5 Logical Agents and Knowledge Representation using Propositional Logic

Syllabus

Logical Agents, Knowledge-based agents, The Wumpus world, Logic, Propositional logic, Propositional theorem proving, Effective propositional model checking, Agents based on propositional logic.

Contents

Propositional Logic

5.1 Characteristic of Propositional Logic

GTU : Summer-20

1. Propositional logic is declarative (pieces of syntax corresponds to facts).
2. Propositional logic allows partial/disjunctive/negated inference (unlike most data structures and databases)
3. Propositional logic is compositional.

   (Meaning of $B_{1,1} \wedge P_{1,2}$ is deried from meaning of $B_{1,1}$ and meaning of $P_{1,2}$).
4. Meaning in propositional logic is context independent (unlike natural language, where meaning depends on context).

5.2 Drawbacks of Propositional Logic

1. Propositional logic has very limited expressive power (unlike natural language).

   Example : Cannot express "pits causes breezes in adjacent squares".

   For such statement one need to write, one sentence for each square.
2. Propositional logic represents statements about the world without reflecting the structure and without modeling these entities explicitly.
3. Therefore some knowledge is hard or impossible to encode in the propositional logic.
4. Two cases that are hard to represent :

   Case i) Statements about similar objects and their relations :

   a) Statements about similar objects and relations needs to be enumerated.

   b) Because of this knowledge-base grows large.

   c) We need to represent many rules to allow inferences. The solution to this problem is introduce variables.

   Case ii) Statements referring to groups of objects :

   a) Statements referring to groups of objects require exhaustive enumeration of objects.

   b) Solution to this problem is to allow quantification in statements. [which is done in first order logic].

5.3 Syntax for Propositional Logic

Syntax : It defines allowable sentences in the model.

Propositional calculus symbols :

1. Symbols

i) They are propositional calculus symbols

P, Q, R, S .....

ii) Predicate symbols : They are used to represent a relation in a domain.

For example :

If we want to represent a sentence like, "Dipu reads book".

We will use the predicate symbol "READS". The simple predicate will be,

READS (Dipu, Book).

iii) Constant symbols : A constant symbol is used to represent objects or entities in a domain. These objects or entities may be physical objects, people, concepts or anything that is to be named.

For example : In the above formula Dipu and Book are constant symbols.

iv) Variable symbols : Variables like x or y are also symbols. They allow the users to be indefinite about which entity is to be reffered to example READ (x, y).

v) Function symbols : They denote functions in the domain of discourse.

For example : "Ram's mother is married to Ram's father", the atomic formula would be,

MARRIED [MOTHER (Ram), FATHER (Ram)] where MOTHER, FATHER and MARRIED are function symbols.

2. Truth symbols

True, False.

3. Connectives

$\wedge, \vee, \neg, \rightarrow, \equiv$

Note Propositional symbols denote propositions, or statements about the world that may be either true or false, such as "the car is red" or "water is wet".

5.3.1 Propositional Calculus Sentence

i) Every propositional symbol and truth symbol is sentence.

For example : True, P, Q, R are sentences.

ii) Atomic sentence : It is indivisible (non-composite) syntactic element. It consists of proposition symbol. Each such symbol stands for a proposition that can be true or false.

iii) Complex sentence : It is constructed from simpler sentences using logical connectives.

Note Logical sentences are also called as well-formed formulas or WFF's.

5.3.2 Connectives

Atomic formulas can be converted to well formed formulas by combining them with connectives.

a) The AND connective : ∧

The "∧" connective is used to represent compound statement like "Nimu wears pink dress".

WEARS (Nimu, Dress) ∧

COLOUR (Dress, Pink)

Where the predicate WEARS gives the relation between person and object and the predicate COLOUR gives the relation between object and colour.

Formulae built by connecting other formulae by ∧'s are called conjunctions and each of the component of the formula is called as **conjunct.**

b) The OR connective : ∨

The symbol "∨" is used to represent **inclusive** "or".

For example :

"Dipu plays badminton or tennis" can be represented by

PLAYS (Dipu, Badminton) ∨ PLAYS (Dipu, Tennis)

Formulas built by connecting other formulas by ∨'s are called disjunctions and each of the component formula is called as **disjunct.**

c) The NOT connective : ¬

The symbol "¬" is used to represent the NOT connective. It is used to negate the truth values of a formula. It changes the value of sentence from T to F and vice versa.

For example :

Dipu did not read book, can be represented as :

¬ READ (Dipu, Book).

A formula with a ¬ in front of it is called **a negation.**

d) The IF connective : ⇒

The "⇒" is used to represent "if-then" statements.

For example :

"If Nimu buys frock, then its colour is pink" can be represented as :

BUYS (Nimu, Frock) ⇒ COLOUR (Frock, Pink)

or

"If Dipu is caught in the rain then she uses umbrella".

CAUGHT IN (Dipu, Rain) ⇒ USES (Dipu, Umbrella)

A formula built by connecting two formulas with '⇒' is called as **implication**. The left hand side of an implication is called the **antecedant** and the right hand side is called the **consequent.**

e) The BICONDITIONAL connective : ⇔

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" iff "Q".

For example :

"Nimu becomes happy iff Nimu eats icecream." This sentence can be represented as,

BECOMES (Nimu, Happy) ⇔ EATS (Nimu, Icecream)

- **Truth table for five logical connectives :**

P	Q	¬P	P ∧ Q	P ∨ Q	P ⇒ Q	P ⇔ Q
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

where T : True

F : False.

Following are Some of the Legal Sentences (WFF's) in Propositional Logic

1) The negation of a sentence is a sentence.

   For example :

   ¬ P and ¬ false are sentences.

2) The conjunction, 'and', of two sentences is a sentence.

   For example :

   P ∧ ¬ P is a sentences.

3) The disjunction, 'or', of two sentences is a sentence.

For example :

$P \vee \neg P$ is a sentence.

4) The implication of one sentence from another is a sentence.

For example :

$P \rightarrow Q$ is a sentence.

5) The equivalence of two sentences is a sentence.

For example :

$P \vee Q \equiv R$ is a sentence.

5.3.3 Grouping of Symbols in Propositional Logic

In propositional calculus, the symbols ( ) and [ ] are used to group the symbols into subexpressions by which their order of evaluation and meaning are controlled.

For example :

$(P \vee Q) \equiv R$ is different from $P \vee (Q \equiv R)$.

5.3.4 Forming a Legal Sentence in Propositional Logic

An expression is a sentence of propositional calculus if and only if it can be formed of legal symbols through some sequence of these rules.

$((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$

is a well formed sentence in the propositional calculus because

i) P, Q and R propositions and thus sentences.

ii) $P \wedge Q$, the conjunction of two sentences.

$\therefore$ It is a sentence.

iii) $(P \wedge Q) \rightarrow R$ is the implication of a sentence for another.

$\therefore$ It is a sentence.

iv) $\neg P$ and $\neg Q$, the negations of sentences are sentences.

v) $\neg P \vee \neg Q$, the disjunction of two sentences, is a sentence.

vi) $\neg P \vee \neg Q \vee R$, the disjunction of two sentences, is a sentence.

vii) $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$

The equivalence of two sentences is a sentence.

5.3.5 Formal Grammar for Propositional Logic

A BNF (Backus Naur - Form) grammar of sentences in propositional logic, is as described below.

Sentence → Atomic sentence | Complex sentence.

Atomic sentence → True | False | Symbol.

Symbol → P | Q | R | ....

Complex sentence → ¬ Sentence

| (Sentence ∧ Sentence)

| (Sentence ∨ Sentence)

| (Sentence ⇒ Sentence)

| (Sentence ⇔ Sentence).

Notice that grammar is very strict about parentheses. Every sentence constructed with binary connectives must be closed in parentheses.

The order of precedence in propositional logic is from highest to lowest. The operators with their precedence order, are shown below,

¬ , ∧ , ∨ , ⇒ , ⇔.

Hence, the sentence,

¬ X ∨ Y ∧ Z ⇒ W

is equivalent to,

((¬ X) ∨ (Y ∧ Z)) ⇒ W.

5.3.6 Inference in the Knowledge Base

A logical inferencing means to decide whether KB ⊨ α for some sentence α.

General inferencing algorithm (truth table enumeration algorithm)

1) This algo is used for deciding entailment in propositional logic.
2) It works like backtracking search algorithm.
3) It performs a recursive enumeration of a finite space of assignments to variables.
4) This algorithm is sound, because it implements directly the definition of entailment.
5) This algorithm is complete, because it works for any knowledge base and 'α', and always terminates (As there are finite models to check).
6) Time complexity of algorithm is $O(2^n)$ (assuming n symbol in all, then there are 2^n models).
7) Space complexity of algorithm is O(n), because the enumeration is depth first.

Note Every known inference algorithm for propositional logic has a worst case time complexity which is exponential in the size of the input.

Note Propositional entailment is CO-NP-complete.
(CO-NP stands for Complement - Nonpolynomial time, where as CO-NP-complete means hardest problem in CO-NP)

The functions for truth table enumeration algorithm :

Function TT Entails (KB, α) Returns True or False

Inputs : (KB, the knowledge base),

(a sentence in propositional logic α),

the query, a sentence in propositional logic.

Symbols ← A list of the proposition symbols in KB and α

return TT CHECK-ALL (KB, α, symbols, [ ])

Function TT-CHECK-ALL (KB, α, symbols, model) returns true or false

```
if EMPTY (symbols) then
  if PL-TRUE (KB, model) then return
       PL-TRUE (α, model)
  else return true
else do
P ← FIRST (symbols); rest ←
       REST (symbols)
return TT-CHECK-ALL (KB, α, rest,
       EXTEND (P, true, model) and
TT-CHECK-ALL(KB, α, rest, EXTEND
       (P, false, model)
```

Note that,

1) In truth-table enumeration algorithm for deciding propositional entailment, TT stands for truth Table.
2) PL-TRUE returns true; if a sentence holds within a model.
3) The variable model represents a partial model-an assignment to only some of the variables.
4) The function call EXTEND (P, true, model) returns a new partial model in which P has the value true.

5.4 Semantics for Propositional Logic

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

The semantics for propositional logic should specify how to compute the truth value of any sentence, given a model. This is done recursively. All sentences are constructed from atomic sentences and the five connectives.

5.4.1 Interpretation of Propositions

1) An interpretation of a set of propositions is the assignment of a truth value, either T or F, to each propositional symbol.
2) The symbol true is always assigned T.
3) The symbol false is assigned F.
4) Propositions :
   i) A proposition is a statement which in English would be declarative sentence.
   ii) Every proposition is either true or false.
   iii) Propositions are sentences, either true or false but not both.
   iv) Sentence is smallest unit in propositional logic.
   v) If a proposition is true then its truth value is "True".
   vi) If a proposition is false then its truth value is "False".

5.4.2 Computation of Truth Values

1) The truth assignment of negation, $\neg$ P, where P is any propositional symbol, is F, if the assignment to P is T, and T if the assignment to P is F.
2) The truth assignment of conjunction, $\wedge$, is T only when both conjuncts have truth value T; otherwise it is F.
3) The truth assignment of disjunction, $\vee$, is F only when both disjuncts have truth value F, otherwise it is T.
4) The truth assignment of implication, $\rightarrow$ is F only when premise or symbol before the implication is T and the truth value of the consequent or symbol after the implication is F; otherwise it is T.
5) The truth assignment of equivalence, $\equiv$ is T only when both expressions have the same truth assignment for all possible interpretations; otherwise it is F.

For example :

Sentence	Truth value	Proposition (Y/N)
i) "Grass is green"	"True"	Yes
ii) "2 + 5 = 5"	"False"	Yes
ii) "Close the door"	-	No
iv) "Is it hot out side ?	-	No
v) "X > 2" where X is variable	-	No (Since X is not defined)
vi) "X = X"	-	No (It is unknown that what is 'X' and "="; "3 = 3" or "air is equal to air or "water is equal to water" has no meaning).

5.4.3 Truth Table

1) The truth assignments of compound propositions are often described by truth tables.
2) A truth table lists all possible truth value assignments to the atomic propositions of an expression and gives the truth value of the expression for each assignment.
3) Thus, a truth table enumerates all possible words of interpretation that may be given to an expression.

For example :

The truth table for $P \wedge Q$, lists truth values for each possible truth assignment of the operands. $P \wedge Q$ is true only when both P and Q are both T or ($\vee$), not ($\neg$), implies ($\rightarrow$), equivalence ($\equiv$) are defined in a similar fashion.

- **Truth Table demonstrating the equivalence of $P \rightarrow Q$ and $\neg P \vee Q$.**

P	Q	$\neg$ P	$\neg P \vee Q$	$P \Rightarrow Q$	$(\neg P \vee Q) \equiv (P \Rightarrow Q)$
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

4) Equivalence of two sentences in propositional logic :

Two expressions in the propositional calculus are equivalent if they have the same value under all truth value assignments.

By demonstrating that two different sentences in the propositional calculus have identical truth tables, we can prove the following equivalences for propositional expressions P, Q and R.

i) $\neg (\neg P) \equiv P$ (Double-negation elimination)

ii) $(P \vee Q) \equiv (\neg P \rightarrow Q)$

iii) The contrapositive low :

$(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$

iv) De Morgan's Law :

$\neg (P \vee Q) \equiv (\neg P \wedge \neg Q)$ and $\neg (P \wedge Q) \equiv (\neg P \vee \neg Q)$

v) The commutative law :

$(P \wedge Q) \equiv (Q \wedge P)$ and $(P \vee Q) \equiv (Q \vee P)$

vi) The associative law :

$((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$

vii) The associative law :

$((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$

viii) The distributive law :

$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

ix) The distributive law :

$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

x) Implication elimination :

$(P \rightarrow Q) \equiv (\neg P \vee Q)$

xi) Biconditional elimination :

$(P \Leftrightarrow Q) \equiv ((P \rightarrow Q) \wedge (Q \rightarrow P))$

5.4.4 Validity

A sentence is valid if it is true in all models.

For example :

(Parrot is green $\vee$ Parrot is not green) is valid, one of the two holds.

Valid sentence is called Tautology. They are necessarily true.

5.4.5 Satisfiability

A sentence is satisfiable if it is true in some model.

If a sentence S is true in model m, then we say that m satisfies S or m is a model of S.

5.4.6 A Complete Example of Knowledge-based Agent

A wumpus world agent :

Wumpus World is a classic artificial intelligence problem, which is used to demonstrate various aspects based on simulation, as well as other AI concepts.

Wumpus was an early computer game in which an agent had to explore a cave made up from a series of interconnected rooms. In one of the rooms in the cave, there was a Wumpus which would kill the agent if it entered that room. Some rooms contained pits, and the agent would die if it entered any of those rooms too. The agent had one arrow with which it could kill the Wumpus. The goal was to locate the gold that was hidden some where in the cave and return to the start without getting killed.

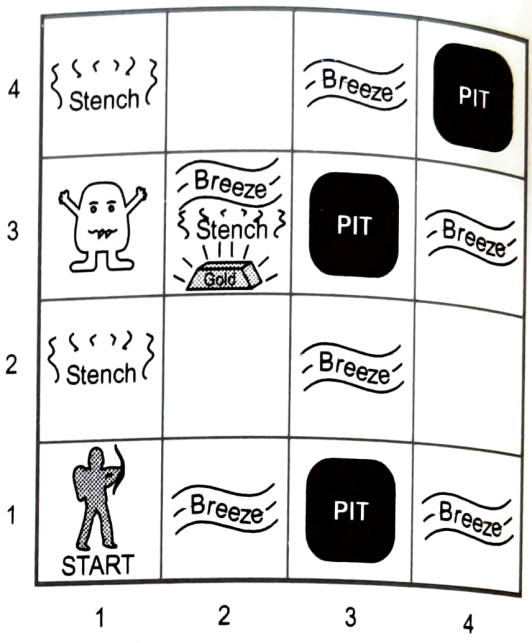

Fig. 5.4.1 A wumpus world agent and its environment

PEAS description :

- **Performance measure :**
  Gold : + 1000, **Death :** – 1000
  – 1 per step, – 10 for using the arrow.
- **Environment :**
  - Squares adjacent to Wumpus are **smelly**.
  - Squares adjacent to pit are **breezy**.
  - **Glitter** iff gold is in the same square.
  - Shooting kills Wumpus if you are facing it. It **screams**.
  - Shooting uses only arrow.
  - Grabbing picks up gold if in same square.
  - Releasing drops the gold in same square.
  - You **bump** if you walk into a wall.
- **Actuators :**
  Left turn, right turn, forward, grab, release, shoot.
- **Sensors :**
  Stench, breeze, glitter, bump, scream.

Wumpus world characterization :

1) Deterministic

   Yes - outcomes exactly specified.

2) Static

   Yes - Wumpus and Pits do not move.

3) Discrete

   Yes

4) Single - agent

   Yes - Wumpus is essentially a natural feature.

5) Fully observable

   No - only **Local** perception.

Exploring the wumpus world :

1) The knowledge base initially contains the rules of the environment.

2) Location : [1, 1]

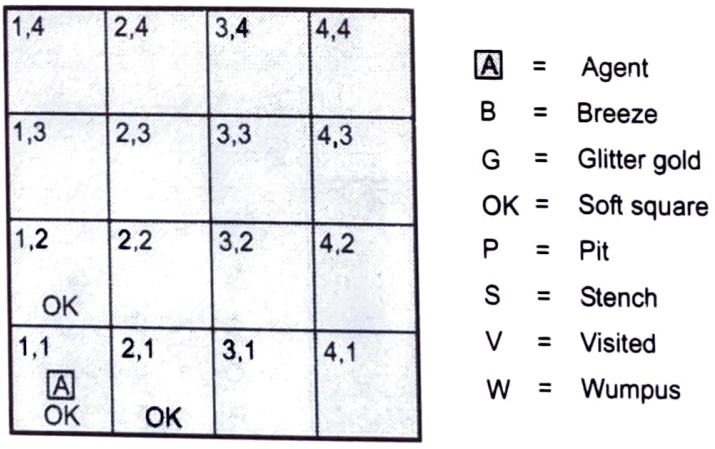

Fig. 5.4.2 Exploring wumpus world-I

Percept : [¬ Stench, ¬ Breeze, ¬ Glitter, ¬ Bump, ¬ Scream]

Action : Move to safe cell [2, 1].

3) Location : [2, 1]

Percept : [¬ Stench, Breeze, ¬ Glitter, ¬ Bump, ¬ Scream].

Infer : Breeze indicates that there is a pit in [2, 2] or [3, 1].

Action : Return to [1, 1] to try next safe cell.

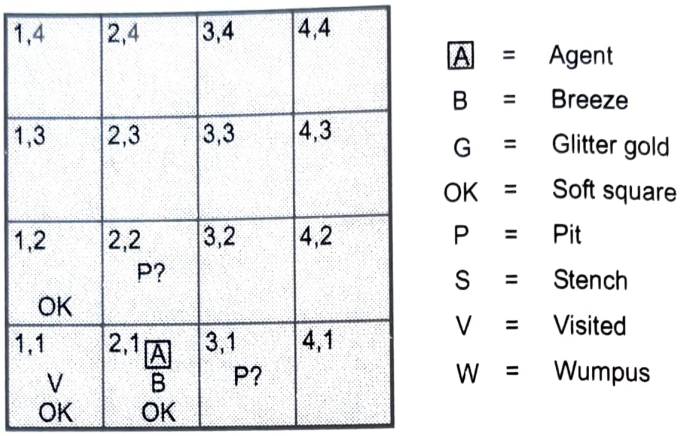

Fig. 5.4.3 Exploring wumpus world - II

4) Location : [1, 2]

Percept : [Stench, ¬Breeze, ¬ Glitter, ¬ Bump, ¬ Scream].

Infer : Wumpus in [1, 3] or [2, 2]

YET ..... not in [1, 1]

Thus .... not in [2, 2] or stench would have been detected in [2, 1]

Thus .... Wumpus is in [1, 3] [2, 2] is safe because of lack of breeze in [1, 2]

Thus .... pit in [3, 1]

Action : Move to next safe cell [2, 2].

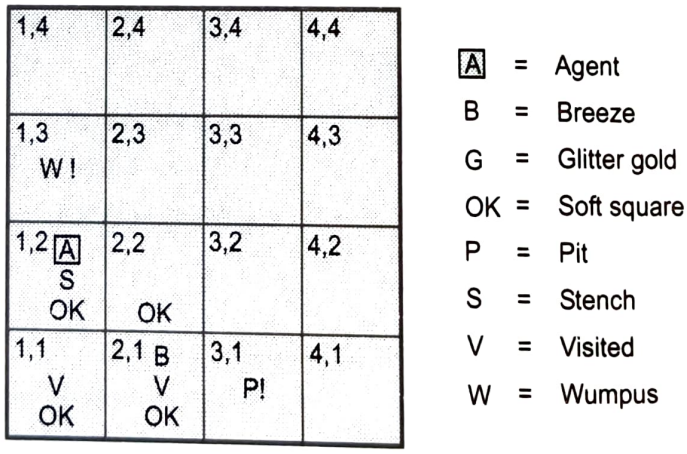

Fig. 5.4.4 Exploring wumpus world - III

5.5 Reasoning Patterns in Propositional Logic

There are standard patterns that can be applied to derive chains of conclusions that lead to the desired goal. These pattern of inference are called inference rules.

5.5.1 The Concept of Monotonicity

1) It says that the set of entailed sentences can only increase as information is added to the knowledge base.

2) Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base. The conclusion of the rule must follow regardless of what else is in the knowledge base.

5.5.2 Inference Rules

1) Modus Ponens

It is represented as,

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

It means that whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given then the sentence β can be inferred.

For example -

If two statements,

i) (Signal Pole Ahead $\wedge$ Signal Red) $\Rightarrow$ Stop

ii) (Signal Pole Ahead $\wedge$ Signal Red) are given then "Stop" can be inferred.

2) And-elimination : It says that from conjunction any conjuncts can be inferred.

It is represented as,

$$\frac{\alpha \wedge \beta}{\alpha}$$

For example -

From the sentence,

(Signal Pole Ahead $\wedge$ Signal Red)

The inference,

"Single Red".

can be drawn.

3) Unit resolution [$P \vee Q, \neg Q \vdash P$] :

Unit resolution rule takes a clause - a disjunction of literals and a literal and produce a new clause.

Rule is,

$$\frac{l_1 \vee \ldots \vee l\,k,\ m}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l\,k}$$

where each l is literal. l_i and m are complementary literals. [It means that one is the negation of other].

4) Resolution : The unit resolution rule can be generalized to the full resolution rule,

$$\frac{l_1 \vee \ldots \vee l_k, m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where l_i and m_j are complementary literals. If we are dealing only with clauses of length two we could write this as

$$\frac{l_1 \vee l_2, \neg l_2 \vee l_3}{l_1 \vee l_3}$$

That is, resolution takes two clauses and produces a new clause containing all the literals of the two original clauses except the two complementary literals.

It is a single inference rule, that yields a complete inference algorithm when coupled with any complete search algorithm.

Conjunctive normal form

1) The resolution rule applies only to disjunctions of literals, so it would seem to be relevant only to knowledge bases and queries consisting of such disjunctions.
2) Every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals.
3) A sentence expressed as a conjunction of disjunctions of literals is said to be in Conjunctive Normal Form (CNF).
4) It is used for representing sentences.

Steps for converting propositional logic sentences into CNF

1) Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$:

   For example -

   Consider following statement,

   $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

   After eliminating $\Leftrightarrow$ in above sentence,

   $(B_{1,1} \rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1})$

2) Eliminate $\rightarrow$, replacing $\alpha \rightarrow \beta$ with $\neg \alpha \vee \beta$:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2}) \vee P_{2,1} \vee B_{1,1})$

3) CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following standard equivalences : -

$\neg (\neg \alpha) \equiv \alpha$ (clouble - negation elimination)

$\neg (\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$ (De Morgan)

$\neg (\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$ (De Morgan)

In the example, we require just one application of the last rule.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \wedge P_{2,1}) \vee B_{1,1})$

4) Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law, distributing $\vee$ over $\wedge$ wherever possible.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \wedge (\neg P_{2,1} \vee B_{1,1}) \vee B_{1,1})$

Above sentence is now in CNF.

Resolution rule forms the basis for a family of complete inference procedures.

- **Refutation completeness :**

It means that resolution can always be used to either confirm or refute a sentence but it cannot be used to enumerate true sentences.

Resolution algorithm

1) It takes input as knowledge base [a sentence in propositional logic] and α (it is query - which is a sentence in propositional logic).
2) Its output is true or false means that knowledge base do resolves α. It shows that knowledge base $\models \alpha$, for this it will be shown that $(KB \wedge \neg \alpha)$.
3) Steps are as follows,
   i) $(KB \wedge \neg \alpha)$ is first converted in to CNF.
   ii) Resolution rule is applied to the resulting clauses.
   iii) Each pair that contains complementary literals is resolved to produce new clause.
   iv) This new clause is added to set if it is not already present.
   v) Step (ii) to (iv) are repeated until one of the two following situations occurs.

a) There are no new clauses that can be added, in which case α does not entail β

OR

b) An application of the resolution rule derives the empty clause, in which case α entails β.

Completeness of resolution

The algorithm of resolution is complete. It means that it will surely produces a result checking all clauses in KB and all the clauses derivable from them. It finds closure set of clauses set which has all derivable clauses from existing clauses set. It checks the input query against the closure set.

Ground resolution theorem

It states that – "If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause".

The ground resolution theorem is called as completeness theorem for resolution.

5.6 Forward and Backward Chaining

5.6.1 The Horn Clause

1) Knowledge base contains simple restricted clauses called as Horn clauses. A Horn clause is a disjunction of literals of which at most one is positive.

2) Horn clause :

- It has atmost one positive literal.
- Horn clause with exactly one position literal are called definite literal.
- The positive literal is called HEAD of the clause.
- The negative literal is called as BODY of the clause.
- A definite clause with no negative literals is called as fact.
- Horn clause with no positive literals can be written as an implication whose conclusion is the literal false. Such horn clause can be used to represent integrity constraints (conditions) in the databases.

3) Inferencing with HORN clause - For inferencing with Horn clause forward and backward chaining methods are used.

5.6.2 Forward Chaining Method

The concept :

1) It is general concept of data-driven reasoning in which the focus of attention starts with known data.
2) It can be used within an agent to derive conclusions from incoming percepts.
3) It is used for determining whether a single proposition symbol Q (the query) is entailed by the knowledge base of Horn clauses.
4) It initially starts from known facts (positive literals) in the knowledge base.

5) If all the premises of an implication are known then its conclusion is added to the set of known facts.

For example :

If $L_{1,1}$ and breeze are known and $(L_{1,1} \wedge \text{Breeze}) \Rightarrow B_{1,1}$ is in the KB then $B_{1,1}$ can be added.

6) This process continues until the query Q is added or until no further inferences can be made.

Forward chaining algorithm :

```
Function PL-FC-ENTAILS ? (KB, q)
        returns true or false
Inputs : KB, the knowledge base, a set of propositional Horn clauses, q, the query, a
   proposition symbol.
Local variables :  Count, a table, indexed by clause, initially the number of premises.
                   inferred, a table, indexed by symbol, each entry initially false.
                   agenda, a list of symbols, initially the symbols known to be true in KB.
while agenda is not empty do
   p ← POP (agenda)
unless inferred [p] do
   inferred [p] ← true
For each Horn clause (in whose premise p appears do decrement count [c])
   if count [c] = o then do
      if HEAD [c] = q then return true
      PUSH (HEAD [c], agenda)
return false.
```

- **Properties of forward chaining algorithm**

1) Forward chaining algorithm is sound
   - It means that every inference is essentially an application of modus ponens.
2) Forward chaining is complete.
   - It means that every entailed atomic sentence will be derived. Algorithm constructs inferred table for final state. In this table each symbol inferred during process has true value and false for all other symbols.
3) For example :

   Consider a simple knowledge base of Horn clause, as given below.

   1. $A \Rightarrow Q$
   2. $B \wedge C \Rightarrow A$

3. $D \wedge B \Rightarrow C$
4. $E \wedge A \Rightarrow B$
5. $E \wedge D \Rightarrow B$
6. E
7. D

AND - OR graph for above Horn clause knowledge base

1) Multiple links joined by arc indicate a conjunction.
2) Multiple links without an arc indicate a disjunction.
3) The links should be proved.
4) Inforce propagates up the graph.
5) Before proceeding ahead, from conjunction point, the propagation waits until all the conjuncts are known.

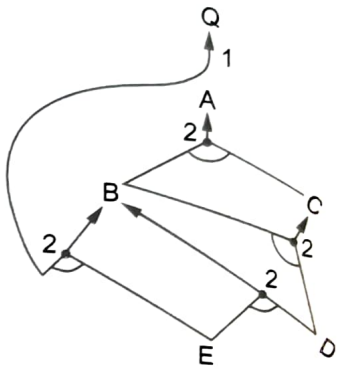

Fig. 5.6.1 AND-OR graph for Horn clause KB

- **The order of clauses considered by forward chaining algorithm for inferencing query Q :**
  1) E and D are given true.
  2) E and D implies B also B and A implies B.
  3) D and B implies C.
  4) B and C implies A.
  5) A implies Q.
  6) Therefore Q is true.

5.6.3 The Backward Chaining Algorithm

The concept :

1) It works backwards from the query i.e if query Q is known to be true then no work is needed.
2) Algorithm finds those implications in the knowledge base that conclude Q.

   If all the premises of one of those implications can be proved true (by backward chaining) then Q is true.
3) Backward chaining works down the graph starting at Q until it reaches a set of known facts that forms the basis for a proof.
4) Backward chaining is a form of goal-directed reasoning.

5) It is useful for answering questions such as "what action should I do now" ?

6) Cost of backward chaining is equal to or less than linear in the size of knowledge base, [As the process touches only relevant facts].

The example - [Referring to same knowledge base considered in forward chaining]

7) The order of clauses considered by backward chaining algorithm for inferecing query Q (in backward style) ---

1. Q is implied by A.
2. A is implied by B and C.
3. B is implied by E and A as well as E and D.
4. C is implied by E and D.
5. E and D are given true in knowledge base.
6. Therefore from Q we found implications sequence that conclude Q.

Therefore Q is true.

5.7 Effective Propositional Inference

For better inferences we need efficient algorithm which are based on model checking. Algorithm approach can be backtracking search or hill-climbing search. The algorithms we are going to discuss are for checking satisfiability.

5.7.1 A Complete Backtracking Algorithm [Davis Putnam Logmann-Loveland Algorithm-(DPLL)]

The concept :

1) It takes input as a sentence in conjunctive normal form, which is a set of clauses.

2) It do recursive depth first enumeration of possible models.

3) It determines if input propositional logic sentence (in CNF) is satisfiable.

DPLL- A improved algorithm

This algorithm is improvement over earlier general inferencing algorithm for proposition logic (The TT - Entails algorithm). There are three improvements over truth table enumeration (TT - entails algorithm) as described below,

1) Early termination :

i) A clause is true if any literal is true.

ii) A sentence is false if any clause is false.

2) Pure symbol heuristic :

i) **Pure symbol :** Pure symbol always appears with the same "sign" in all clauses.

Example :

In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.

ii) Make a pure symbol literal true.

3) Unit clause heuristic :

i) Unit clause : It is a clause which have only one literal in the clause.

ii) The only literal in a unit clause must be true.

Example :

If the model contains B = false, then $(B \vee \neg C)$ becomes a unit clause because it is equivalent to $(\text{false} \vee \neg C)$ or just $\neg C$.

The DPLL algorithm for checking satisfiability of a sentence in propositional logic

```
Function DPLL-SATISFIABLE (S) returns true or false
Inputs : S, a sentence in propositional logic.
Clause ← The set of clauses in the CNF representation of S.
symbols ← A list of the proposition symbols in S.
return DPLL (clauses, symbols, [ ])
```

DPLL algorithm (the searching procedure)

```
Function DPLL (clauses, symbols, model)
    returns true or false
if every clause in clauses is
    true in model then return true
if some clause in clauses is false
    in model then return false
P, value ← FIND-PURE-SYMBOL (symbols, clauses, model)
if P is non-null then return DPLL
    (clauses, symbols, -P, EXTEND (P, value, model)
P value ← FIND-UNIT-CLAUSE (clauses, model)
if P is non-null then return DPLL
    (clauses, symbols-P, EXTEND (D, value, model)
P ← FIRST (symbols); rest ← REST (symbols)
    return DPLL (clauses, rest,
    EXTEND (P,true, model)) or DPLL (clauses, rest, EXTEND (P, false, model)).
```

Notes i) FIND-PURE-SYMBOL is a function which returns pure symbol (with value) or null.

ii) FIND-UNIT-CLAUSE is a function which returns a unit clause (with value) or null.

5.8 Local Search Algorithms for Inferencing in Propositional Logic

The local search algorithm like Hill-climbing and simulated-annealing can be applied directly to satisfiability problems. They work properly when correct evaluation function is provided to them. The task of algorithm is to find an assignment that satisfies every clause. Therefore, an evaluation function that counts the no. of unsatisfied clauses will be a good evaluation function choice.

5.8.1 Local Search Algorithms Characteristics

1) These algorithm take steps in the space of complete assignments, flipping the truth table value of one symbol at a time.
2) The space usually contains many local minima, to escape from which various forms of randomness are required.
3) The algorithm is expected to achieve balance between greediness and randomness.

5.8.2 Local Search Algorithm for Checking Satisfiability

The WALKSAT algorithm

[It is a **"random walk"** algorithm implementation for satisfiability].

Characteristics of WALKSAT algorithm :

1) It is local search algorithm.
2) Evaluation function :

   The evaluation function is the min-conflict heuristic of minimizing the number of unsatisfied clauses.
3) It balances between greediness and randomness.
4) It is more useful when we expect a solution to exist. for e.g. 8 queen problem.
5) It is incomplete algorithm. It return satisfying model when it succeed but when it fails it has problem. When algorithm fails to satisfy the sentence it may enter into infinite loop because of infinite values of max-flips.
6) Local search algorithm like WALKSAT, cannot always detect unsatisfiability, which is very necessary for deciding entailment.

For example :

An agent cannot reliably use local search to prove that a square is safe in the Wumpus world.

5.8.3 Steps in WALKSAT Algorithm

```
Function WALKSAT (Clauses, p, max, flips) returns a satisfying model or failure.
    Inputs : Clauses, a set of clauses in propositional logic. p, the probability of choosing to

    do a "random walk" move, typically around 0.5 max-flips, number of flips allowed
            before giving up.
    Model ← A random assignment of true/false to the symbols in clauses.
        For i=1 to max-flips do
    if model satisfies clauses then return model
    with probability p flip  the value in
        model of a randomly selected
        symbol from clause
    else flip whichever symbol in clause
        maximizes the number of satisfied clauses
return failure.
```

5.8.4 Hard Satisfiability Problems

Easy problems can be solved using any old algorithm but we need algorithm that can solve hard problems.

If we look at satifiability problems in CNF then we can have a under constrained (fixed and minimum constraints with some initial assignments to variables) problem with few clauses.

For example :

Following is 3-CNF sentence [in 3-CNF each clause contains three randomly generated distinct symbols] with five symbols and five clauses.

$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C)$

$\wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

Here 16 of the total 32 possible assignments are model of this sentence therefore on an average it would just take two random guesses to find a model.

It can be stated that,

if m = Number of clauses

n = Number of symbols.

Then we can calculate probability of satisfiability as the ratio m/n,

That is,

$$\text{Probability of satisfiability} = \frac{\text{Number of clauses}}{\text{Number of symbols}}$$

As we can see that, for above 3 CNF sentence this probability,

(5 clauses/5 symbols =1) is exactly 1.

For small values of m and n this probability will be near to 1.

For large values of m and n this probability will be near to 0.

When the m/n = 4.3 the probability drops down at mid level and remains constant. It **means that at this stage it is "nearly satisfiable" or as well "nearly unsatisfiable". This is called as critical point while satisfying clauses.** If we are solving a problem where in critical points do come then it is a **hard problem.**

➤ **Note that :**

1) Problems near critical points (hard problems) are mush more difficult than other random problems.
2) DPLL works effectively on harder problems

   It takes averagly few thousand steps compared with $2^{50} \approx 10^{15}$ for truth table enumeration.
3) WALKSAT is much faster than DPLL for all ranges of problems.
4) Practically, for general purpose problem solutions we can combine min-conficts heuristic and random walk behaviour.

5.9 Knowledge based Agents

5.9.1 Inference based Agent in Wumpus World - Agent based on Propositional Logic

It uses inference algorithms and knowledge base like a general knowledge based agent.

Wumpus world

1) This agent reasons logically about the location of pits (P), Wumpus (W), and safe squares (S).
2) It begins with a knowledge base that states the "physics" of the Wumpus world.
3) The agent begins its exploration from a square known to be safe.
   - $R_1 : \neg P_{1,1}$
   - $R_2 : \neg W_{1,1}$

4) For every square [x, y], if the agent perceives breeze....

$-R_3 - R_{18} : B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

5) For every square [x, y], if the agent perceives stench....

$- R_{19} - R_{34} : B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

6) How do we express there is exactly one Wumpus ?

We can represent it in 2 sentences,

a) There is at least one Wumpus $R_{35} : W_{1,1} \vee W_{1,2} \vee .... \vee W_{4,4}$.

b) For any two squares, one must be Wumpus-free.

i) This should be stated for every pair of squares.

ii) [n(n-1)]/2 number of sentences.

$-R_{36} : \neg W_{1,1} \vee \neg W_{1,2}$

$-R_{37} : \neg W_{1,1} \vee \neg W_{1,3}$

:

:

$-R_{155}$:

7) We have used 64 distinct symbols in the knowledge base.

Wumpus-World-Agent program

1) ASK-Entailment computation :

TT-Entails would have to enumerate 2^{64} rows.

DPLL/WALKSAT performs better. DPLL works because it uses unit propagation heuristic. WALKSAT suffers from incompleteness.

2) For a large Wumpus world, the number of sentences in the knowledge base will be huge.

3) We could not capture the property "breeze in a square indicates pit in at least one of directly adjacent square".

Location, orientation in wumpus world

1) Can we add propositions like,

$L_{1,1} \wedge$ Facing Right $\wedge$ Forward $\rightarrow L_{2,1}$.

2) Knowledge base will entail both $L_{1,1}$ and $L_{2,1}$ by using inference rules.

3) What we intended to capture is this

$L_{1,1}^{(t)} \wedge$ Facing Right $\wedge$ Forward $\rightarrow L_{2,1}^{(t+1)}$

Facing Right $\wedge$ Turn Left $^{(t)} \rightarrow$ Facing Up $^{(t+1)}$

4) For every times step, one such statement ???

Assuming the problem is solvable in 100 times steps, then 10 thousands of additional sentences are needed.

5.9.2 Circuit based Agent in Wumpus World

These are agents which evaluate logical expression directly in the form of circuits.

Characteristics of circuit based agent

1) Reflex agent with state.
2) Percepts are inputs to sequential circuit.

   It has network of gates and registers.
3) Outputs are registers corresponding to actions.
4) Circuits are evaluated in a data flow model.
5) Value stored at each proposition symbol gives the truth value of the corresponding symbol at the current time t.
6) State estimation, is the general task of keeping track of environment state given a stream of percepts.
7) For logic based systems : Maintain a representation of the set of all logically possible world states, given axioms and percepts.
8) Basic trick : Successor-state axioms define truth of proposition at t+1 from propositions at t.
9) For example :

$$\text{Alive}^t \Leftrightarrow \text{Scream}^t \wedge \text{Alive}^{t-1}$$

$$L^t_{1,1} \Leftrightarrow (L^{t-1}_{1,1} \wedge (\neg\, \text{Forward}^{t-1} \vee \text{Bump}^t)) \vee$$

$$(L^{t-1}_{1.2} \wedge (\text{Facing Down}^{t-1} \wedge \text{Forward}^{t-1})) \vee (L^{t-1}_{2,1} \wedge (\text{Facing Left}^{t-1} \wedge \text{Forward}^{t-1}))$$

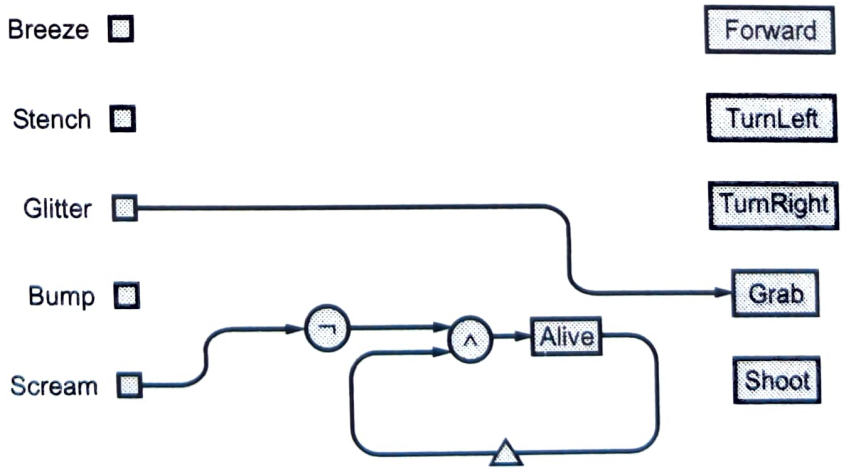

Fig. 5.9.1 Part of a circuit based agent for the wumpus world. Figure depicts circuit for grabbing the gold and the circuit for determining whether the wumpus is alive

Note
- Registers are shown as rectangles.
- One-step delays are shown as small rectangle.

Circuit based agent's location

1) One register for each L_{xy}
2) Agent is in [1, 1] at t if
   - The agent was in [1, 1] at t^{-1} and has no moved, tried but bump.
   - It was at [1, 2] facing down and moved forward.
   - It was at [2, 1] facing left and move forward.

$$L^{t}_{1,1} < > L^{t-1}_{1,1} \wedge (\neg \text{Forward}^{t-1} \vee \text{Bump}^{t}))$$

$$\vee (L^{t-1}_{1,2} \wedge (\text{Facing Down}^{t-1} \wedge \text{Forward}^{t-1}))$$

$$\vee (L^{t-1}_{2,1} \wedge (\text{Facing Left}^{t-1} \wedge \text{Forward}^{t-1}))$$

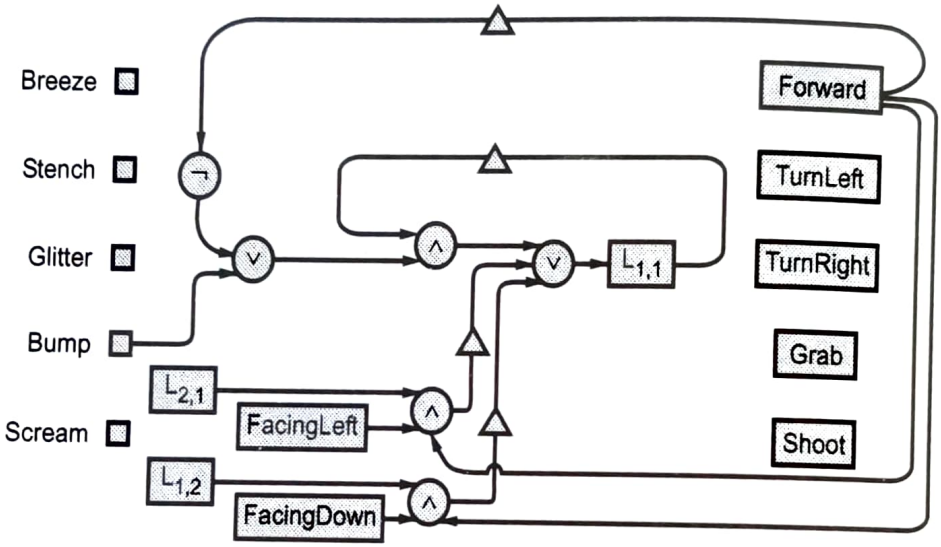

Fig. 5.9.2 The circuit for determining if the agent is at [1, 1]. Each location and orientation register has a similar circuit attached

A problem related to circuit base agent

What will be the initial truth value contained in $B_{4,4}$.

- The agent cannot set a truth value for it.
- Need to represent unknown, environment.
- Use of knowledge propositions

  $K(B_{4,4})$, $K(\neg B_{4,4})$

5.9.3 Comparison between Inference Based Agent (IBA) and Circuit Based Agent (CBA)

1) Conciseness

IBA - It require seperate copies of its knowledge base for every time step.

CBA - It do not require seperate copies of its knowledge base for every time step.

Both agents requires physics (expressed as sentences or circuits) for every square and therefore they are not suitable for large scale environment.

2) Computational efficiency

IBA - It takes exponential time for inferencing. (Time grows exponentially with respect to number of symbols)

CBA - It takes linear time for inferencing which is dependent on circuit size.

3) Completeness

IBA :

1) It is complete but with huge time requirements for completeness.
2) It also stores previous states/percept therefore memory requirement is also more.

CBA :

1) It is incomplete for various reasons :
   a) Acyclicity restriction in circuit itself which is practically not possibles.
   b) For a complete circuit CBA will take exponential time with respect to circuit size.
2) It forgets all previous states therefore can not draw conclusions based on previous states/percepts.

4) Ease of construction

IBA - As environment and conditions are simple and clear. It is easy to represent in propositional logic language.

CBA - The environment considered is limited therefore a circuit is small acyclic and semicomplete. Hence describing it is easy when relation between percepts and action is simple and straightforward. It is easy to describe and construct knowledgebase for such agents.

Answer in Brief

1. *What is propositional logic ? Explain the knowledge representation using propositional logic.* **(Refer section 5.1)**
2. *Give the five logical connectives used to construct complex sentences and give the formal grammar of propositional logic.* **(Refer section 5.3)**
3. *Explain propositional logic.* **(Refer section 5.1)**
4. *What are the limitations in using propositional logic to represent the knowledge base ?*

Ans. : Propositional logic has following limitations to represent the knowledge base.

1) It has limited expressive power.

2) It cannot directly represents properties of individuals or relations between individuals.

3) Generalizations, patterns, regularities cannot easily be represented.

4) Many rules (axioms) are required to write so as to allow inferencing.

5. *Give the five logical connectives used to construct complex sentences and give the formal grammar of propositional logic.* **(Refer section 5.1)**

6. *Write the algorithm for deciding entailment in propositional logic.* **(Refer section 5.1)**

5.10 University Question with Answer

Summer - 20

Q.1 *Define propositional logic.* **(Refer section 5.1)** **[1]**

❑❑❑