual object. This is *prepositional* type of dependency (denoted by $\Leftarrow$). There are different forms of this prepositional dependency, each of which is expressed by writing the preposition over the link, that indicates the prepositional relationship.

A language may use *inflections*, or nothing may be used instead of prepositions to indicate prepositional dependency. Here we will discuss a language-free system, which represents the relation of the parts conceptually.

The CDs are intended to be used in reasoning with natural language constructs, independent of any language or the phrases in the language. This has resulted in a small number of primitive actions, about 10–12, and a set of dependencies which connect the primitive actions with each other and with their actions, objects, instruments, etc.

The CDs have two main objectives:

1. If the meaning of any two sentences is the same, they should be represented the same CD, regardless of the particular words are used in these.
2. Any information, which may be present implicitly in the sentence, should be represented explicitly in the CD.

For item 1 above, the examples are 'get' and 'receive', both will have the same CD representation. For item 2, the machine must extract the implicit part from the sentence.

The CDs have:

1. a set of primitive actions,
2. a set of states for representation and result of the action,
3. a set of dependencies, or conceptual relationships, which could exist between primitives, states, and objects.

The representation of English sentences could be constructed by joining together the building blocks to form a CD graph. The CD provides four *conceptualization primitives* using which the world of meaning is built. These are:

ACTs: Actions
PPs: Picture producers(objects)
AAs: Action Aiders (they modify the actions)
PAs: Picture Aiders (they modify the objects)

All the actions are assumed to reduce to one or more of the primitive ACTs, out of the following actions only [7]:

PROPEL: Apply physical pressure to an object (push)
MOVE: Move body parts by owner
GRASP: Grab an object by actor(grasp)
ATRANS: Transfer of relationship (give)
PTRANS: Transfer of physical location of an object (go)
INGEST: Ingest an object by an animal (eat)
EXPEL: Expel from an animal's body (cry)
MTRANS: Transfer mental information(tell)
MBUILD: Mentally make a new information(decide)
ATTEND: Focus sense organ
CONC: Conceptualize or think about an idea (think)
SPEAK: Produce sound (say)

At the conceptual level, a CD framework is responsible for representing the meaning of a piece of written language in language-free terms. The conceptualization is written on a straight line, in a conceptual dependency analysis. The dependents written perpendicular to the line are attributes of their governor, except when they are part of another conceptualization line. The whole of conceptualizations can relate to other conceptualizations as actors or attributes.

Each primitive comprises a set of slots associated with it, from the set of conceptual dependencies. Associated with each slot are restrictions as to what sorts of objects could appear in that slot. For example, the following are slots for PTRANS.

ACTOR: It is either human or animate object, that initiates the PTRANS
OBJECT: It is a physical object, that is moved (PTRANSed)
FROM: The PTRANS begins at this location
TO: PTRANS ends at this location

Figure 7.13 shows the basic roles of conceptualization primitives of CD, and Fig. 7.14 shows a general case of primitives along with an example.

The inference rules are written based on these primitives to make explicit the information which is implicitly presented in the English or any other language text. For example, using the primitive PTRANS, it is possible to make inference about the OBJECT that was PTRANSed (physically transferred), which was initially at the FROM location, and after the PTRANS is carried out, it is at the TO location. The same inferences shall be made no matter what type of PTRANS was present, like, flying, driving, walking, falling, etc.

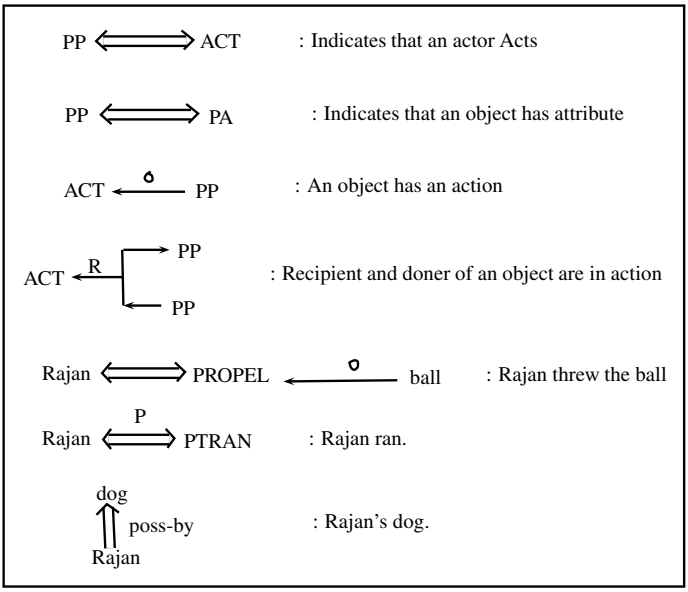

Fig. 7.13 Conceptual dependency representations

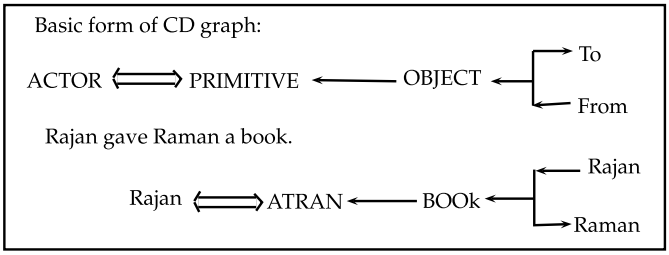

Fig. 7.14 A CD graph

7.6.1 The Parser

The CD framework can be used for natural language parsing. A CD-based system analyzes sentences into their conceptual representation by operating on pieces of every sentence, and lookup for potential conceptual cases. All the conceptualizations are checked against a list of expressions to see if that part of the concept has occurred before. Those concepts which never occurred, are meaningless in the system. Consider the sentence: "tall boy went to the park with the dog." Part of the parser output is as shown in Fig. 7.15.

In this sentence, the problem is, where to attach the concept "with dog"? Is it to the "park" or to the "tall boy"? The problem can be solved by conceptual semantics. The semantics for 'go' contains a list of conceptual prepositions. Under the preposition

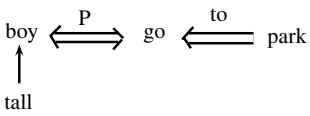

Fig. 7.15 Parser output for "tall boy went to the park"

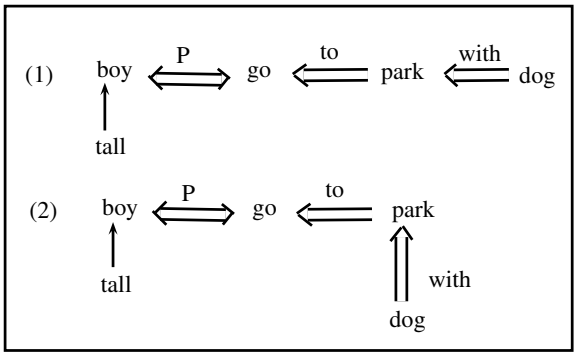

Fig. 7.16 CD for parsing a sentence: tall boy went...

'with' there is a description: "any movable physical object", and since the dog is a physical object, the dependency is applicable. As per the sentence, the parse tree (1), in Fig. 7.16 is allowed, while (2) is rejected.

The parser tries to analyze a sentence in a way analogous to human method. It handles input one word at a time as it is encountered, checks potential linkages with its own knowledge of the world and past experience, and puts its output into a language-free formalism that can be acted on.

The CD parser is a conceptual analyzer rather than syntactic parser.

Consider a sentence: "big boy gives apples to pig". The input sentence is processed word by word, and parsed into CD as shown with steps in Fig. 7.17.

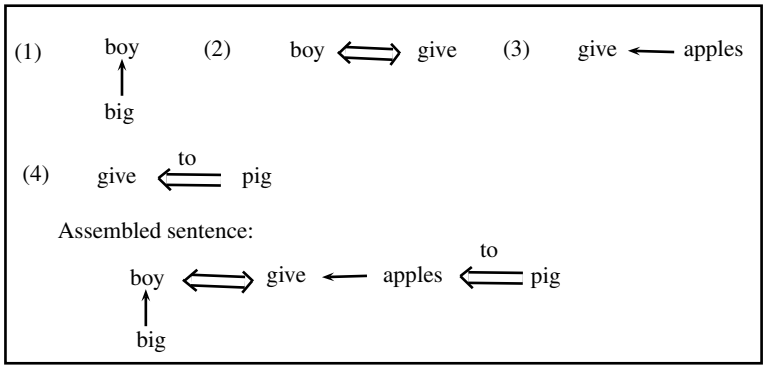

Fig. 7.17 Steps for parsing using CD

7.6.2 Conceptual Dependency and Inferences

The representation of text in canonical form has the benefit to perform inference using that text, because the canonical form allows to write inference rules as general as possible. If the representation does not capture the similarities in the meaning of the text, the rules about what can be inferred from a given text needs be duplicated by writing one rule for every form of the representation, even though they may have the same meaning relevant to the inference. To illustrate that inference is facilitated by CD, the inferences are used to build a "causal chain" to connect the events like in a story, as in the following sentence.

> Simon hit Anne. Anne's mother took Anne to the hospital. Anne's mother called Simon's mother. Simon's mother slapped Simon.

An inference system could draw many inferences through this story, e.g., 1. Simon's mother slapped Simon because she was angry at him for hitting Anne, 2. Anne was taken to the hospital because she was hurt, 3. Anne's mother called Simon's mother because she wanted to complain later. Inferences are based on a set of rules, organized around inference categories. The total number of categories in this case is 16, some of these are:

1. *Causative inferences*: These are hypothesized as possible causes or preconditions of actions.
2. *Specification inferences*: These fill in missing 'slots' in a CD primitive, such as the ACTOR or INSTRUMENT of an ACTion.
3. *Resultantive inferences*: These are inferred as likely results of the actions.
4. *Function inferences*: These are inferred as likely functions of objects.

The inference rules are applied in an undirected fashion. When a system reads a sentence, the inference rules for this are automatically applied without a goal, such as building a causal chain of events. So, it is fortuitous (having no cause or apparent cause): the inferences are applied to new representation in an undirected fashion, which some times result in the confirmation of another representation.

The undirected inferences are analogous to the spontaneous nature of inferences made by people, which some times seem uncontrollable for people. This is convincing as one cannot learn new facts without inferring things about that. However, this undirected behavior leads to problems: When processing a story, an expert system, which is based on CD, would not know which inferences are most likely to lead to building a coherent causal chain to represent the story. Hence, it would lead to a combinatorial explosion in the number of inferences that the system needs to consider to build the causal chain to represent the story. In other words, the expert system lacks the commonsense knowledge about what inferences are most likely to be relevant in a given situation. For another example,

> Simon picked up the menu. He decided on fish.

As an example, consider a situation, where one visits a restaurant, and decides to take a seat for eating (see Fig. 7.18).

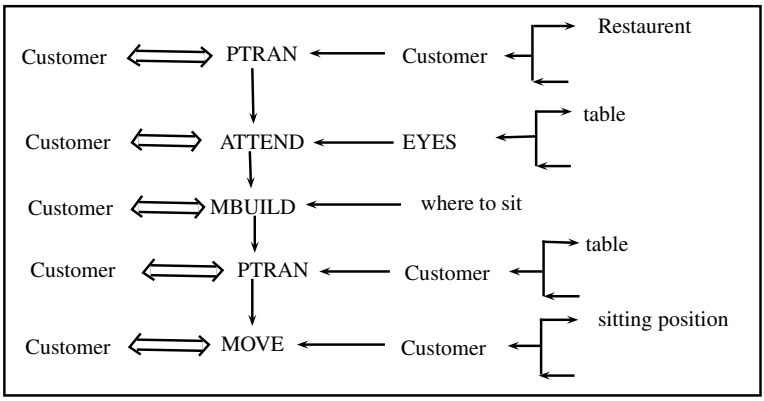

Fig. 7.18 CD for "Customer moves to Restaurant"

Once a CD with a sequence of the graph is represented, it is possible to infer many things from the representation. Like, from the Fig. 7.18, it is possible to infer many things, like: "Who went to a restaurant?", "Why the customer searched some thing in a restaurant?", etc.

7.6.3 Scripts

The scripts represent stereotypical sequences of events, like going to a restaurant, buying from a store, etc. The theory of scripts has an emphasis to understand and quick access those events that always happen in a stereotypical event sequence, without worrying about those inferences which would most likely be irrelevant.

Scripts are larger knowledge structures, used to solve problems using undirected inference. The scripts are pre-compiled sets of likely inferences, with elements of each set packed together so that they can be searched more efficiently, and produces lesser number of irrelevant inferences.

A script is a set of *roles* (participants) involved in the script, as well as common objects used. A script comprises *scenes*, such that each scene describes typical events in a portion of a script. For example, in *restaurant*'s script, roles may be *waiter*, *customer*, *restaurant* itself, and *food*. Scene may be, ENTER, ORDER, EAT, PAY, LEAVE. Each scene's details are represented as a sequence of CD representations. For example, the ENTER scene in a restaurant may consist of a causal chain as shown in Fig. 7.18, and the sequence of CDs are:

Scene-name: ENTER
C PTRAN *C* into restaurant
C ATTEND eyes to tables
C MBUlLD where to sit

C PTRANS *C* to table
C MOVE *C* to siting position

In above, *C* is a customer.

7.6.4 Conceptual Dependency Versus Semantic Nets

The conceptual dependency networks and its derivatives are often grouped into the family of semantic net representations, because the CD is a *content theory*, whereas the semantic networks are a *structure theory*. The distinction between these two types of theories lies in their emphasis: the semantic nets theory is about how the knowledge should be organized—there are nodes with arcs connecting them. There is also some general notion about the structure's semantics, i.e., how to interpret a particular semantic network. In any representation, apart from the knowledge organization, there is also a general notion about inheritance. Both of these points are about structural information. The semantic networks say nothing about what will be represented, e.g., what labels should be used for nodes and arcs, and what arcs to use where? It is only up to the user to decide about these details, and what a semantic network says is about the (structural) form that the representation will take.

The CD theory was an attempt to enumerate the types of nodes and arcs which could be used to build representations. Instead of specifying the structure of representations, the CD theory specifies the contents. Note that the basic conventions used for drawing CD graphs also specified structures, but these graphs were not really the essence of the theory. The essence of the CD was in the *primitives*, and in the types (names) of dependencies which could be used to link the primitives together.

The above highlighted distinctions between CDs and semantic nets would become clear if one thinks of trying to implement the CDs and semantic nets in first-order predicate logic (FOPL). In CDs, it is not difficult to imagine that the primitives: ACTs, dependencies, and states would specify a set of predicates to be used when one writes the predicate calculus statements to represent sentences. There will be a need to adopt some translation conventions, in order to make all assertions of FOPL type, but these would be quite straightforward.

But, it does not make a sense to implement a semantic net in FOPL, as the two representations are in competition with each other: each representation provides a different syntax for distinguishing between predicates, arguments, and relations. There would be nothing left to semantic nets, if they are translated into predicate logic. Putting it another way, the semantic nets would not add anything to FOPL. This statement is in contrast to CD, which adds the CD primitives to the predicates used, as and when needed.

7.7 Summary

In a network-based representation, the pieces of knowledge are clustered together into coherent semantic groups. It provides a more natural way of mapping knowledge between the natural language and these networks. A semantic network has *ako* (a kind of), has-parts, color, and has-property are binary relations. Semantic networks have primitives, and inference in the semantic networks is provided through inheritance. The semantic networks are based on the associationist theory, which defines the meaning of an object in terms of a network of associations with other objects.

A natural language understanding program must understand the *intentions*, *beliefs*, *hypothetical reasoning*, *plans*, and *goals*. Conceptual Graphs (CG) portray mental perception which consists of basic primitive concepts and relationships which exist between them. In CG, a sentence is represented as a verb node, with various case links to the node representing other participants in the action.

A *frame* can be viewed as generalized *semantic network*. In a frame, there is stress on instances or classes, rather than nodes, and on slots and their values instead of links and connections. Inheritance moves the default values in frames from classes to instances through activation of the appropriate when-constructed procedure. Frames represent some *stereotypical situation*, like, a classroom, house, a machine with various parts, etc, and have slots, which may hold values, or procedures. Procedures in frames may be sleeping procedures (daemons) and may be awakened when required. Frames have the language to describe them. The Frames have applications in machine vision.

A Description Logic models *concepts*, *roles* and *individuals*, and their relationships. The fundamental modeling concept of a $\mathscr{DL}$ is the axiom: "a logical statement relating roles and/or concepts". It is a family of formal knowledge representation languages, which is more expressive than propositional logic and has more efficient decision properties than first-order predicate logic. It is used in formal reasoning on the concepts of an application domain.

The Conceptual Dependency (CD) framework is a stratified linguistic system that attempts to provide a computational theory of simulative performance. Every conceptualization must have a two-way dependency link. The CDs are intended to be used in reasoning with natural language constructs, independent of any language or the phrases in the language. The CDs have a set of primitive actions, a set of states for representation and result of the action, and a set of dependencies, or conceptual relationships, which could exist between primitives, and states. An English language sentence can be parsed into CDs. The inference using CDs is unambiguous.

Scripts represent stereotypical sequences of events, such as going to a restaurant. By using the script, the theory was that the understand had quick access to those events which always happen in a stereotypical event sequence.

Exercises

1. Explain the difference between Ontologies and Semantic networks.
2. Describe the logical, structural, semantic, and procedural parts of semantic networks.
3. There are many words in the English language which can be used as noun and verb, for example, "book" in "Book my ticket" and "This is my book" have used the word "book" as verb and noun, respectively. In the following words, what are their different parts of speech?
   milk, house, liquid, airborne, group, set.
   Suggest a method in each case, as to how you will reason the true meaning.
4. Suggest a data structure for the implementation of semantic nets such that retrieval can be as fast as possible.
5. Represent the relationships between quadrangle, parallelogram, rhombus, rectangle, and square in the form of a semantic network. Is the semantic network unique, or are there many different forms it can take?
6. Represent the following statements using semantic networks:

   a. "Rajan teaches his students a lot of innovative things."
   b. "Raman tells Rajan's students a number of useful things."
   c. Mike and Mary's telephone number is the same.
   d. John believes that Mike and Mary's telephone number is the same.

7. Represent the following knowledge in a semantic network:

Dogs are Mammals	Birds have Wings
Mammals are Animals	Bats have Wings
Birds are Animals	Bats are Mammals
Fish are Animals	Dogs chase Cats
Worms are Animals	Cats eat Fish
Cats are Mammals	Birds eat Worms
Cats have Fur	Fish eat Worms

8. Represent the following in partitioned semantic networks:

   a. Every player kicked a ball.
   b. All players like the referee.
   c. Andrew believes that there is a fish with lungs.

9. Represent the following statements using semantic networks:

   a. "John tells his students a lot of useful things."
   b. "Andrea tells John's students an enormous number of useful things."

   Suppose you wanted to build an AI system that was able to work out "who tells John's students the greatest number of useful things." How could you do that?
10. Suppose you learn that "Tom is a cat". What additional knowledge about Tom can be derived from your representation? Explain how.

11. Suppose Tom is unlike most cats and does not eat fish. How could one deal with this in the semantic network?
12. Formulate the solutions as to how the semantic networks can be used in the following cases?

    a. Natural language understanding
    b. Information retrieval
    c. Natural language translation
    d. Learning systems
    e. Computer vision
    f. Speech generation system

13. "The inferencing in semantic networks make use of unification, chaining, modus ponens, and resolution." Justify each, taking a suitable example.
14. Explain, using semantic networks, how we can map an object's perception to concepts, and identify these concepts. Give examples.
15. How semantic networks help in understanding the meaning of words in natural language sentences? Explain.
16. Represent the following as a series of frames:

Dogs are Mammals	Birds have Wings
Mammals are Animals	Bats have Wings
Birds are Animals	Bats are Mammals
Fish are Animals	Dogs chase Cats
Worms are Animals	Cats eat Fish
Cats are Mammals	Birds eat Worms
Cats have Fur	Fish eat Worms

17. Express the following sentences in Description Logic:

    a. All employees are humans.
    b. A mother is a female who has a child.
    c. A parent is a mother or a father.
    d. A grandmother is a mother who has a child who is a parent.
    e. Only humans have children that are humans.

18. Translate the logic expressed in Fig. 7.10 into $\mathscr{DL}$.
19. Select one or more answers from the following. Also, justify the answer(s) selected by you.

    a. What type of reasoning is performed using semantic networks?

    (A) Deductive (B) Default
    (C) Inductive (D) Abductive
    (E) Hierarchical

    b. In the Description Logic, the domain is always,

    (A) Open world (B) Closed world
    (C) Depends on the domain used (D) None of above

References

1. Collins AM, Quillian MR (1969) Retrieval time from semantic memory. J Verbal Learn Verbal Behav 8(2):240–247
2. Deliyanni A, Kowalski RA (1979) Logic and semantic networks. Commun ACM 22(3):184–192
3. Faucher C (2001) Easy definition of new facets in the frame-based language Objlog+. Data Knowl Eng 38:223–263
4. Harmelen FV et al (2008) Handbook of knowledge representation. Elsevier, pp 213–237
5. https://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-02.pdf. Accessed 19 Dec 2017
6. http://www.jfsowa.com/pubs/semnet.htm. Accessed 12 Feb 2018
7. Lytinen SL (1992) Conceptual dependency and its descendants. Comput Math Appl 23(2–5):51–73 Pergamon Press
8. Minsky M (1974) A framework for representing knowledge. MIT-AI Laboratory Memo-306
9. Pike R, Kehler T (1968) The role of frame-based representation in reasoning. Commun ACM 28(9):904–920
10. Quillian MR (1967) Word concepts: a theory and simulation of some basic semantic capabilities. Behav Sci 12(5):410–443
11. Quillian MR (1968) Semantic information processing. Cambridge, Mass., MIT Press, pp 216–270
12. Simmons RF (1973) Semantic networks: their computation and use for understanding English sentences. In: Schank RC, Colby KM (eds) Computer models of thought and language. W.H. Freeman and Co, San Francisco, CA
13. Simmons RF, Chester D (1977) Inferences in quantified semantic networks. Proceedings of the fifth international joint conference on artificial intelligence. MIT, pp 267–273
14. Sowa J (1976) Conceptual graphs for a data base interface. IBM J Res Develop 336–355