ARTIFICIAL INTELLIGENCE

CHAPTER – 7 Logical Agents

Knowledge and reasoning are important for artificial agents because they enable successful behaviors that would be very hard to achieve otherwise. Knowledge and reasoning also play a role in dealing with partially observable environments. A knowledge based agent can combine general knowledge with current percepts to infer hidden aspects of the current state prior to selecting actions.

Knowledge based agent:

The central component of a knowledge based agent is its knowledge base or KB. A knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language. For adding new sentences to the knowledge base and for querying, we use tasks called TELL and ASK respectively. Both these tasks involve inference that is deriving new sentences from old.

In logical agents, inference must obey the requirement that when one ASKS a questions of the knowledge base, the answer should follow from what has been told.

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

A knowledge based agent program takes a percept as input and returns an action. The agent maintains a knowledge base KB which may initially contains some background knowledge.

MAKE – PERCEPT – SENTENCE takes a percept as and a time and returns a sentence asserting that the agent perceived the percept at the given time.

MAKE – ACTION – QUERY takes a time as input and returns a sentence that asks what action should be performed at that time.

Wumpus World:

The Wumpus World is a cave consisting of rooms connected by passageways. There is a beast called wumpus inside the cave that eats anyone who enters its room. The wumpus can be shot by

an agent but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into the room. The only feature or goal is to find a heap of glittering gold.

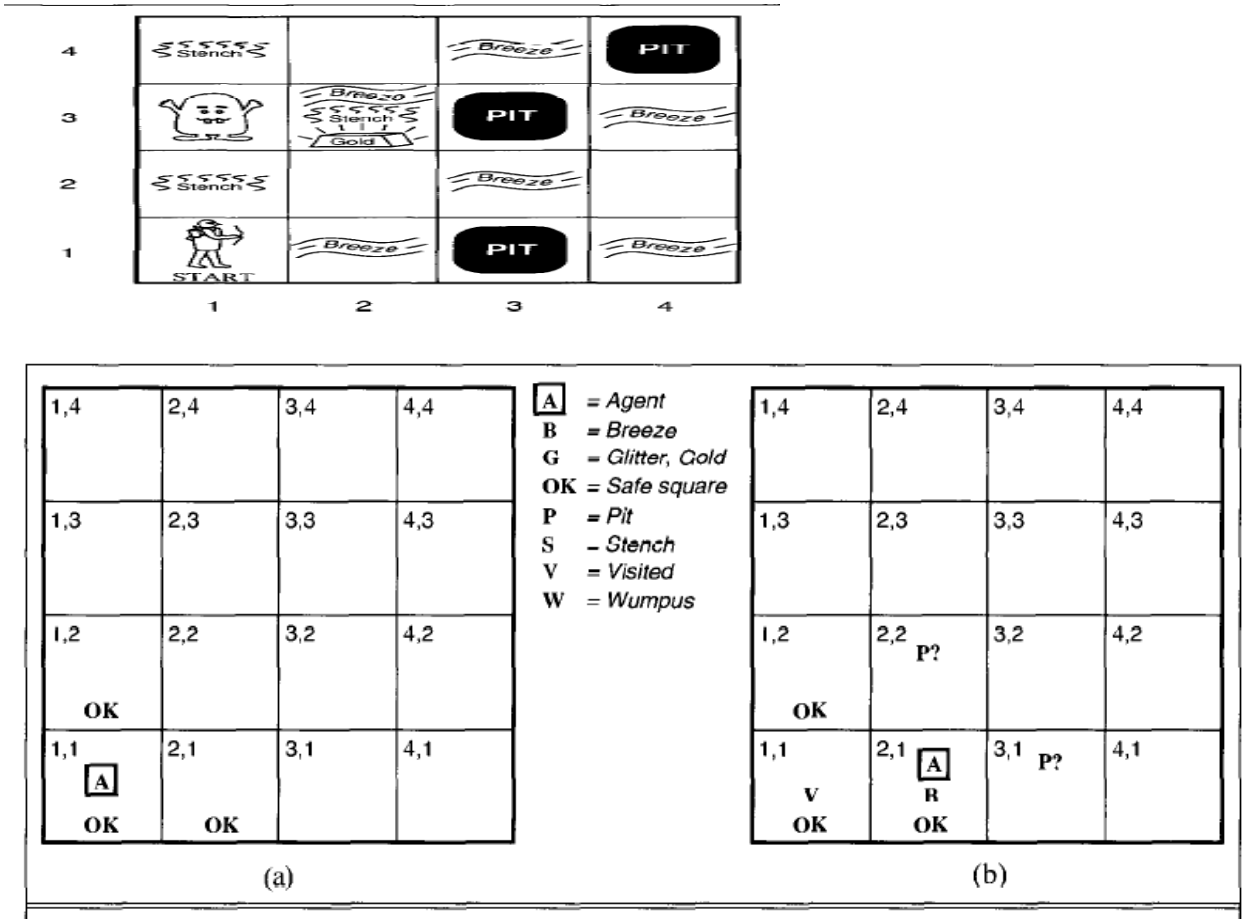
Task environment is given as –

a) Performance measure b) Environment c)Actuators d)Sensors

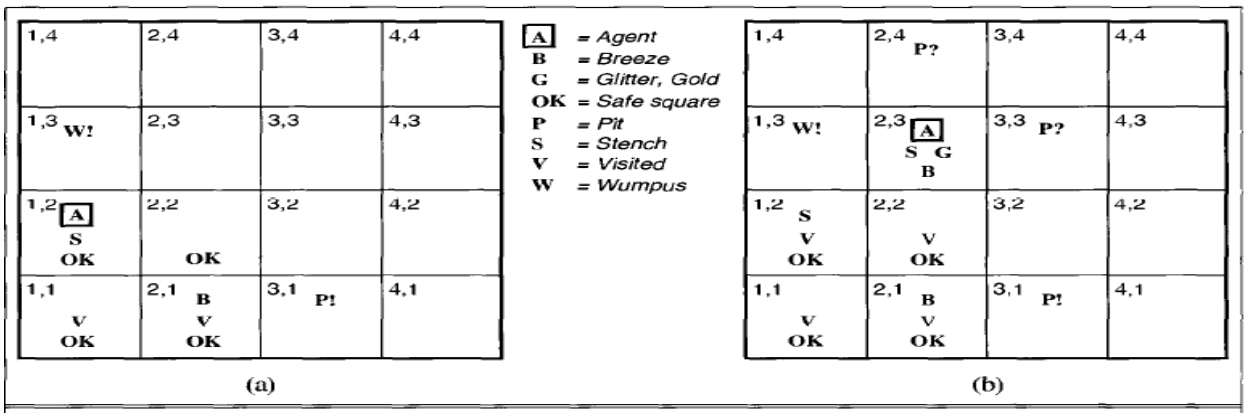
The agent has five sensors each of which gives information –

1. In the square containing the wumpus and in the directly adjacent squares the agent will perceive stench.
2. In the squares directly adjacent to the pit, the agent will perceive breeze.
3. In the square where the gold is, the agent will perceive glitter.
4. When an agent walks into a wall, it will perceive a bump.
5. When the wumpus is killed, it emits a woeful scream that can be perceived anywhere in the cave.

The agent's initial knowledge base contains the rules of the environment. The first percept is [None, None, None, None, None] from which the agent can conclude that its neighboring squares are safe.



Since there was no stench or breeze in [1, 1], the agent can infer that [1, 2] and [2, 1] are free of dangers. Hence, they are marked with OK. The agent then decides to move forward to [2, 1] where it detects a breeze so it concludes that there is a pit in a neighboring square. At this point, there is only one known square that is OK and has not been visited yet. So the prudent agent will turn around, go back to [1, 1] and then proceed to [1, 2]. The new percept is [stench, none, none, none, none]. The stench in [1, 2] means that there must be a wumpus nearby. Wumpus cannot be in [1, 1] by rules of the game and it cannot be in [2, 2]. Therefore the agent can infer that the wumpus is in [1, 3] denoted by W! The lack of breeze in [1, 2] implies that there is no pit in [2, 2].



We have already inferred that there must be a pit in either [2, 2] or [3, 1]. The agent has now proved that there is neither a pit nor a wumpus in [2, 2] so it is OK to move there. The agent then proceeds to [2, 3] where it detects a glitter, so it should grab the gold and thereby end the game.

Hence the fundamental property of logical reasoning is that – in each case where the agent draws a conclusion from the available information, that conclusion is guaranteed to be correct if the available information is correct.

Logic:

Knowledge base consists of sentences. These sentences are expressed according to the **syntax** of the representation language which specifies all the sentences that are well formed. A logic must also define **semantics** of the language. The semantics of the language defines the **truth** of each sentence with respect to each **possible world**. In standard logics, every sentence must be either true or false in each possible world or **model.** Model is a mathematical abstraction, each of which fixes the truth or falsehood of every relevant sentence. Logical **entailment** between sentences is an idea that a sentence follows logically from another sentence represented mathematically as $\alpha \models \beta$ which means that the sentence alpha entails the sentence beta.

Definition of entailment can be applied to derive conclusions – that is to carry out **logical inference.** We must perform a **model check** on the inference algorithm because it enumerates all possible models to check that alpha is true in all models in which KB is true.

If an inference algorithm 'i' can derive alpha from KB then $KB \vdash_i \alpha$ which is pronounced as "alpha is derived from KB by i" or "i derives alpha from KB".

An inference algorithm that derives only entailed sentences is called **sound** or **truth preserving.**

An inference algorithm is **complete** if it can derive any sentence that is entailed.

Grounding is the connection between logical reasoning processes and the real environment in which the agent exists. Agent's sensors help in creating this connection.

Propositional logic:

The syntax of propositional logic defines allowable sentences. The **atomic sentences** – the indivisible syntactic elements consist of a single **proposition symbol**. Each such symbol stands for a proposition that can be true or false. **Complex sentences** are constructed from simpler sentences using **logical connectives**.

Five connectives are used commonly:

$\neg$ (not) $\wedge$ (and) $\vee$ (or) $\Rightarrow$ (implies) $\Leftrightarrow$ (if and only if)

Backus – Naur form grammar of sentences in propositional logic –

Sentence	→	*AtomicSentence* | *ComplexSentence*
AtomicSentence	→	**True** | **False** | *Symbol*
Symbol	→	**P** | **Q** | **R** | ...
ComplexSentence	→	$\neg$ *Sentence*
	|	(*Sentence* $\wedge$ *Sentence*)
	|	(*Sentence* $\vee$ *Sentence*)
	|	(*Sentence* $\Rightarrow$ *Sentence*)
	|	(*Sentence* $\Leftrightarrow$ *Sentence*)

Every sentence constructed with binary connectives must be enclosed in parentheses. The order of precedence in propositional logic is from highest to lowest. $\neg, \wedge, \vee, \Rightarrow,$ and $\Leftrightarrow$

Semantics defines the 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 proposition symbol. The semantics for propositional logic must specify how to compute the truth value of any sentence given a model. Atomic sentences are very easy –

a) True is true in every model and False is false in every model.
b) The truth value of every other proposition symbol must be specified directly in the model.

For complex sentences, rule is –

For any sentence s and any model m, the sentence (not)s is true in m if and only if s is false in m.

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. The truth value of any sentence s can be computed with respect to any model m by simple process of recursive evaluation.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Standard logical equivalences:

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Reasoning patters in propositional logic:

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

a) Modus ponens $\alpha \rightarrow \beta , \alpha$

B

Whenever any sentences of the form α»β and α are given, then the sentence β can be inferred.

b) And – elimination $\frac{\alpha \wedge \beta}{\alpha}$

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

These rules can be used in any particular instances where they apply generating sound inferences without the need for enumerating models. Sequence of applications of inference rules is called proof.

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

Resolution:

Search algorithms are complete in the sense that they will find any reachable goal but if the available inference rules are inadequate, then the goal is not reachable – no proof exists that uses only those inference rules. Resolution is a single inference rule that yields a complete inference algorithm when coupled with any complete search algorithm.

Unit resolution inference rule –

$$\frac{l_1 \vee \ldots . l_k, \quad m}{l_1 \vee \ldots . l_{i-1} \vee l_{i+1} \vee \ldots l_k}$$

where each l is a literal and l_i and m are complementary literals.

Forward and backward chaining:

The completeness of resolution makes it a very important reference method. Real world knowledge bases often contain only clauses of restricted kind called Horn clauses. A Horn clause is a disjunction of literals of which at most one is positive.

The restriction to one positive literal is important for 3 reasons –

a) Every horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.

   Horn clauses with exactly one positive literal are called definite clauses. The positive literal is called the head and the negative literal forms the body of the clause. Definite clauses form the basis for logic programming. A horn clause with no positive literal can be written as an implication whose conclusion is literal false. Sentences with integrity constraints in the database world are used to signal errors in the data.

b) Inference with Horn clauses can be done through forward chaining and backward chaining algorithms.
c) Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base.

The forward chaining algorithm PL-FC-ENTAILS?(KB,q) determines whether a single proposition symbol q – the query is entailed by a knowledge base of Horn clauses. It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts. Forward chaining is sound: every inference is essentially an application of Modus Ponens. Forward chaining is also complete: every entailed atomic sentence will be derived.

Forward chaining is an example of the concept of data driven reasoning – that is reasoning in which the focus of attention starts with known data. It can be used within an agent to derive conclusions from incoming percepts, often without a specific query.

```
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 c in whose premise p appears do
        decrement count[c]
        if count[c] = 0 then do
          if HEAD[c] = q then return true
          PUSH(HEAD[c], agenda)
  return false
```

The back ward chaining algorithm works backward from the query. If the query q is known to be true, then nothing has to be done. Otherwise, the algorithm finds those implications in the knowledge base that conclude q. If all the premises of one of those implications can be proved true, then q is true. Back ward chaining is a form of goal directed reasoning.