

# **MODULE 3      ARTIFICIAL INTELLIGENCE TECHNIQUES IN PROGRAMMING AND NATURAL LANGUAGES**

## **UNIT 1      KNOWLEDGE REPRESENTATION**

### **CONTENTS**

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
  - 3.1 Overview of Knowledge Representation
    - 3.1.1 Characteristics
    - 3.1.2 History of Knowledge Representation and Reasoning
  - 3.2 Knowledge Representation Languages
  - 3.3 Domain Modeling
  - 3.4 Ontological Analysis
  - 3.5 Classic
    - 3.5.1 The Classic Language
    - 3.5.2 Enhancements to Classic
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Reading

### **1.0 INTRODUCTION**

Knowledge Representation (KR) has long been considered one of the principal elements of Artificial Intelligence, and a critical part of all problems solving [Newell, 1982]. The subfields of KR range from the purely philosophical aspects of epistemology to the more practical problems of handling huge amounts of data. This diversity is unified by the central problem of encoding human knowledge - in all its various forms - in such a way that the knowledge can be used. This goal is perhaps best summarized in the *Knowledge Representation Hypothesis*:

Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantically attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge [Smith, 1982].

A successful representation of some knowledge must, then, be in a form that is *understandable* by humans, and must cause the system using the knowledge to *behave* as if it knows it. The "structural ingredients" that accomplish these goals are typically found in the *languages* for KR, both implemented and theoretical, that have been developed over the years.

## 2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain the meaning of knowledge representation
- describe the history of history of knowledge representation and reasoning
- list some characteristics of kr
- list 4 main features of kr language.

## 3.0 MAIN CONTENT

### 3.1 Overview of Knowledge Representation

Knowledge Representation (KR) research involves analysis of how to accurately and effectively reason and how best to use a set of symbols to represent a set of facts within a knowledge domain. A symbol vocabulary and a system of logic are combined to enable inferences about elements in the KR to create new KR sentences. Logic is used to supply formal semantics of how reasoning functions should be applied to the symbols in the KR system. Logic is also used to define how operators can process and reshape the knowledge. Examples of operators and operations include negation, conjunction, adverbs, adjectives, quantifiers and modal operators. Interpretation theory is this logic. These elements--symbols, operators, and interpretation theory-- are what give sequences of symbols meaning within a KR.

A key parameter in choosing or creating a KR is its *expressivity*. The more expressive a KR, the easier and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR. However, more expressive languages are likely to require more complex logic and algorithms to construct equivalent inferences. A highly expressive KR is also less likely to be complete and consistent.

Less expressive KRs may be both complete and consistent. Auto epistemic temporal modal logic is a highly expressive KR system, encompassing meaningful chunks of knowledge with brief, simple symbol sequences (sentences). Propositional logic is much less expressive but highly consistent and complete and can efficiently produce inferences with minimal algorithm complexity. Nonetheless, only the limitations of an underlying knowledge base affect the ease with which inferences may ultimately be made (once the appropriate KR has been found). This is because a knowledge set may be exported from a knowledge model or knowledge base system (KBS) into different KRs, with different degrees of expressiveness, completeness, and consistency. If a particular KR is inadequate in some way, that set of problematic KR elements may be transformed by importing them into a KBS, modified and operated on to eliminate the problematic elements or augmented with additional knowledge imported from other sources, and then exported into a different, more appropriate KR.

In applying KR systems to practical problems, the complexity of the problem may exceed the resource constraints or the capabilities of the KR system. Recent developments in KR include the concept of the Semantic Web, and development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Mark-up Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

There are several KR techniques such as frames, rules, tagging, and semantic networks which originated in Cognitive Science. Since knowledge is used to achieve intelligent behaviour, the fundamental goal of knowledge representation is to facilitate reasoning, drawing conclusions. A good KR must be both declarative and procedural knowledge. What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand:

A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.

It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?

It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.

It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences. It is a medium of human expression, i.e., a language in which we say things about the world."

Some issues that arise in knowledge representation from an AI perspective are:

How do people represent knowledge?

What is the nature of knowledge?

Should a representation scheme deal with a particular domain or should it be general purpose?

How expressive is a representation scheme or formal language?

Should the scheme be declarative or procedural?

There has been very little top-down discussion of the knowledge representation (KR) issues and research in this area is a well aged quillwork. There are well known problems such as "spreading activation" (this is a problem in navigating a network of nodes), "subsumption" (this is concerned with selective inheritance; e.g. an ATV can be thought of as a specialization of a car but it inherits only particular characteristics) and "classification." For example a tomato could be classified both as a fruit and a vegetable.

In the field of artificial intelligence, problem solving can be simplified by an appropriate choice of *knowledge representation*. Representing knowledge in some ways makes certain problems easier to solve. For example, it is easier to divide numbers represented in Hindu-Arabic numerals than numbers represented as Roman numerals.

### 3.1.1 Characteristics

A good knowledge representation covers six basic characteristics:

Coverage, which means the KR covers a breadth and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.

Understandable by humans. KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (Polar bears are bears, which are animals).

It should also have simple primitives that combine in complex forms.

Consistency. If John closed the door, it can also be interpreted as the door was closed by John. By being consistent, the KR can eliminate redundant or conflicting knowledge.

Efficient

Easiness for modifying and updating.

Supports the intelligent activity which uses the knowledge base

To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopaedia (e.g. Wikipedia) is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

### **3.1.2 History of Knowledge Representation and Reasoning**

In computer science, particularly artificial intelligence, a number of representations have been devised to structure information.

KR is most commonly used to refer to representations intended for processing by modern computers, and in particular, for representations consisting of explicit objects (the class of all elephants, or Clyde a certain individual), and of assertions or claims about them ('Clyde is an elephant', or 'all elephants are grey'). Representing knowledge in such explicit form enables computers to draw conclusions from knowledge already stored ('Clyde is grey').

Many KR methods were tried in the 1970s and early 1980s, such as heuristic question-answering, neural networks, theorem proving, and expert systems, with varying success. Medical diagnosis (e.g., Mycin) was a major application area, as were games such as chess.

In the 1980s formal computer knowledge representation languages and systems arose. Major projects attempted to encode wide bodies of general knowledge; for example the "Cyc" project (still ongoing) went through a large encyclopaedia, encoding not the information itself, but the information a reader would need in order to understand the encyclopaedia: naive physics; notions of time, causality, motivation; commonplace objects and classes of objects.

Through such work, the difficulty of KR came to be better appreciated. In computational linguistics, meanwhile, much larger databases of language information were being built, and these, along with great increases in computer speed and capacity, made deeper KR more feasible.

Several programming languages have been developed that are oriented to KR. Prolog developed in 1972, but popularized much later, represents propositions and basic logic, and can derive conclusions from known premises. KL-ONE (1980s) is more specifically aimed at knowledge representation itself. In 1995, the Dublin Core standard of metadata was conceived.

In the electronic document world, languages were being developed to represent the structure of documents, such as SGML (from which HTML descended) and later XML. These facilitated information retrieval and data mining efforts, which have in recent years begun to relate to knowledge representation. Development of the Semantic Web, has included development of XML-based knowledge representation languages and standards, including RDF, RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

### **3.2 Knowledge Representation Languages**

William Woods defines the properties of a KR Language as follows:


A KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning [Woods, 1975].

Wood's definition is merely a simplification of the KR Hypothesis where "reasoning" is the only method of "engendering the behavior that manifests that knowledge." Reasoning is essential to KR, and especially to KR languages, yet even simple reasoning capabilities can lead to serious tractability problems [Brachman and Levesque, 1987], and thus must be well understood and used carefully.

One of the most important developments in the application of KR in the past 20 years has been the proposal [Minsky, 1981], study [Woods, 1975] [Brachman, 1977] [Brachman, 1979], and development [Brachman and Schmolze, 1985] [Fox, Wright, and Adam, 1985] [Bobrow and Winograd, 1985] of frame-based KR languages.

While frame-based KR languages differ in varying degrees from each other, the central tenet of these systems is a notation based on the specification of objects (concepts) and their relationships to each other. The main features of such a language are:

1. *Object-orientedness*. All the information about a specific concept is stored with that concept, as opposed, for example, to rule-based systems where information about one concept may be scattered throughout the rule base.
2. *Generalization/Specialization*. Long recognized as a key aspect of human cognition [Minsky, 1981], KR languages provide a natural way to group concepts in hierarchies in which higher level concepts represent more general, shared attributes of the concepts below.
3. *Reasoning*. The ability to state in a formal way that the existence of some piece of knowledge implies the existence of some other, previously unknown piece of knowledge is important to KR. Each KR language provides a different approach to reasoning.
4. *Classification*. Given an abstract description of a concept, most KR languages provide the ability to determine if a concept fits that description, this is actually a common special form of reasoning.



```
article-10::  
  title: "Domain Modeling"  
  author: person-1  
  published-in: journal-64  
journal-64::  
  name: "AI Magazine"  
  volume: 10  
  number: 4  
  location: Library
```

Figure A A simple database representation.

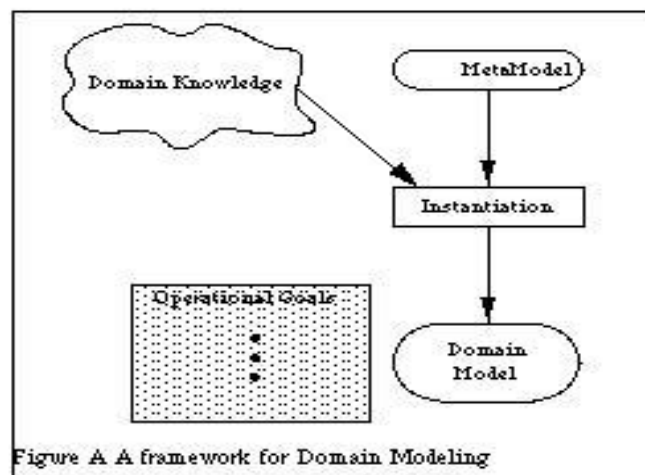
Object orientation and generalization help to make the represented knowledge more understandable to humans; reasoning and classification help make a system behave as if it knows what is represented. Frame-based systems thus meet the goals of the KR Hypothesis.

It is important to realize both the capabilities and limitations of frame-based representations, especially as compared to other formalisms. To begin with, all symbolic KR techniques are derived in one way or another from First Order Logic (FOL), and as a result are suited for representing knowledge that doesn't change. (Figure one Simple database representation) .

Different KR systems may be able to deal with non-monotonic changes in the knowledge being represented, but the basic assumption has been that change, if present, is the exception rather than the rule.

Two other major declarative KR formalisms are *production systems* and *database systems*. Production systems allow for the simple and natural expression of *if-then rules*. However, these systems have been shown to be quite restrictive when applied to large problems, as there is no ordering of the rules, and inferences cannot be constrained away from those dealing only with the objects of interest. Production systems are subsumed by frame-based systems, which additionally provide natural inference capabilities like classification and inheritance, as well as knowledge-structuring techniques such as generalization and object orientation.

Database systems provide only for the representation of simple assertions, without inference. Rules of inference are important pieces of knowledge about a domain. For example, consider the bibliographic database in Figure 1. If someone were interested in `article-10` and wanted to know where it was, that person would have to be smart enough to realize that *an article can be found in the location of the journal in which it is published*. That sentence is a rule, it is *knowledge*. It is knowledge that cannot be expressed in a database system. The person doing the retrieval of the information in the database must have that knowledge in order to type the SQL statement that will get the proper location. In a frame-based system that knowledge can be expressed as a rule that will fire when `article-10` is accessed, thus the user is not required to know it.





Frame-based systems are currently severely limited when dealing with *procedural knowledge* [Winograd, 1975]. An example of procedural knowledge would be Newton's Law of Gravitation - the attraction between two masses is inversely proportional to the square of their distances from each other. Given two frames representing the two bodies, with slots holding their positions and mass, the value of the gravitational attraction between them cannot be (Figure 2-A framework for Domain Modelling).

Inferred declaratively using the standard reasoning mechanisms available in frame-based KR languages, though a function or procedure in a programming language could represent the mechanism for performing this "inference" quite well. Frame-based systems that can deal with this kind of knowledge do so by adding a procedural language to its representation. The knowledge is *not* being represented in a frame-based way, it is being represented as C or (more commonly) LISP code which is accessed through a slot in the frame [Bobrow and Winograd, 1985]. This is an important distinction - there is knowledge being.

Encoded in those LISP functions that is not fully accessible. The system can reason *with* that knowledge, but not *about* it; in other words we can use some attached procedure to compute (or infer) the value of one slot based on some others, but we cannot ask how that value was obtained.

### 3.3 Domain Modeling

Domain modeling is the field in which the application of KR to specific domains is studied and performed. Figure 2 shows a framework for discussing domain modeling that seems to map well onto most examples [Iscoe, Tam, and Liu, 1991].

The amorphous shape labelled *Domain Knowledge* refers to the knowledge possessed by the domain expert that must be encoded in some fashion. This knowledge is not well defined and is fairly difficult for others to access. The box labelled *Meta-Model* refers to the KR formalism, typically a KR language that will be used as the *symbol level* [Newell, 1982] for the machine representation of this knowledge. The box labelled *instantiation* refers to the *process* of taking the domain knowledge and physically representing it using the meta-model, this process is sometimes referred to as *knowledge acquisition* [Schoen, 1991]. The box labelled *domain model* refers to the knowledge-base that results from the instantiation, and the *operational goals* are typically not represented formally, but refer to the reason the domain model was built and what it will be used for.

Specific examples of real-world domain modelling efforts and how they fit into this framework can be found in [Iscoe, 1991], and it has become clear that the most prevalent operational goal across modelling efforts today is understanding the domain of a large software system [Arango, 1989]. One thing that seems to be universally lacking in efforts with this operational goal is the realization that a software system operating within a domain *is a part of that domain*, and deserves as much attention and detail in the model as any other part. The main reason for this oversight is that there is a historical reason for distinguishing *procedural* from *declarative* knowledge [Winograd, 1975], and as a result the two are typically represented differently: domain models are represented with frame based KR languages and programs are represented with programming languages.

This traditional separation between programs and domain models causes problems during the instantiation of a domain model that includes not only knowledge of the objects and attributes, but knowledge of the procedural aspects of the processes associated with the domain as well. The problems stem from the fact that domain modelling is a discipline in which advances are made incrementally, by building upon previous systems [Simon, 1991]. Some of the most significant results are in the form of methodologies which help other domain modellers to avoid pitfalls and use techniques that work [Gruber, 1993].

The predominant methodologies for domain modelling clearly indicate that the instantiation of the model is the most time consuming part, and that the most important part of instantiation is *ontological analysis* [Alexander, Freiling, and Shulman, 1986] (which is more fully described in the next section). Ontologies for general taxonomies of objects are abundant, and there seem to be clear guidelines for developing new ones.

The problem is that for the knowledge required to represent procedural knowledge and reason about it (not with it); there are few guidelines, especially when the procedures requiring representation are implemented as software. There is not much background to draw upon, other than software information systems, as far as ontologies and methodologies for modelling what software does. Ontological analysis ended up being a large part of the effort for this research, since it had never been done before.

### 3.4 Ontological Analysis

The word *ontology* means "the study of the state of being." *Ontology* describes the states of being of a particular set of things.

This description is usually made up of axioms that define each thing. In knowledge representation, ontology has become the defining term for the part of a domain model that excludes the *instances*, yet describes what they can be. Ontological analysis is the process of defining this part of the model.

What makes up a specific domain ontology is restricted by the representational capabilities of the *meta-model* - the language used to construct the model. Each knowledge representation language differs in its manner and range of expression. In general, ontology consists of three parts: *concept* definitions, *role* definitions, and further inference definitions.

The concept definitions set up all the *types* of objects in the domain. In object oriented terms this is called the class definitions, and in database terms these are the entities. There can be three parts to the concept definitions:

- Concept taxonomy. The taxonomy is common to most knowledge representation languages, and through it is specified the nature of the categories in terms of generalization and specialization.

- Role defaults which specify for each concept what the default values are for any attributes.

- Role restrictions which specify for a concept any constraints on the values in a role, such as what types the values must be, how many values there can be, etc.

A role is an attribute of an object. In object-oriented terms it is a slot, in database terms (and even some KR languages) it is a relation. In the simplest case, a role for an object just has a value; the object mailbox-4 might have a role number-of-messages, for example, that would have a value which is a number. Roles also express relationships between objects. The same object might have a role called owner which relates mailbox-4 to the object person-2. Roles which represent relationships are unidirectional. A role definition may have up to three parts as well:

- The role taxonomy which specifies the generalization/specialization relationship *between* roles. For example, ontology for describing cars might include roles called has-engine, has-seats, and has-headlights, which relate objects that represent cars to objects that represent engines, seats, and headlights, resp. The role has-parts, then, could be expressed as the generalization of all these roles, and the result is that all the values of all the more specialized roles would also be values of the more general role.

Role inverses which provide a form of inference that allows the addition of a role in the opposite direction when the forward link is made. For example, if the inverse of has-engine was engine-of, then when the has-engine link between the object that represents the car and the object that represents the engine is made, the engine-of link will automatically be added between the engine object and the car object.

Role restrictions. The role itself may be defined such that it can only appear between objects of certain types (domain/range restrictions), or can only appear a specified number of times (cardinality restriction). This is the same information specified in role restriction for concepts, some representation languages consider this information to be part of the role, and some consider it to be part of the concept.

The final part of ontology is the specification of additional inference that the language provides. Examples of this are forward and/or backward chaining rules, path grammars, subsumption and/or classification, demons, etc. An explanation of the inference mechanisms used in this research will be given in the next section.

### 3.5 Classic

Classic is a frame-based knowledge representation language that belongs to the family of *description logics* [Brachman, et al., 1991]. It is descended from KL-ONE [Brachman and Schmolze, 1985], and has been specifically designed to be mindful of the tractability of its own inferences [Brachman, et al., 1989].

Classic knowledge-bases are composed of four kinds of objects: *concepts*, *roles*, *rules*, and *individuals*. Ontology in Classic consists of concept taxonomy, role taxonomy, role inverses, role restrictions and defaults. The role restrictions and defaults are specified as part of the concept definitions. Classic rules are forward chaining rules.

#### 3.5.1 The Classic Language

Throughout this document, it has been necessary to make explicit use of the notation and terminology of Classic to explain and describe some of the more detailed aspects of this research. This section contains a brief introduction to the language of Classic in order to make it clear precisely how one goes about describing objects. This introduction only presents the subset of Classic which was used in this research. The Classic language is specifically designed to make it possible to describe objects in such a way that it is possible to determine automatically whether one object is subsumed by another. The peculiarities of the language arise from the goal of making this determination not only possible, but tractable.

To begin with, a typical concept description looks like this:

```
(defconcept information-filter
  (and kbeds-object
    (all information-filter-of valid-mail-recipient)
    (at-least one information-filter-of)
    (all has-filter-condition kbeds-mail-message)
    (at-least one has-filter-condition)
    (at-most one has-filter-condition)
    (all has-filter-action kbeds-filter-action)
    (at-least one has-filter-action))
  :disjoint kbeds-thing)
```

This says an `information-filter` is subsumed by (or is more specialized than, or is a subclass of) `kbeds-object` (and therefore is also described by that concept), and that all the *fillers* (in Classic a filler is the value of a particular role on a particular object) for its `information-filter-of` role must be individuals of `valid-mail-recipient`, and that there must be at least one filler for that role, and similarly for the role `has-filter-condition` except that there can also be at most one filler (in other words, there can be only one filler), and so on. The disjoint specification identifies this concept as a member of a disjoint set, which means that an individual cannot be subsumed by more than one concept in that set. The most obvious example of a disjoint set would be the gender set, containing the concepts "male person" and "female person."

In the terminology of Classic, the description above is *told* information. Told information is precisely the information that is explicitly typed into the knowledge base. This is opposed to *derived* information, which is all the information that Classic derives or infers through its various mechanisms. (Figure 3 – A Simple taxonomy of primitive concepts).

This description actually shows a *primitive* concept - one which Classic will not automatically try to classify. Classic will also not automatically try to find which individuals are subsumed by a primitive concept, this information must be told. This subtle notion may seem irrelevant, but it is the norm in most representation languages (when a concept is created the modeller explicitly names the parent concepts, and when an individual is created, the modeller explicitly names the concept that the new individual is an instance of).

It is important in Classic because there is another kind of concept, the *defined* concept, which Classic actually does automatically classify and find individuals of. For example, in figure 3 a simple taxonomy of primitive concepts is shown. Let us suppose we create a defined concept called vegetarian-mammal as follows: (and mammal (all food plant)). Next we create another defined concept called fruit-eating-person: (and person (all food fruit)).

Classic will derive that vegetarian-mammal *subsumes*

fruit-eating-person

(why?

because mammal subsumes person and plant subsumes fruit). If we created two individuals, joe and apple, and tell Classic that they are

instances of person and fruit,

resp., and further tell Classic that

apple is a filler for joe's food

role, Classic will derive that joe is

an instance of fruit-eating-

person (and therefore also

vegetarian-mammal). Again, Classic will never derive that an individual is an instance of a primitive concept, it must always be told that.

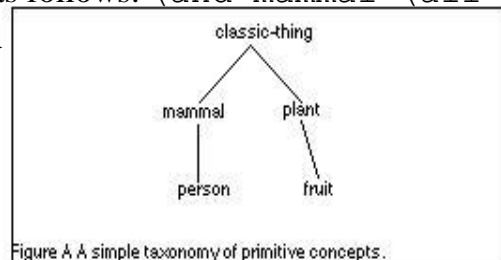


Figure A A simple taxonomy of primitive concepts.

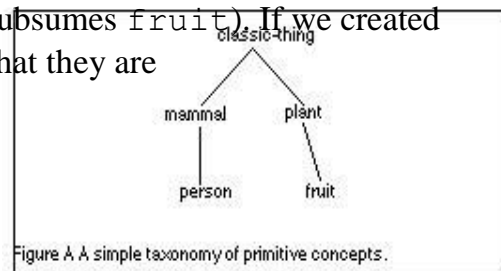


Figure A A simple taxonomy of primitive concepts.

This automatic classification of individuals of defined concepts through subsumption is a simple, yet extremely powerful process. It is the key to several significant advances in software information systems described in later sections.

Another important point about Classic is the *Open World Assumption*. Classic does not assume that the information it knows is all the information there is. Returning to the example above, we have the following *told* information about two individuals, joe: (and person (fills food apple)), and apple: fruit. Classic will then add all the *derived* information it can to these individuals, yielding joe:

(and person mammal classic-thing (fills food apple)), and apple: (and fruit plant classic-

thing). Where is the information about joe being a fruit-eating-person?

The truth is that Classic cannot derive this yet. The definition of fruit-eating-person specifies that *all* the fillers for

the `food` role of an individual must be instances of `fruit`. Classic does not assume that because it knows one (or two, or zero, etc.) fillers for an individual's role, that it knows them all. In fact, Classic assumes the opposite: it assumes it does not know them all.

There are two ways for Classic to figure out that it knows all the fillers for an individual's role. The first way is for it to be told, by *closing* the role. When a role is closed on an individual, it tells Classic that there can be no more filler. In the above example, the user would have to close the `food` role on `joe` in order for Classic to derive that `joe` is an instance of `fruit-eating-person` (Classic always tries to reclassify individuals when their roles are closed). The second way is for Classic to derive that a role is closed on an individual if there is an *at-most* restriction on the role. For example, if the concept `person` (or `mammal`) additionally had `(at-most one food)` in its description, then since `joe` is told to be an instance of `person` that restriction would apply to him, and since he already has one filler in his `food` role, *and* since he can have at most one filler in his `food` role, he can have no more and the role is derived to be closed.

The final part of ontology in Classic is the rules. Classic rules come in two forms, *description* rules and *filler* rules. All classic rules have as their antecedent a named concept, and are fired on an individual when the individual is classified as an instance of the concept.

The consequent of a classic description rule is a classic description which, when the rule fires on an individual, is merged into the description of the individual. For example, if we had a description rule like: `vegetarian-mammal --> (at-most 0 has-prey)`, the rule would fire on `joe` when he is classified as a `fruit-eating-person` and would add the *at-most* restriction to the description of `joe`. Classic would then also derive that `joe's has-prey` role is closed as well.

The consequent of a classic filler rule is the name of a role and a LISP function that will be invoked when the rule fires. The function is passed the individual the rule fired on and the role named in the consequent, and returns a list of new fillers for that role and individual. One useful application for filler rules is to create inter-role dependencies. For example, the concept `rectangle` has three roles: `length`, `width`, and `area`. We could define a function for calculating the area in LISP as follows:

```
(defun calculate-area (rect role)
```

```
(let ((length (car (cl-fillers rect @length)))
      (width (car (cl-fillers rect @width))))
  (* length width)))
```

And then define a filler rule: *rectangle --> area calculate-area*, the filler for the *area* role would automatically be generated based on the fillers in the *length* and *width* roles.

### 3.5.2 Enhancements to Classic

It was necessary to extend Classic in several ways in order to support this research. Each extension had a different motivation, which may not be entirely clear until that aspect of the research is discussed. These extensions are explained here, however, so that the sections involving the research do not need to sidetrack into explanations of the underlying support.

The first extension to Classic was a facility for supporting what some other representation languages call *path grammars* or *role transitivity* [Fox, Wright, and Adam, 1985]. A very common form of inference in frame-based representations is one in which the fillers for one role in a class of individuals can always be found by following the same *role path*. For example, an individual representing an article in a journal might have a role called *published-in* which is filled with an individual representing a journal. Journal individuals could have a role called *location* which is filled with some string indicating the place where the journal is physically located. It makes sense that the article individual should also have a location that is *the same as the location of the journal it is published in*, and this can be represented as a path rule. A path rule, like all Classic rules, has for its antecedent a concept name, and for its consequent a role and a role path (an ordered list of roles). When a rule fires on an individual, classic follows the role path to the end, and fills the role specified in the rule with all the values it finds at the end of the path. In the journal article example, the path rule would be: *article --> location (published-in location)*. An individual of *article* might be described: `(and article (fills published-in journal-10))`, and the *journal* individual *journal-10*: `(and journal (fills location "Shelf 10"))`. Classic would fire the path rule on the individual of *article* and follow the path: the first part of the path is *published-in*, which gets us to *article-10*, and the next part of the path is *location* which gets us to the string `"Shelf 10."` This value is then derived to be the filler for the *location* role of the individual of *article*. If a particular path ends "early," that is, the path leads to an intermediate individual that has no fillers for the next role in the path, no fillers are returned for that particular branch of the path.



The path rule facility was further expanded to allow for the expression of *specialization overrides*. A specialization override is a type of inference in which a value that would normally be derived to fill a role is blocked if and only if there is already a value filling the role that is more specialized. The most common example of this is in object-oriented languages, a class inherits all the methods of its superclass, except the ones that are already defined by the class.

The next enhancement to Classic was a facility for dumping individuals into a file; in essence there is no way to save the current state of a classic knowledge-base. While this may not sound like a significant extension, there is one aspect of dumping (or, more accurately, of loading) a knowledge-base that is very intricate: the order in which roles are closed. When a role is told to be closed on an individual, it means there can be no more filler for that role - told or derived. However, when derived role filler depends on filler or fillers in other individuals, the role cannot be closed until the fillers it depends on are closed. There is an implicit ordering of role closing based on all the ways Classic can derive information.

The most significant enhancement to Classic was the addition of a facility for representing spanning objects [Welty and Ferrucci, 1994]. In reality, this enhancement is in the process of being made to Classic by its support team, and the spanning object facility used for this research was actually applied to the "dumped" knowledge-based - that is, the spanning functions worked by generating a text file containing Classic descriptions, then the knowledge-base was cleared and the text file could be loaded in. Until support for multiple universes of discourse is added to Classic (which will happen in the next major release), this was the only way to proceed. The only limitation this presented was an inability to change the first universe from the second.

## 4.0 CONCLUSION

Knowledge representation (KR) is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inferencing from those knowledge elements, creating new elements of knowledge. The KR can be made to be independent of the underlying knowledge model or knowledge base system (KBS) such as a semantic network.

## 5.0 SUMMARY

In this unit, you learnt that:

Knowledge Representation (KR) research involves analysis of how to accurately and effectively reason and how best to use a set of symbols to represent a set of facts within a knowledge domain.

A good knowledge representation covers six basic characteristics

In computer science, particularly artificial intelligence, a number of representations have been devised to structure information.

A KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning [Woods, 1975].

Classic is a frame-based knowledge representation language that belongs to the family of *description logics* [Brachman, et al., 1991]

## 6.0 TUTOR-MARKED ASSIGNMENT

1. List four (4) Characteristics of Knowledge representation.
2. Explain Knowledge representation.
3. Explain Domain Modeling in KR.

## 7.0 REFERENCES/FURTHER READING

Walker, A. McCord, M., John, F. Sowa, & Walter, G. Wilson (1990). *Knowledge Systems and Prolog* (Second Edition). Addison-Wesley.

Arthur, B. M. (1998). *Knowledge Representation*. Lawrence Erlbaum Associates.

Chein, M. ; Mugnier, M.L. (2009). *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer. ISBN 978-1-84800-285-2.

Davis, R.; Shrobe, H. E. Representing Structure and Behavior of Digital Hardware, IEEE Computer, Special Issue on Knowledge Representation, 16(10):75-82.

Hermann Helbig K. (2006). *knowledge Representation and the Semantics of Natural Language*. Springer, Berlin, Heidelberg, New York.

- Jean-Luc, Hainaut, Jean-Marc, H.; Vincent Englebert, Henrard, J., Roland, D.  
Understanding Implementations of IS-A Relations. ER 1996: 42-57.
- Jeen Broekstra, Michel Klein, Stefan Deckerc, Dieter Fenselb, Frank van Harmelenb and  
Ian Horrocks Enabling knowledge representation on the Web by extending RDF  
Schema, , April 16 2002.
- John F. Sowa (2000). *Knowledge Representation: Logical, Philosophical, and  
Computational Foundations*. New York: Brooks/Cole.
- Philippe Martin "Knowledge representation in RDF/XML, KIF, Frame-CG and  
Formalized-English", Distributed System Technology Centre, QLD, Australia, July  
15-19, 2002
- Pople H, Heuristic Methods for imposing structure on ill-structured problems, in AI in  
Medicine, Szolovits (ed.). AAAS Symposium 51, Boulder: Westview Press.
- Randall Davis, Howard Shrobe, and Peter Szolovits; What Is a Knowledge Representation?  
AI Magazine, 14(1):17-33,1993
- Ronald Fagin, Joseph Y. Halpern, Yoram Moses, Moshe Y. Vardi *Reasoning About  
Knowledge*, MIT Press, 1995, ISBN 0-262-06162-7.
- Ronald J. Brachman, Hector J. Levesque (eds). *Readings in Knowledge Representation*,  
Morgan Kaufmann, 1985, ISBN 0-934613-01-X
- Ronald J. Brachman, Hector J. Levesque *Knowledge Representation and Reasoning*,  
Morgan Kaufmann, 2004 ISBN 978-1-55860-932-7
- Ronald J. Brachman; What IS-A is and isn't. An Analysis of Taxonomic Links in Semantic  
Networks; IEEE Computer, 16 (10); October 1983
- Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (3rd  
ed.). Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-604259-7, p. 437-  
439.