KNOWLEDGE REPRESENTATION

It refers to the design of data structures capable of storing knowledge in useful form. It generally deals with the problem of how to model the world sufficiently for intelligent action. For a knowledge-based intelligent program, we need:

i) To represent knowledge about the world in a *formal language*
ii) To reason about the world using *inferences* in the language
iii) To decide what action to take by *inferring* that the selected action is good

Classification of Knowledge Representations

Knowledge representations can generally be classified into the following:

i) Declarative Representation declares every piece of knowledge and permits the reasoning system to use the rules of inference to come up new pieces of information.
e.g. Consider the following statements:
"All carnivorous have sharp teeth"
"Cheetah is a carnivore"
Using declarative representation, the statements can be represented as follows:
$\forall$ x (carnivore(x) $\rightarrow$sharp_teeth(x))
carnivore(cheetah)
Using the above representation it is possible to deduce that "Cheetah has sharp teeth"

ii) Procedural representation represents knowledge as procedures and the inferencing mechanisms manipulate the procedures to arrive at the result. E.g. the following are procedures for the above statements:

```
procedure carnivore (x);
  if (x=Cheetah) then return true
          else return false
end procedure carnivore (x).

procedure sharp_teeth (x);
  if carnivore(x) then return true
          else return false
end procedure sharp_teeth (x).
```

To see whether cheetah has sharp teeth, one should activate procedure sharp_teeth with variable x instantiated to value cheetah. The procedure then calls procedure carnivore (x) with the value of (x=Cheetah). Procedure carnivore returns true and so is procedure sharp_teeth.

Other forms of knowledge representation are: structural, classification and meta-knowledge.

Knowledge Representation Languages: – special notations that allows to easily represent and reason with complex knowledge about the world. New facts should be easily inferred from existing knowledge.

General requirements for a KR language:

(i) *Representational adequacy:* - This is the ability to represent all of the kinds of knowledge that is needed in a given domain.
(ii) *Inferential adequacy:* -This is the ability to represent all of the kinds of inferential procedures (procedures that manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old).
(iii) *Inferential efficiency:* - This is the ability to represent efficient inference procedures (for instance, by incorporating into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions).
(iv) *Acquisitional efficiency:* - This is the ability to acquire new information easily.

Knowledge Representations Schemes/ Languages
The following are some of the schemes used to represent knowledge.

1. Logic
2. Semantics Nets
3. Rules
4. Frames

1. LOGIC

Logic is a formal system in which the formulas or sentences have true or false values. To build a logic-based representation:

i). User defines a set of primitive symbols and the associated semantics
ii). Logic defines the ways of putting these symbols together so that the user can define legal sentences in the language that represent true facts in the world
iii). Logic defines ways of inferring new sentences from existing ones

The types of logic to be considered are:
i) Propositional logic or Boolean logic
ii) Predicate calculus or first order predicate logic (FOL)

i) Propositional Logic (PL)

Propositional logic (Boolean logic) consists of symbols that represent whole propositions (facts). For example B may represent 'It is sunny outside', that may be true or false. Propositions may be combined using Boolean connectives which generate sentences with more complex meanings.

Facts: - are claims about the world that are True or False.

A **sentence** (also called a formula or well-formed formula or wff) is defined as:

- A symbol
- If S is a sentence, then ~S is a sentence, where "~" is the "not" logical operator
- If S and T are sentences, then (S v T), (S ^ T), (S => T), and (S <=> T) are sentences.

Examples of PL sentences:

- (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")
- Q => P (here meaning "If it is humid, then it is hot")
- Q (here meaning "It is humid.")

Note: Syntactically correct logical formulas are called *well-formed formulas* (wff). Such wff are thus used to represent real-world facts.

Truth: - A sentence is True if the state of affairs it describes is actually the case in the world. So, truth can only be assessed with respect to the semantics.

Propositional Calculus Semantics
An **interpretation** of a set of propositions is the assignment of a truth value, either T or F to each propositional symbol. The symbol true is always assigned T, and the symbol false is assigned F.

Logical Connectors
Logical connectors (&, ∨, ¬, →, ↔) are used to connect formulas (propositions or predicates) so as to represent more complex facts. The truth value of these expressions depends on the truth values of their components, according to the following rules:

X & Y is TRUE if X is TRUE and Y is TRUE; otherwise X & Y is FALSE.

X ∨ Y is TRUE if either X is TRUE or Y is TRUE or both.

$\neg X$ is TRUE if X is FALSE, and FALSE if X is TRUE.

$X \rightarrow Y$ means X implies Y. This implication is true if X is false or Y is TRUE.

$X \leftrightarrow Y$ is TRUE if both X and Y are TRUE, or both X and Y are false.

The truth table below summarizes the above operations

		AND	OR	IMPLICATION	NEGATION	EQUIVALENCE
X	**Y**	**$X \wedge Y$**	**$X \vee Y$**	**$X \rightarrow Y$**	**$\neg X$**	**$X \leftrightarrow Y$**
T	T	T	T	T	F	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

Validity
A valid sentence (also called a tautology) is a sentence that is True under all interpretations. Hence, no matter what the world is actually like or what the semantics is, the sentence is True. For example "It's raining or it's not raining.”. Validity usually enables the machine to check premises to determine if the conclusion is true.

Contradiction
An inconsistent sentence (also called a contradiction) is a sentence that is False under all interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."

Prove in Propositional Calculus using a truth table
Using truth table, show the validity of the following statement:

$P \wedge Q \Leftrightarrow Q \wedge P$

P	**Q**	**$P \wedge Q$**	**$Q \wedge P$**	**$P \wedge Q \Leftrightarrow Q \wedge P$**
T	T	T	T	T
T	F	F	F	T
F	T	F	F	T
F	F	F	F	T

The sentence is True under all interpretations (tautology). Therefore it is valid

Exercise:
Show the validity of the following statements:
(i) $((P \vee H) \wedge \neg H) \Rightarrow P$
(ii) $P \wedge (Q \wedge R) \Leftrightarrow (P \wedge Q) \wedge R$
(iii) $P \vee (Q \vee R) \Leftrightarrow (P \vee Q) \vee R$
(iv) $P \wedge Q \Leftrightarrow Q \wedge P$
(v) $P \vee Q \Leftrightarrow Q \vee P$
(vi) $P \wedge (Q \vee R) \Leftrightarrow (P \wedge Q) \vee (P \wedge R)$
(vii) $P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)$

Rules of Inference
The inference rules of logic are used to infer new facts from the explicitly represented ones. The following is a list of inference rules.

i) Modus Ponens (or implication-Elimination): - If ($P \rightarrow Q$) and P is true, then Q is true, written also as ((($P \rightarrow Q$) and P) |-- Q)
e.g.
→ P (meaning " it is hot”)
→ Q => P ("If it is humid, then it is hot")
→ Q (meaning "It is humid.")

If we know that “Q => P “ and “Q”, then we can infer “P”

ii) **And-Elimination:** given a conjunction, you can infer any conjunct. Represented as:
$A_1 \wedge A_2 \wedge A_3 \wedge \ldots\ldots \wedge A_n \vdash A_i$

iii) **And-Introduction:** given a list of sentences, you can infer their conjunction. Represented as:
$A_1, A_2, A_3, \ldots\ldots, A_n \vdash A_1 \wedge A_2 \wedge A_3 \wedge \ldots\ldots \wedge A_n$

iv) **Or-Introduction**: given a sentence, you can infer all its disjunctions with any thing else. Represented as:
$A_I \vdash A_1 \vee A_2 \vee A_3 \vee \ldots\ldots \vee A_n$

v) **Double-Negation Elimination:** given a doubly negated sentence infer a positive sentence. Represented as:
$\neg\neg A \vdash A$

vi) **Unit resolution:** given a disjunction, if one of the disjuncts is false then infer the other. This is a special case of resolution. Represented as: $A \vee B, \neg B \vdash A$, alternatively: $A \vee B, \neg A \vdash B$

vii) **Resolution:** implication is transitive. Represented as: $A \vee B, \neg B \vee C \vdash A \vee C$ or equivalently: $\neg A \Rightarrow B, B \Rightarrow C \vdash \neg A \Rightarrow C$.

viii) **The ∀-elimination (or universal specialization) rule: -** For any *well-formed formulas* (wff) of the form "$\forall x\ \Phi(x)$", we can conclude "$\Phi(A)$" for any individual "A". This rule is also written as
$((\forall x\ \Phi(x)) \vdash\!\!-\ \Phi(A))$
For instance, if we know that
$\forall x\ Man(x) \rightarrow Mortal(x)$, we can apply this to the individual Socrates, to get
$Man(Socrates) \rightarrow Mortal(Socrates)$

Using Inference Rules to Prove a Query/Goal/Theorem
A proof is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference. The last sentence is the query (also called goal or theorem) that needs to be proved.

Example of a prove for the "weather problem" given above.

1.	Q	Premise
2.	Q => P	Premise
3.	P	Modus Ponens(1,2)
4.	(P ^ Q) => R	Premise
5.	P ^ Q	And Introduction(1,3)
6.	R	Modus Ponens(4,5)

Note
Propositional Logic (PL) is not a very expressive language because:
- Hard to identify "individuals." E.g., Mary, 3
- Can't directly talk about properties of individuals or relations between individuals. E.g., tall(Bill)

ii) First Order Predicate Logic
Predicate logic (first order logic/ Predicate calculus), provides a way of representing the world in terms of objects and predicates on objects (properties of objects or relations between objects). *Connectives* are used to combine predicates.
- The objects from the real world are represented by constant symbols (a, b, c …). E.g. the symbol "Tom" may represent a certain individual called Tom.

- Properties of objects may be represented by predicates applied to those objects (P (a) ...): e.g. "male(Tom)" represents that Tom is a male.
- Relationships between objects are represented by predicates with more arguments: e.g. "father (Tom, Bob)" represents the fact that Tom is the father of Bob.
- The value of a predicate is one of the boolean constants T (i.e. true) or F (i.e. false). e.g. "father(Tom, Bob) = T" means that the sentence "Tom is the father of Bob" is true. "father(Tom, Bob) = F" means that the sentence "Tom is the father of Bob" is false.

Besides constants, the arguments of the predicates may be functions (f,g,...) or variables (x,y,...).
Variable symbols represent potentially any element of a domain and allow the formulation of general statements about the elements of the domain.

Quantifiers
Quantifiers are statements that allow descriptions about the universe (objects environments) at once instead of numerating the objects, i.e. it extends PL by allowing quantification over individuals of a given domain of discourse. Two standard quantifiers in predicate logic are universal and existential quantifiers.

i) **Universal quantification (∀) : -** This quantification is used when the predicate is true for all objects. For example: $\forall(x)$, cat(x) $\Rightarrow$ mammal(x).

ii) **Existential quantification (∃): -** This quantifier is used when a predicate P is true for some object in the universe. For example some is tall may be denoted as $\exists$x tall(x). Someone likes a given person is denoted by $\exists$x, a likes(a, x).

Atomic sentences: - Are sentences that consist of a predicate name followed by a number of arguments (arity).

Axioms
Axioms are basic facts about a domain. Axioms and definitions are used to prove theorems. *Independent axioms* are those axioms that cannot be derived from other axioms.

An axiomatic system
To solve artificial intelligence problems by using the logic representation one has to define an axiomatic system. An *axiomatic system* consists of a set of facts and inference rules, representing real-world knowledge. It can be used, for instance, to infer new facts or to prove that a certain fact is true.

Example
The following is an example of an axiomatic system:

1. Marcus was a man
2. Marcus was a Pompeian
3. All Pompeians were Romans
4. Caesar was a ruler
5. All Romans were either loyal to Caesar or hated him
6. Everyone is loyal to someone
7. People only try to assassinate rulers they are not loyal to
8. Marcus tried to assassinate Caesar
9. All men are persons

Qs. Translate these sentences into formulas in predicate logic

1. man(Marcus)
2. Pompeian(Marcus)

3. ∀x (Pompeian(x) → Roman(x))
4. ruler(Caesar)
5. ∀x (Roman(x) → loyalto(x,Caesar)∨hate(x,Caesar))
6. ∀x ∃y (person(x) → person(y) & loyalto(x,y))
7. ∀x ∀y (person(x) & ruler(y) & tryassassinate(x,y) → ¬loyalto(x,y))
8. tryassassinate(Marcus,Caesar)
9. ∀x (man(x) → person(x))

Natural deduction: It involves applying a sequence of rules of inferences using either sentences in the KB or sentences derived earlier in the proof, until the conclusion sentences is derived.

Example:
In the above axiomatic system, use natural deduction to prove that Marcus was not loyal to Caesar (¬loyalto(Marcus, Caesar))

From	∀x (man(x) → person(x))
deduce	man(Marcus) → person(Marcus) by universal specialization.

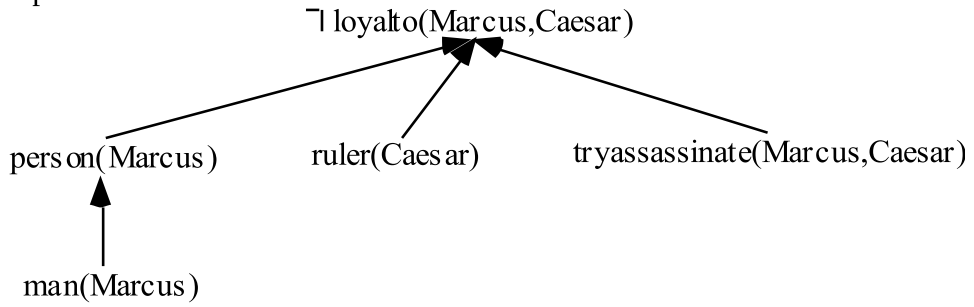
From	man(Marcus) → person(Marcus)
and	man(Marcus)
deduce	person(Marcus) by modus ponens.

From	∀x ∀y (person(x) & ruler(y) & tryassassinate(x,y) → ¬loyalto(x,y)
deduce	person(Marcus) & ruler(Caesar) & tryassassinate(Marcus, Caesar) → ¬loyalto(Marcus, Caesar)

by applying the universal specialization twice.

From	person(Marcus) & ruler(Caesar) & tryassassinate(Marcus, Caesar) → ¬loyalto(Marcus, Caesar)
and	person(Marcus) & ruler(Caesar) & tryassassinate(Marcus, Caesar)
deduce	¬loyalto(Marcus, Caesar) by modus ponens.

The following is the proof tree:



The Clausal Form of Logic

This is a restricted subset of the standard form of logic. It has the advantage that it bears greater resemblance to other formalisms used for databases and programming. Moreover, simple, efficient, and reasonably natural resolution theorem provers have been developed for it.

The arrow of clausal form "←"is written in the opposite direction to that normally used in the standard form of logic i.e.

A ← B (A if B)

Instead of

B → A (if B then A).

The notation "A ← B" is used in order to draw attention to the conclusion of the clause.

In the clausal form of logic all the expressions are clauses. A *clause* is an expression of the form

A1,...,Am ← B1,...,Bn

where A1,...,Am, B1,...,Bn are atomic formulae, $n \geq 0$ and $m \geq 0$.
The atomic formulae B1,...,Bn are the joint conditions of the clause and A1,...,Am are the alternative conclusions.

There are no symbols used in clausal representation other than the arrow (←) and the comma(,). However, information that was conveyed by the other logical symbols AND, OR, and NOT are implied based on the position in the clause. Every atomic formula to the left of the arrow is assumed to be separated by an OR. Every atomic formula to the right of the arrow is assumed to be separated by an AND. Some examples of similar constructs between standard and clausal form of logic include:

$Y \leftarrow X$	is the same as	$X \rightarrow Y$
$Z \leftarrow X, Y$	is the same as	$X \wedge Y \rightarrow Z$
$D, E \leftarrow A, B, C$	is the same as	$A \wedge B \wedge C \rightarrow D \vee E$

LOGIC PROGRAMMING AND PROLOG

Logic programming refers to a family of languages and an associated programming style based on writing programs as a set of assertions (facts and inference rules). The execution of a logic program is a kind of deduction on the specified facts and inference rules. An example of such programming language is PROLOG.

PROLOG program

A PROLOG program consists of **facts** for describing object features or relationships, and **rules** for inferring new properties and relationships from other properties and relationships.

For instance: female(pam). % expresses the fact that pam is a female
parent(pam, bob). % expresses the fact that pam is a parent of bob

A rule has the following form: X :- Y1, Y2, ... ,Yn. which is read as "X is true if Y1 is true and Y2 is true and ... and Yn is true"
X represents the head of the rule and Y1, Y2, ... ,Yn represents the body. ":-" is read "if"
For instance: mother(X, Y):- parent(X, Y), female(X)
means X is mother of Y if X is parent of Y and X is a female.

Generally *PROLOG facts and rules are Horn clauses.*

A Horn clause is a clause with at most one positive literal (or a clause with at most one atom in the left hand side of :-).

Example

The following is a PROLOG program that describes a particular extended family

```
parent(pam, bob).                 % Pam is a parent of Bob
parent(tom, bob).                 % Notice that in Prolog the constants
parent(tom, liz).                 % are written in lower case
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
male(tom).            % Tom is a male
male(bob).
```

```
female(pam).            % Pam is a female
female(liz).
female(ann).
female(pat).
male(jim).

mother(X, Y) :-                     % X is the mother of Y if
            parent(X, Y),           %  X is a parent of Y and
            female(X).              %  X is female
predecessor(X, Y) :-                % Rule pr1:     X is a predecessor of Y if
                parent(X, Y).       % X is a parent of Y
predecessor(X, Y) :-                % Rule pr2:     X is a predecessor of Y if
              parent(X, Z),         % X is a parent of Z and
             predecessor(Z, Y).     % Z is a predecessor of Y
```

Querying PROLOG Programs

A question to PROLOG is always a sequence of one or more predicates (called goals) as, for instance:

```
?- parent(X, liz).          % Who is Liz's parent ?
X = tom                     % Prolog returns all the values of X
                            % for which parent(X, liz) is true

?- predecessor(pam, X).     % Who are Pam's successors ?
X = bob;                    % i.e. who is a person that Pam is his or her predecessor ?
X = ann;                    % typing ; after a solution asks Prolog to look for
X = pat;                    % an additonal solution
X = jim
```

i). Question with constants

If the question consists of a predicate with no variables like, for instance,

```
?- parent(bob, pat).    % Is Bob a parent of Pat ?
```

then Prolog tries to demonstrate that this fact logically follows from the facts and the rules in the program. It returns Yes if the proof is successful and No otherwise.

```
?- parent(bob, pat).    % Is Bob a parent of Pat ?
yes                     % Prolog answers yes because it found this fact
                        % as an explicitly asserted fact in the program.
```

PROLOG uses the **closed world assumption** which states that all relevant, true assertions are explicitly represented or are derivable from those explicitly represented.

Therefore, any assertion that is neither explicitly represented nor derivable, is considered to be false.

ii). Question with variables

If the question consists of one or more predicates with variables such as:

?- parent(Y, jim), parent(X, Y). Find X and Y such that Y is a parent of Jim

then PROLOG will look for all the instances of the variables from the question (X and Y in the above example) such that the predicates in the question logically follows from the facts and the rules in the program. One says that PROLOG tries to satisfy the goals "parent(Y, jim)" and "parent(X, Y)".

PROLOG returns the found pairs of the values of X and Y, or No if no such pair is found.

```
?- parent(Y, jim), parent(X, Y).
X = bob
Y = pat
```