**Unit II --- Representation Of Knowledge**  **Artificial Intelligence CS6659**

**Propositional Logic: A very simple logic**

**Syntax**

The **syntax** of propositional logic defines the allowable sentences.

1. The **atomic sentences -** the indivisible syntactic elements-consist of a single **proposition symbol.**
2. **Each such symbol stands for a proposition that can be true or false.**
3. There are two proposition symbols with fixed meanings:

True is the always-true proposition and False is the always-false proposition.

**Complex sentences** are constructed from simpler sentences using **logical connectives.**

There are five connectives in common use:

1. ˄ (and) – conjunction

Ex: P ˄Q

1. ˅ (or) – Disjunction

ex: P ˅Q

1. → (implies) – Implication or Conditional

Ex: (P ˄Q) ═> R

(P ˄Q) – premise or antecedent

R – conclusion or consequent

1. ↔ (equivalent ) – Equivalence or Biconditional ( if and only if)

Ex: (P ˄Q) ↔ ( Q ˄ P)

1. ⌐ (not) – Negation is the only connective that operates on a single sentence.

Ex: ⌐P



1. every sentence constructed with binary connectives must be enclosed in parentheses.
2. This ensures that the syntax is completely unambiguous.

The order of precedence in propositional logic is (from highest to lowest): Hence, the sentence



is equivalent to the sentence 



**Semantics**

1. The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
2. In propositional logic, a model simply fixes the truth value-true or false-for every proposition symbol.

For example, if the sentences in the knowledge base make use of the proposition symbols

then one possible model is 

1. The semantics for propositional logic must specify how to compute the truth value of any sentence, given a model.
2. This is done recursively.
3. All sentences are constructed from atomic sentences and the five connectives;
4. Therefore, we need to specify how to compute the truth of atomic sentences and how to compute the truth of sentences formed with each of the five connectives.
5. Atomic sentences are easy:
6. True is true in every model and False is false in every model.
7. The truth value of every other proposition symbol must be specified directly in the model.

For complex sentences, we have rules such as

1. For any sentence S and any model m, the sentence ⌐S is true in m if and only if Sis false in m.
2. Such rules reduce the truth of a complex sentence to the truth of simpler sentences
3. The rules for each connective can be summarized in a **truth table** that specifies the truth value of a complex sentence for each possible assignment of truth values to its components.



1. "If P is true, then I am claiming that Q is true. Otherwise I am making no claim." The only way for this sentence to be false is if P is true but Q is false.
2. The truth table for a biconditional, *P=>* Q, shows that it is true whenever both

P => Q and Q => P are true. In English, this is often written as "P if and only if Q" or *"P* if and only if Q."

**Equivalence, validity, and satisfiability**

1. The first concept is logical **equivalence**: two sentences α and β are logically equivalent if they are true in the same set of models.



**Standard logical equivalences the symbols *a, β* and *γ* stand for arbitrary sentences of propositional logic. Example, we can easily show (using truth tables) that P ˄ Q and Q ˄ P are logically equivalent.**

1. The second concept we will need is **validity**: A sentence is valid if it is true in *all* models.
2. For example, the sentence P V ⌐Pis valid.
3. Valid sentences are also known as *tautologies*-they are *necessarily* true and hence vacuous.
4. Because the sentence *True* is true in all models, every valid sentence is logically equivalent to *True.*
5. The final concept we will need is **satisfiability.**
6. A sentence is satisfiable if it is true in *some* model.
7. If a sentence *a* is true in a model m, then we say that m **satisfies** α , or that m **is a model of** α.
8. Satisfiability can be checked by enumerating the possible models until one is found that satisfies tlie sentence.
9. Determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete.
10. With appropriate transformations, search problems can also be solved by checking satisfiability.
11. Validity and satisffiability are of course connected: α is valid if and only if ⌐α is unsatisfiable. contrapositively, α is satisfiable iff ⌐α is not valid.

**Unsatisfiability:**

1. Proving ,β from α by checking the unsatisfiability of (α ˄ ⌐β )corresponds exactly to the standard mathematical proof technique of *reductio ad absurdurn* (literally, "reduction to an absurd thing").
2. It is also called proof by **refutation** or proof by **contradiction.**
3. One assumes a sentence, β to be false and shows that this leads to a contradiction with known axioms *α.*
4. This contradiction is exactly what is meant by saying that the sentence is (α ˄ ⌐β) unsatisfiable.

**Reasoning Patterns In Propositional Logic**

* Standard patterns of inference that cani be applied to derive chains of conclusions that lead to the desired goal.These patterns of inference are called **inference rules.**

1. **Modus Ponens:**



* The notation means that, whenever any sentences of the form α ═> β and α are given, then the sentence β can be inferred.

1. **And-Elimination:**



* From a conjunction, any of the conjuncts can be inferred

1. All of the logical equivalences can be used as inference rules.

****

1. For example,the equivalence for biconditional elimination yields the two inference rules

****

Not all inference rules work in both directions like this.

1. **And – Introduction** :

From a list of sentences their conjunction is inferred.

1 , 2, ...............n

1  2  ........ n

1. **Or- Introduction** :

From a sentence , its disjunction with anything is inferred.

i

1  2  .....… n

1. **Double – Negation Elimination** :

From a double elimination positive sentence is inferred

⌐ ⌐ α

α

1. **Unit resolution :**

From a disjunction, if one of the disjuncts is false, then the true one is inferred.

α , ⌐β

α

1. **Resolution :**

Implication is transitive.

α , ⌐β γ ⌐α β , β γ

or

α γ ⌐α γ

**SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC**

**Models for first-order logic:**

1. The **domain** of a model is the set of objects it contains; these objects are sometimes called **domain elements.**
2. Figure shows a model with five objects:
3. Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.
4. In the figure, Richard and John are brothers. Formally speaking, a relation is just the set of **tuples** of objects that are related.
5. (A tuple is a collection of objects arranged in, a fixed order and is written with angle brackets surrounding the objects.)
6. Thus, the brotherhood relation in this model is the set

*{<Richard the Lionheart, King John>, <King John, Richard the Lionheart>}* → **Eqn 1**

1. The crown is on King John's head, so the "on head relation contains just one tuple, (the crown, King John).
2. The "brother" and "on head" relations are binary relations-that is, they relate pairs of objects.
3. The model also contains unary relations, or properties: the "person" property is true of both Richard and John; the "king" property is true only of John (presumably because Richard is dead at this point); and the "crown" property is true only of the crown.
4. Each person has one left leg, so the model has a unary "left leg" function that includes the following mappings:

*(Richard the Lionheart)* ***→*** *Richard's left leg →* ***Eqn 2***

*(King John) → John's left leg.*

Models in first-order logic require total functions, that is, there must be a value for every input tuple.

Thus, the crown must have a left leg and so must each of the left legs.

****

Figure: A model with five objects

**Symbols and interpretations:**

1. The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions.
2. The symbols, therefore, come in three kinds: constant symbols, which stand for objects; predicate symbols, which stand for relations; and function symbols, which stand for functions.
3. We adopt the convention that these symbols will begin with uppercase letters.
4. For example, we might use the constant symbols Richard and John; the predicate symbols Brother, OnHead, Person, King, and Crown; and the function symbol LeftLeg.

****

1. Each predicate and function symbol comes with an arity that fixes the number of arguments.
2. The semantics must relate sentences to models in order to determine truth. For this to happen, we need an **interpretation** that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.
3. One possible interpretation for our example-which we will call the **intended interpretation**-is as follows:

*Richard* refers to Richard the Lionheart and *John* refers to the evil King John.

*Brother* refers to the brotherhood relation, that is, the set of tuples of objects given in Eqn 1

*OnHead* refers to the "on head” relation that holds between the crown and King John; *Person, King, and Crown* refer to the sets of objects that are persons, kings, and crowns.

*LeftLeg* refers to the "left leg" function, that is, the mapping given in Eqn 2.

1. There are five objects in the model, so there are 25 possible interpretations just for the constant symbols Richard and John.
2. The truth of any sentence is determined by a model and an interpretation for the sentence's symbols.
3. Therefore, entailments, validity, and so on are defined in terms of *all possible models and all possible interpretations.*
4. It is important to note that the number of domain elements in each model may be unbounded-for example, the domain elements may be integers or real numbers.
5. Hence, the number of possible models is unbounded, as is the number of interpretations.

**Terms**

1. A **term** is a logical expression that refers to an object.
2. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object.
3. In the general case, a complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.
4. A complex term is just a complicated kind of name.
5. It is not a "subroutine call" that "returns a value." There is no LeftLeg subroutine that takes a person as input and returns a leg.
6. The formal semantics of terms is straightforward.
7. Consider a term f (t1.......... tn). The function symbol f refers to some function in the model (call it F); the argument terms refer to objects in the domain (call them d 1, .............dn,); and the term as a whole refers to the object that is the value of the function F applied to d 1, .............dn.
8. For example, suppose the *LeftLeg* function symbol refers to the function shown in Equation (2) and *John* refers to King John, then *LeftLeg(John)* refers to King John's left leg. In this way, the interpretation fixes the referent of every term.

**Atomic sentences**

1. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms:

*Brother(Richard, John)*

1. These states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John.
2. Atomic sentences can have complex terms as arguments. Thus,

*Married (Father(Richard), Mother( John))*

states that Richard the Lionheart's father is married to King John's mother (again, under a

suitable interpretation).

1. *An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.*

**Complex sentences**

1. We can use logical connectives to construct more complex sentences, just as in propositional calculus.
2. The semantics of sentences formed with logical connectives is identical to that in the propositional case.
3. Here are four sentences that are true in the model of Figure under our intended interpretation:

*⌐Brother (LeftLeg (Richard, )J ohn)*

*Brother (Richard, John) A Brother (John, Richard)*

*King (Richard) V King (John)*

*⌐King (Richard) => King (John) .*

**Quantifiers**

1. **Quantifiers** express properties of entire collections of objects, instead of enumerating the objects by name.
2. First-order logic contains two standard quantifiers, called ***universal*** and ***existential.***

**Universal quantification ()**

1. Rules such as "Squares neighbouring the wumpus are smelly" and "All kings are persons" are the bread and butter of first-order logic.
2. The second rule, "All kings are persons,'' is written in first-order logic as

King(x) Person(x)

is usually pronounced "For all . . .". (Remember that the upside-down A stands for "all.")

1. Thus, the sentence says, "For all x, if x is a king, then z is a person." The symbol x is called

a **variable**. By convention, variables are lowercase letters.

1. A variable is a term all by itself, and as such can also serve as the argument of a function-for example, *LeftLeg(x).* A term with no variables is called a **ground term**.
2. Intuitively, the sentence  *P,* where *P* is any logical expression, says that *P* is true for very object *x.* More precisely, P is true in a given. model under a given interpretation if P is true in all possible **extended interpretations** constructed from the given interpretation, where each extended interpretation specifies a domain element to which x refers.
3. We can extend the interpretation in five ways:

x → Richard the Lionheart,

*x →* King John,

*x* → Richard's left leg,

x → John's left leg,

*x* → the crown.

1. The universally quantified sentence King(x) Person(x) is true under the original interpretation if the sentence King(x) Person(x) is true in each of the five extended interpretations.
2. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king Richard the Lionheart is a person.

King John is a king King John is a person.

Richard's left leg is a king Richard's left leg is a person.

John's left leg is a king John's left leg is a person.

The crown is a king the crown is a person.

1. The sentence  *King (x) A Person (x)* would be equivalent to asserting

Richard the Lionheart is a king A Richard the Lionheart is a person,

King John is a king A King John is a person,

Richard's left leg is a king A Richard's left leg is a person, and so on.

**Existential quantification ( )**

1. Universal quantification makes statements about every object.
2. King John has a crown on his head, we write

x Crown(x) OnHead (x, John)

x is pronounced "There exists an x such that . . ." or "For some x . . .".

1. The sentence x P says that *P* is true for at least one object *x.* More precisely,x P is true in a given model under a given interpretation if P is true in *at* least *one* extended interpretation that assigns *x* to a domain element.
2. For our example, this means, that at least one of the following must be true:

Richard the Lionheart is a crown A Richard the Lionheart is on John's head;

King John is a crown A King John is on John's head;

Richard's left leg is a crown A Richard's left leg is on John's head;

John's left leg is a crown A John's left leg is on John's head;

The crown is a crown A the crown is on John's head.

1. The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model. This is entirely consistent with the original sentence "King John has a crown on his head."
2. Just as appears to be the natural connective to use with , is the natural connective to use with .
3. Consider the following sentence:

*x Crown ( x )*  *OnHead ( x, John )*

1. Applying the semantics, we see that the sentence says that at least one of the following assertions is true:

Richard the Lionheart is a crown Richard the Lionheart is on John's head;

King John is a crown King John is on John's head;

Richard's left leg is a crown Richard's left leg is on John's head;

and so on.

1. Now an implication is true if both premise and conclusion are true, or *if* ***its premise is false.*** So if Richard the Lionheart is not a crown, then the first assertion is true and the existential is satisfied.

**Nested quantifiers**

1. To express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type.
2. For example, "Brothers are siblings" can be written as

*y Brother(x, y)*  *Sibling(x, y)* .

1. Consecutive quantifiers of the same type can be written as one quantifier with several variables. For example, to say that siblinghood is a symmetric relationship, we can write

*y Sibling (x, y)*  *Sibling (y ,x)*

1. In other cases we will have mixtures. "Everybody loves somebody" means that for every person, there is someone that person loves:

1. On the other hand, to say "There is someone who is loved by everyone," we write,

1. Some confusion can arise when two quantifiers are used with the same variable name.
2. Consider the sentence

*x [Crown( x )*V *(x Brother (Richard,x ))]*.

Here the *x* in *Brother (Richard, x)* is *existentially* quantified.

1. The rule is that the variable belongs to the innermost quantifier that mentions it; then it will not be subject to any other quantification.
2. Another way to think of it is this:  *Brother (Richard, x)* is a sentence about Richard (that he has a brother), not about *x;* so putting a *x* outside it has no effect.
3. It could equally well have been written *z Brother (Richard, z).* Because this can be a source of confusion, we will always use different variables.

**Connections between**  **and**:

1. The two quantifiers are actually intimately connected with each other, through negation. Asserting that everyone dislikes parsnips is the same **as** asserting there does not exist someone who likes them, and vice versa:

*x ⌐ Likes ( x , Parsnips)* is equivalent to ⌐ *x Likes(x, Parsnips)* .

1. We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

*x Likes (x, Ice Cream)* is equivalent to ⌐ *x ⌐Likes(x, Ice Cream)* .

1. Because is really a conjunction over the universe of objects and is a disjunction, it should not be surprising that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:



**Equality**

1. We can use the equality symbol to make statements to the effect that two terms refer to the same object. For example,

*Father( John) = Henry*

says that the object referred to by *Father(John)* and the object referred to by *Henry* are the same.

1. The equality symbol can be used to state facts about a given function, as we just did for the *Father* symbol. It can also be used with negation to insist that two terms are not the same object.
2. To say that Richard has at least two brothers, we would write

*x,* y *Brother( x , Richard ) Brother(y, Richard) ⌐(* *x = y)* .

The sentence

*x,* y *Brother ( x , Richard )*  *Brother( y , Richard )* does not have the intended meaning.

1. In particular, it is true in the model of Figure where Richard has only one brother.
2. To see this, consider the extended interpretation in which both *x* and *y* are assigned to King John.

The addition of *⌐(* *x = y)*  rules out such models. The notation *x*  *y* is sometimes used as an

abbreviation for *⌐(* *x = y).*

**Knowledge engineering in first order logic:**

A knowledge engineer is someone who investigates a particular domain, learns what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain.

**The knowledge engineering process**

Knowledge engineering projects vary widely in content, scope, and difficulty, but all such projects include the following steps:

1. Identify the task.
2. Assemble the relevant knowledge.
3. Decide on a vocabulary of predicates, functions, and (constants.
4. Encode general /knowledge about the domain.
5. Encode a description of the specific problem instance.
6. Pose queries to the inference procedure and get answers.
7. Debug the knowledge base.
8. ***Identify the task.***The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance.
9. ***Assemble the relevant knowledge.*** The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know-a process called knowledge acquisition. At this stage, the knowledge is not represented formally. The idea is to understand the scope of the knowledge base, as determined by the task, and to understand how the domain actually works.
10. ***Decide on a vocabulary of predicates, functions, and (constants***. That is, translate the important domain-level concepts into logic-level names. This involves many questions of knowledge engineering style.
11. ***Encode general /knowledge about the domain.***The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content.
12. ***Encode a description of the specific problem instance****.* If the ontology is well thought out, this step will be easy. It will involve writing simple atomic sentences about instances of concepts that are already part of the ontology.
13. ***Pose queries to the inference procedure and get answers.***This is where the reward is: we can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing.
14. ***Debug the knowledge base.*** Alas, the answers to queries will seldom be correct on the first try. More precisely, the answers will be correct for the knowledge base as written, assuming that the inference procedure is sound, but they will not be the ones that the user is expecting.

The electronic circuits domain

1. **Identify the task:**
2. There are many reasoning tasks associated with digital circuits. At the highest level, one analyzes the circuit's functionality.
3. For example, what are all the gates connected to the first input terminal? Does the circuit contain feedback loops? These will be our tasks in this section.
4. There are more detailed levels of analysis, including those related to timing delays, circuit area, power consumption, production cost, and so on. Each of these levels would require additional knowledge.



* A digital circuit C1, purporting to be a one-bit. full adder. The first two inputs are the two bits to be added and the third input is a carry bit.
* The first output is the sum, and the second output is a carry bit for the next adder.
* The circuit contains two XOR gates, two AND gates and one OR gate.

1. **Assemble the relevant knowledge**
2. What do vie know about digital circuits? For our purposes, they are composed of wires and gates.
3. Signals flow along wires to the input terminals of gates, and each gate produces a signal on the output terminal that flows along another wire.
4. To determine what these signals will be, we need to know how the gates transform their input signals.
5. There are four types of gates: AND, OR, and XOR gates have two input terminals, and NOT gates have one.
6. All gates have one output terminal. Circuits, like gates, have input and output terminals.
7. If we were interested in designing a product that would be profitable, then the cost of the circuit and its speed relative to other products on the market would be important.
8. **Decide on a vocabulary**
9. First, we need to be able to distinguish a gate from other gates. This is handled by naming gates with constants: *XI, X2,* and so on. Although each gate is connected into the circuit in its own individual way, its behavior-the way it transforms input signals into output signals-depends only on its ***type.***
10. For example, we can write *Type(X1)*= *XOR.* This introduces the constant *XOR* for a particular type of gate; the other constants will be called *OR,* AND, and *NOT.*
11. The *Type* function is not the only way to encode the ontological distinction. We could have used a binary predicate, *Type(X1 XOR),* or several individual type predicates, such as *XOR(X1)*
12. A gate or circuit can have one or more input terminals and one or more output terminals. Thus, gate *X1* could have terminals named *XI In1 XI In2,* and *XI Out1.*
13. The connectivity between gates can be represented by the predicate *Connected,* which takes two terminals as arguments, as in *Connected (Out(1, XI), In(1, X2)).*
14. **Encode general knowledge of the domain**

We need only **seven simple rules** to describe everything we need to know about circuits:

1. If two terminals are connected, then they have the same signal: 
2. The signal at every terminal is either 1 or 0 (but not both):



1. Connected is a commutative predicate:



1. An OR gate's output is 1 if and only if any of its inputs is 1: 
2. An **A.ND** gate's output is 0 if and only if any of its inputs is 0:



1. An XOR gate's output is 1 if and only if its inputs are different: 
2. A NOT gate's output is different from its input:



1. **Encode the specific problem instance**

The circuit shown in Figure is encoded as circuit *C1* with the following description. First, we categorize the gates:



Then, we show the connections between them:



1. **Pose queries to the inference procedure**
2. What combinations of inputs would cause the first output of Cl (the sum bit) to be 0 and the

second output of C1 (the carry bit) to be l?



1. The answers are substitutions for the variables *il, i2,* and *i3* such that the resulting sentence is entailed by the knowledge base. There are three such substitutions:



1. What are the possible sets of values of all the terminals for the adder circuit?



1. This final query will return a complete input-output table for the device, which can be used to check that it does in fact add its inputs correctly. This is a simple example of **circuit verification.**
2. **Debug the knowledge base**
3. We can perturb the knowledge base in various ways to see what kinds of erroneous behaviors emerge.
4. For example, suppose we omit the assertion that 1 0 12.
5. Suddenly, the system will be unable to prove any outputs for the circuit, except for the input cases 000 and 110. We can pinpoint the problem by asking for the outputs of each gate.
6. For example, we can ask 

which reveals that no outputs are known at *XI* for the input cases 10 and 01.

1. Then, we look at the axiom for XOR gates, as applied to XI :

****

1. If the inputs are known to be, say, 1 and 0, then this reduces to

****

Now the problem is apparent: the system is unable to infer that 

so we need to tell it that 1 0.