ARTIFICIAL INTELLIGENCE & ROBOTICS

(18MIT25E)

UNIT III

Knowledge Representation Issues: Approaches to Knowledge Representation – The Frame Problem – Computable Functions & Predicates – Resolution – Procedural versus Declarative Knowledge.

TEXT BOOKS:

1. **ELAINE RICH AND KEVIN KNIGHT, ARTIFICIAL INTELLIGENCE, TMH, SECOND EDITION**
2. **CRAIG JJ, INTRODUCTION TO ROBOTICS,MECHANICS AND CONTROL, PEARSON EDUCATION, NEW DELHI, 2004**

-Dr.P.Radha

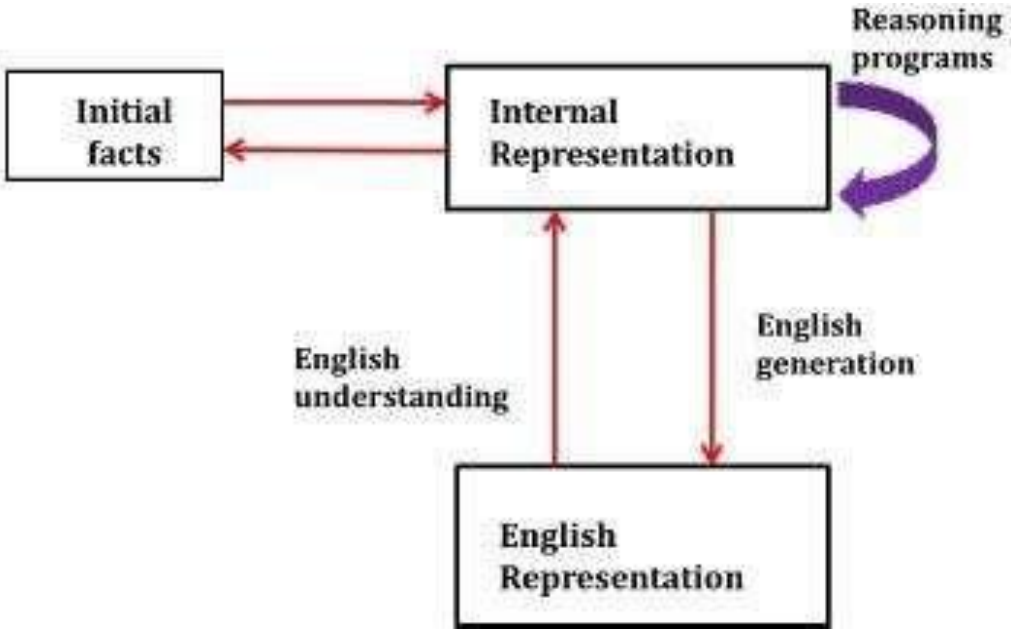
KNOWLEDGE REPRESENTATION

- For the purpose of solving complex problems encountered in AI, we need both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions to new problems.
- A variety of ways of representing knowledge (facts) have been exploited in AI programs. In all variety of knowledge representations , we deal with two kinds of entities.

- Facts: Truths in some relevant world. These are the things we want to represent.

- Representations of facts in some chosen formalism

- **Representations and Mappings**
  - In order to solve complex problems encountered in artificial intelligence, one needs both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions.
  - Knowledge and Representation are two distinct entities. They play central but distinguishable roles in the intelligent system.
  - Knowledge is a description of the world. It determines a system's competence by what it knows.
  - Moreover, Representation is the way knowledge is encoded. It defines a system's performance in doing something.
- Different types of knowledge require different kinds of representation


Reasoning
programs
Initial
facts
Internal
Representation
English
understanding
English
generation
English
Representation

- One way to think of structuring these entities is as two levels:
- Knowledge level at which facts are described.
- Symbol level at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

- Rather than thinking on one level on top of another, we will focus on facts, on representations, and on the two-way mappings that must exist between them , these links are called as Representation Mappings.
- Forward representation mapping maps from facts to representation
- Backward representation mapping goes from representation to facts.

- One representation of facts is to represent using natural language (English) sentences.

Example:

English Sentence: Spot is a dog

Represented in Logic as dog(Spot)

Approaches to Knowledge Representation

Mapping between Facts and Representation

- Knowledge is a collection of facts from some domain.
- Also, We need a representation of "facts" that can manipulate by a program.
- Moreover, Normal English is insufficient, too hard currently for a computer program to draw inferences in natural languages.
- Thus some symbolic representation is necessary.

A good knowledge representation enables fast and accurate access to knowledge and understanding of the content.

A knowledge representation system should have following properties.

1. Representational Adequacy
    - The ability to represent all kinds of knowledge that are needed in that domain.
2. Inferential Adequacy
    - Also, The ability to manipulate the representational structures to derive new structures corresponding to new knowledge inferred from old.
3. Inferential Efficiency
    - The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising direction.
4. Acquisitional Efficiency
    - Moreover, The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Relational Knowledge

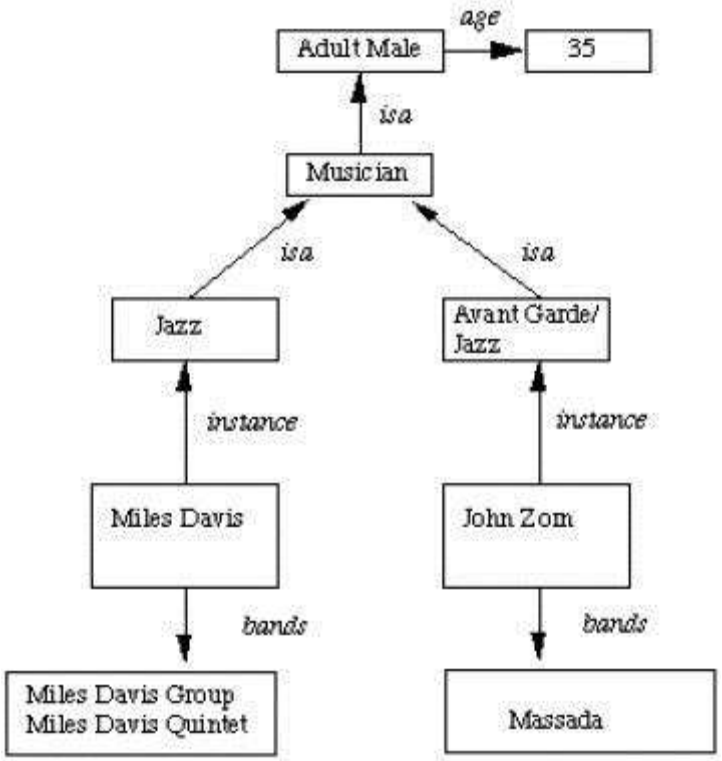
- The simplest way to represent declarative facts is a set of relations of the same sort used in the database system.
- Provides a framework to compare two objects based on equivalent attributes. o Any instance in which two different objects are compared is a relational type of knowledge.
- The table below shows a simple way to store facts.
    - Also, The facts about a set of objects are put systematically in columns.
    - This representation provides little opportunity for inference.

Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right - Right
Mays	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

- Given the facts, it is not possible to answer a simple question such as: “Who is the heaviest player?”
- Also, But if a procedure for finding the heaviest player is provided, then these facts will enable that procedure to compute an answer.
- Moreover, We can ask things like who “bats – left” and “throws – right”.

- **Inheritable Knowledge**
  - Here the knowledge elements inherit attributes from their parents.
  - The knowledge embodied in the design hierarchies found in the functional, physical and process domains.
  - Within the hierarchy, elements inherit attributes from their parents, but in many cases, not all attributes of the parent elements prescribed to the child elements.
  - Also, The inheritance is a powerful form of inference, but not adequate.
  - Moreover, The basic KR (Knowledge Representation) needs to augment with inference mechanism.
  - Property inheritance: The objects or elements of specific classes inherit attributes and values from more general classes.
  - So, The classes organized in a generalized hierarchy.

Inheritable knowledge



- Boxed nodes — objects and values of attributes of objects.
- Arrows — the point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The steps to retrieve a value for an attribute of an instance object:

1. Find the object in the knowledge base
2. If there is a value for the attribute report it
3. Otherwise look for a value of an instance, if none fail
4. Also, Go to that node and find a value for the attribute and then report it
5. Otherwise, search through using is until a value is found for the attribute.

- **Inferential Knowledge**
  - This knowledge generates new information from the given information.
  - This new information does not require further data gathering form source but does require analysis of the given information to generate new knowledge.
  - Example: given a set of relations and values, one may infer other values or relations. A predicate logic (a mathematical deduction) used to infer from a set of attributes. Moreover, Inference through predicate logic uses a set of logical operations to relate individual data.
  - Represent knowledge as formal logic: All dogs have tails ∀x*: dog(x)* → *hastail(x)*
  - Advantages:
    - A set of strict rules.
    - Can use to derive more facts.
    - Also, Truths of new statements can be verified.
    - Guaranteed correctness.
  - So, Many inference procedures available to implement standard rules of logic popular in AI systems. e.g Automated theorem proving

The Frame Problem

- **In artificial intelligence, the frame problem describes an issue with using first-order logic to express facts about a robot in the world. Representing the state of a robot with first-order logic requires the use of many axioms that simply imply that things in the environment do not change arbitrarily. For example, Hayes describes a "block world" with rules about stacking blocks together. The frame problem is the problem of finding adequate collections of axioms for a viable description of a robot environment.**

- The frame problem occurs even in very simple domains. A scenario with a door, which can be open or closed, and a light, which can be on or off, is statically represented by two propositions open and on. If these conditions can change, they are better represented by two predicates open(t) and on(t) that depend on time;
- such predicates are called fluents. A domain in which the door is closed and the light off at time 0, and the door opened at time 1, can be directly represented in logic[clarification needed] by the following formulae.
- ¬open(0)
- ¬on(0)
- open(1)

Computable Functions and Predicates

- To express simple facts, such as the following greater-than and less-than relationships: gt(1,O) lt(0,1) gt(2,1) lt(1,2) gt(3,2) lt(2,3)
- It is often also useful to have computable functions as well as computable predicates. Thus we might want to be able to evaluate the truth of gt(2 + 3,1)
- To do so requires that we first compute the value of the plus function given the arguments 2 and 3, and then send the arguments 5 and 1 to gt.

- Explicit formal type of knowledge
  - Explicit knowledge
  - Exists outside a human being;
  - It is embedded.
  - Can be articulated formally.
  - Also, Can be shared, copied, processed and stored.
  - So, Easy to steal or copy
- Drawn from the artifact of some type as a principle, procedure, process, concepts.

Consider the following set of facts, again involving Marcus:

- Marcus was a man.
  man(Marcus)
- Marcus was a Pompeian.
  Pompeian(Marcus)
- Marcus was born in 40 A.D.
  born(Marcus, 40)
- All men are mortal.
  x: man(x) → mortal(x)
- All Pompeians died when the volcano erupted in 79 A.D.
  erupted(volcano, 79) ∧ ∀ x : [Pompeian(x) → died(x, 79)]
- No mortal lives longer than 150 years.
  ∀ x: ∀ t1: ∀ t2: *mortal(x) ∧ born(x, t1) ∧ gt(t2 – t1,150) → dead(x, t2)*
- It is now 1991.
  now = 1991

- So, Above example shows how these ideas of computable functions and predicates can be useful. It also makes use of the notion of equality and allows equal objects to be substituted for each other whenever it appears helpful to do so during a proof.
  - So, Now suppose we want to answer the question “Is Marcus alive?”
  - The statements suggested here, there may be two ways of deducing an answer.
  - Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.
  - Also, As soon as we attempt to follow either of those paths rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking.

So we add the following facts:

- Alive means not dead.

$\forall$ x: $\forall$ t: [alive(x, t) $\rightarrow$ $\neg$ dead(x, t)] $\wedge$ [$\neg$ dead(x, t) $\rightarrow$ alive(x, t)]

- If someone dies, then he is dead at all later times.

$\forall$ x: $\forall$ t1: $\forall$t2 *: died(x, t1)* $\wedge$ *gt(t2, t1)* $\rightarrow$ *dead(x, t2)*

- So, Now let's attempt to answer the question "Is Marcus alive?" by proving: $\neg$ ***alive(Marcus, now)***

Resolution

Propositional Resolution

- Convert all the propositions of F to clause form.
- Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
- Repeat until either a contradiction is found or no progress can be made:
  - Select two clauses. Call these the parent clauses.
  - Resolve them together. The resulting clause, called the resolvent, will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals L and ¬ *L* such that one of the parent clauses contains *L* and the other contains ¬*L*, then select one such pair and eliminate both *L* and ¬ *L* from the resolvent.
- If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of classes available to the procedure

- The Unification Algorithm
  - In propositional logic, it is easy to determine that two literals cannot both be true at the same time.
  - Simply look for L and ¬L in predicate logic, this matching process is more complicated since the arguments of the predicates must be considered.
  - For example, man(John) and ¬man(John) is a contradiction, while the man(John) and man(Spot) is not.
  - Thus, in order to determine contradictions, we need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical.
  - There is a straightforward recursive procedure, called the unification algorithm, that does it.

Algorithm: Unify(L1, L2)

- If L1 or L2 are both variables or constants, then:
  - If L1 and L2 are identical, then return NIL.
  - Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return (L2/L1).
  - Also, Else if L2 is a variable, then if L2 occurs in L1 then return {FAIL}, else return (L1/L2) Else return {FAIL}.
- If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL}.
- If LI and L2 have a different number of arguments, then return {FAIL}.
- Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L2.)
- For I ← 1 to the number of arguments in L1 :
  - Call Unify with the i^{th} argument of L1 and the i^{th} argument of L2, putting the result in S.
  - If S contains FAIL then return {FAIL}.
  - If S is not equal to NIL then:
    - Apply S to the remainder of both L1 and L2.
    - SUBST: = APPEND(S, SUBST).
- Return SUBST.

Resolution Procedure

- Resolution is a procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.
- Resolution produces proofs by refutation.
- In other words, ***to prove a statement (i.e., to show that it is valid), resolution attempts to show that the negation of the statement produces a contradiction with the known statements (i.e., that it is unsatisfiable).***
- The resolution procedure is a simple iterative process: at each step, two clauses, called the parent clauses, are compared (resolved), resulting in a new clause that has inferred from them. The new clause represents ways that the two parent clauses interact with each other. Suppose that there are two clauses in the system:

- ***winter* V *summer***
- ¬ ***winter* V *cold***

- Now we observe that precisely one of winter and ¬ winter will be true at any point.
- If winter is true, then cold must be true to guarantee the truth of the second clause. If ¬ winter is true, then summer must be true to guarantee the truth of the first clause.
- Thus we see that from these two clauses we can deduce ***summer V cold***
- This is the deduction that the resolution procedure will make.
- Resolution operates by taking two clauses that each contains the same literal, in this example, ***winter***.
- Moreover, The literal must occur in the positive form in one clause and in negative form in the other. The resolvent obtained by combining all of the literals of the two parent clauses except the ones that cancel.
- If the clause that produced is the empty clause, then a contradiction has found.

Procedural versus Declarative Knowledge

- We have discussed various search techniques in previous units. Now we would consider a set of rules that represent,
  - Knowledge about relationships in the world and
- Knowledge about how to solve the problem using the content of the rules

Procedural vs Declarative Knowledge Procedural Knowledge

- A representation in which the control information that is necessary to use the knowledge is embedded in the knowledge itself for e.g. computer programs, directions, and recipes; these indicate specific use or implementation;
- The real difference between declarative and procedural views of knowledge lies in where control information reside.

- For example, consider the following
- *Man (Marcus)*
- *Man (Caesar)*
- *Person (Cleopatra)*
- $\forall x: Man(x) \rightarrow Person(x)$
- *Now, try to answer the question.* Type equation here.
- $\exists y: Person(y)$
- The knowledge base justifies any of the following answers.
- *Y=Marcus Y=Caesar Y=Cleopatra*

- We get more than one value that satisfies the predicate.
- If only one value needed, then the answer to the question will depend on the order in which the assertions examined during the search for a response.
- If the assertions declarative then they do not themselves say anything about how they will be examined. In case of procedural representation, they say how they will examine

Declarative Knowledge

- A statement in which knowledge specified, but the use to which that knowledge is to be put is not given.
- For example, laws, people's name; these are the facts which can stand alone, not dependent on other knowledge;
- So to use declarative representation, we must have a program that explains what is to do with the knowledge and how.

- For example, a set of logical assertions can combine with a resolution theorem prover to give a complete program for solving problems but in some cases, the logical assertions can view as a program rather than data to a program.
- Hence the implication statements define the legitimate reasoning paths and automatic assertions provide the starting points of those paths.
- These paths define the execution paths which is similar to the ‘if then else ‘‘in traditional programming.
- So logical assertions can view as a procedural representation of knowledge