CS 3011: Artificial Intelligence

Knowledge-Based Agents and Logic

Instructors: Dr. Durgesh Singh

CSE Discipline, PDPM IIITDM, Jabalpur -482005

Entailment

- Entailment means that one thing **follows** from another

$$\alpha \models \beta$$

- Sentence α entails sentence β if and only if in every world in which α is true, β is also true.
  - Sentence (or Knowledge base) α entails sentence β iff β is true in all worlds where α is true
    - E.g., the KB containing “Tom won" and “ Johan won” entails “Either the Tom won or Johan won“
    - E.g., x=0 entails xy=0

Wumpus World-A simple knowledge base

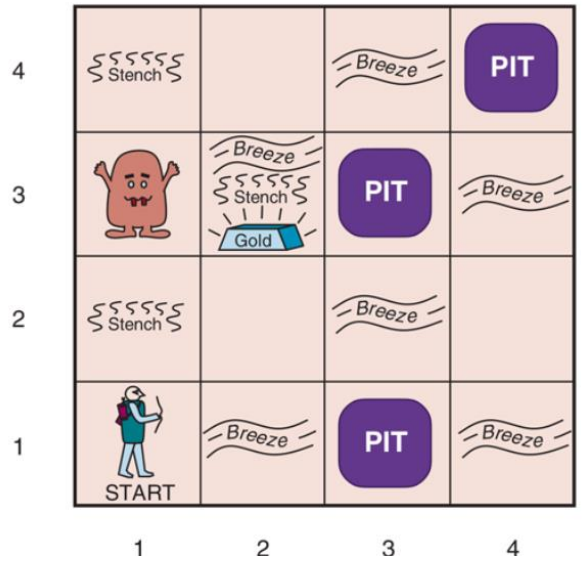
$P_{x,y}$ is true if there is a pit in $[x,y]$.

$W_{x,y}$ is true if there is a wumpus in $[x,y]$, dead or alive.

$B_{x,y}$ is true if there is a breeze in $[x,y]$.

$S_{x,y}$ is true if there is a stench in $[x,y]$.

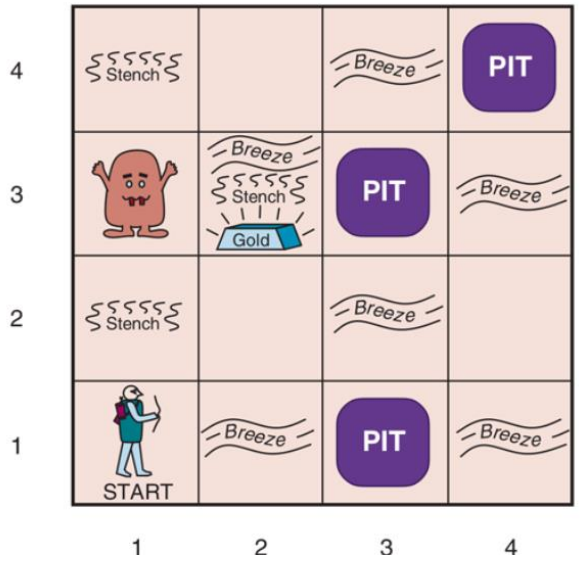
$L_{x,y}$ is true if the agent is in location $[x,y]$.



Wumpus World-A simple knowledge base

- There is no pit in [1,1]:

$$R_1 : \quad \neg P_{1,1} .$$



$P_{x,y}$ is true if there is a pit in $[x,y]$.

$W_{x,y}$ is true if there is a wumpus in $[x,y]$, dead or alive.

$B_{x,y}$ is true if there is a breeze in $[x,y]$.

$S_{x,y}$ is true if there is a stench in $[x,y]$.

$L_{x,y}$ is true if the agent is in location $[x,y]$.

Wumpus World-A simple knowledge base

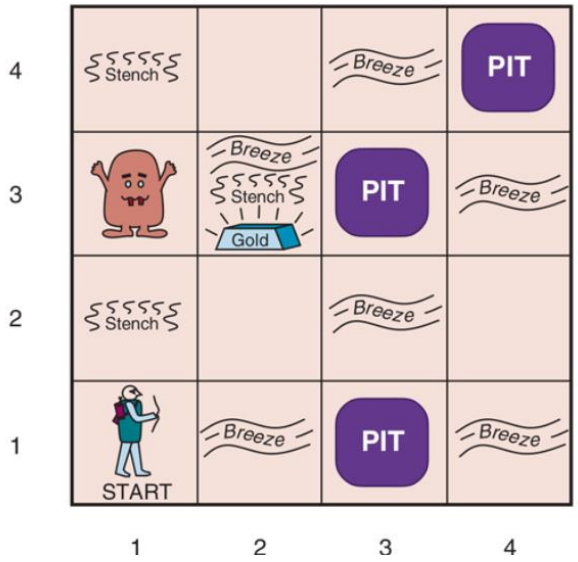
- A square is breezy if and only if

there is a pit in a neighboring square.

$$R_2 : \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}) .$$
$$R_3 : \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}) .$$

- Now we include the breeze percepts for the first two squares visited in the specific world the agent

$$R_4 : \quad \neg B_{1,1} .$$
$$R_5 : \quad B_{2,1} .$$

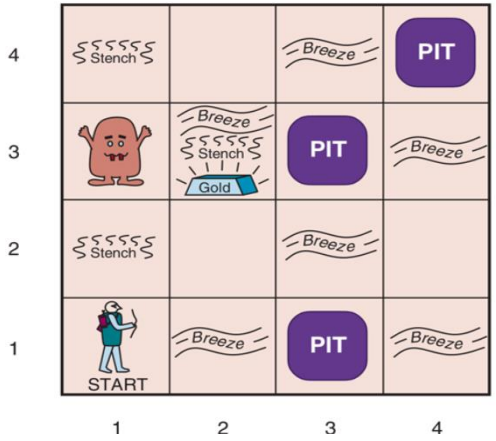


Inference

- Our goal now is to decide whether $KB \models \alpha$ for some sentence α.
- Model-checking approach that is a direct implementation of the definition of entailment: enumerate the models, and check that α is true in every model in which KB is true.
- For example, is $\neg P_{1,2}$ entailed by our KB?

KB=

$$
\begin{aligned}
R_1 &: \quad \neg P_{1,1} . \\
R_2 &: \quad B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1}) . \\
R_3 &: \quad B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1}) . \\
R_4 &: \quad \neg B_{1,1} . \\
R_5 &: \quad B_{2,1} .
\end{aligned}
$$



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

A truth table constructed for the knowledge base given in the text. KB is true if R_1 through R_5 are true, which occurs in just 3 of the 128 rows (the ones underlined in the right-hand column). In all 3 rows, $P_{1,2}$ is false, so there is no pit in [1,2]. On the other hand, there might (or might not) be a pit in [2,2].

Tautology, Contradiction, and Contingency

- A tautology is a compound proposition whose truth value is true for all possible values of its propositional variables

  e.g. $P \vee \neg P$

- A contradiction (or absurdity) is a compound proposition which is always false.

  e.g. $P \wedge \neg P$

- A contingency is compound proposition which is sometimes TRUE and sometimes False.
  - e.g. $P \wedge Q$

Two Famous Inference Rules

- Modus Ponens

$$\frac{P \Rightarrow Q \ , \ P}{Q}$$

Given that P implies Q, and I know that P is true, then I can infer Q

- And Rule or Simplification

$$\frac{P \wedge Q}{P}$$

Given that P AND Q is true, I can infer that P is true, and I can also infer that Q is true.

Sample Rules of Inference

- Modus Ponens: $\{\alpha \Rightarrow \beta, \alpha\} \vdash \beta$
- And Elimination: $\{\alpha \wedge \beta\} \vdash \alpha$; $\{\alpha \wedge \beta\} \vdash \beta$
- And Introduction: $\{\alpha, \beta\} \vdash \alpha \wedge \beta$
- Or introduction: $\{\alpha\} \vdash \alpha \vee \beta$
- Double negation Elimination: $\{\neg\neg\alpha\} \vdash \alpha$
- Implication Elimination: $\{\alpha \Rightarrow \beta\} \vdash \neg\alpha \vee \beta$
- Unit resolution: $\{\alpha \vee \beta, \neg\beta\} \vdash \alpha$
- Resolution: $\{\alpha \vee \beta, \neg\beta \vee \gamma\} \vdash \alpha \vee \gamma$

Clause

- Clause: special form of representation of propositions and very useful for inferencing.
- Literal: A single proposition or its negation
  e.g., **P** , ¬ P
- A clause is a disjunction of literals

  e.g., P ∨ Q v ¬ R
- Given any proposition can we convert it to the clausal form.
- Example

Conjunctive Normal Form

- Every sentence of propositional logic is logically equivalent to a conjunction of clauses.
- A sentence expressed as a conjunction of clauses is said to be in **conjunctive normal form** or **CNF**

Example: Convert in CNF

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

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

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

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. Move $\neg$ inwards using de Morgan's rules and double-negation:

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

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

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

Resolution

- Resolution is a technique of inference

Suppose x is a literal and S1 and S2 are two sets of propositional sentences represented in clausal form

If you have $(x \vee s1) \wedge (\neg x \vee S2)$

Then we get $s1 \vee s2$

Here, $s1 \vee s2$ is the resolvent, and x is the resolved upon

Resolution Algorithm

Algorithm works using **proof by contradiction**.

To show $KB \models \alpha$ we show that $KB \wedge \neg\alpha$ is not satisfiable

Apply resolution to $KB \wedge \neg\alpha$ in CNF

and Resolve pairs with complementary literals

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

if l_i and m_j are complimentary literals

and add new clauses

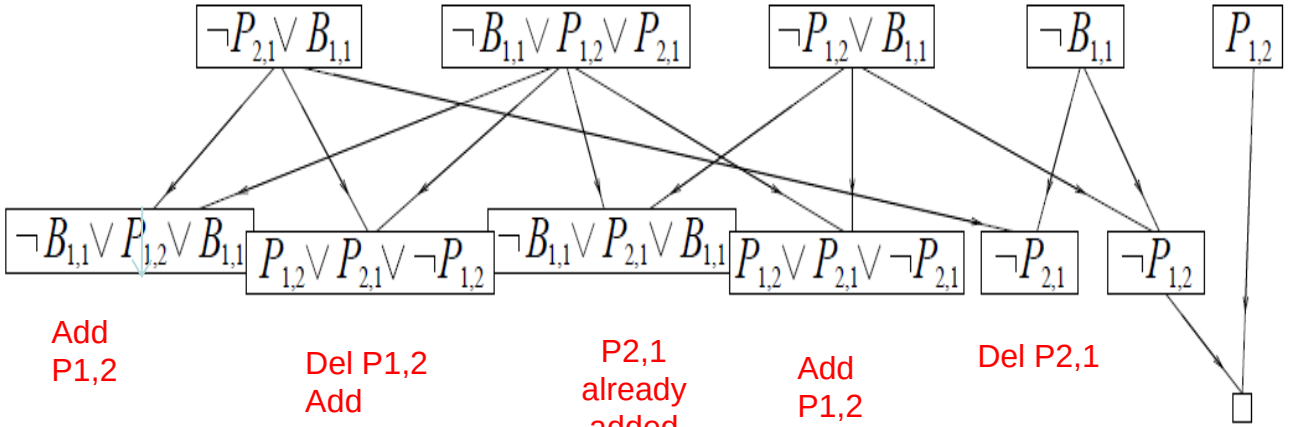
until

- there are no new clauses to be added
- two clauses resolve to the *empty* clause which means $KB \models \alpha$

Resolution Example

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

$$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \; \alpha = \neg P_{1,2}$$



Add P1,2

Del P1,2
Add P2,1

P2,1 already added

Add P1,2

Del P2,1

This returns an empty clause.
So, alpha is true.

Example

Prove R

1	P v Q
2	$P \rightarrow R$
3	$Q \rightarrow R$

Example

Proof by Resolution: If either subject EC173 or EC220 is required, then all students will take programming course. EC173 and EC240 are required. Prove that all students will take programming course.

Horn clauses and definite clauses

- **Definite clause**: disjunction of literals with exactly one positive literal
- **Horn clause:** disjunction of literals of with at most one positive literal
- Inference with Horn clauses can be done through the forward-chaining and backward-chaining algorithms

Note: Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal

Forward chaining

- It determines if a single proposition symbol q—the query is entailed by a knowledge base of definite 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 [Modus Ponens].
- This process continues until the query q is added or until no further inferences can be made.

Example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

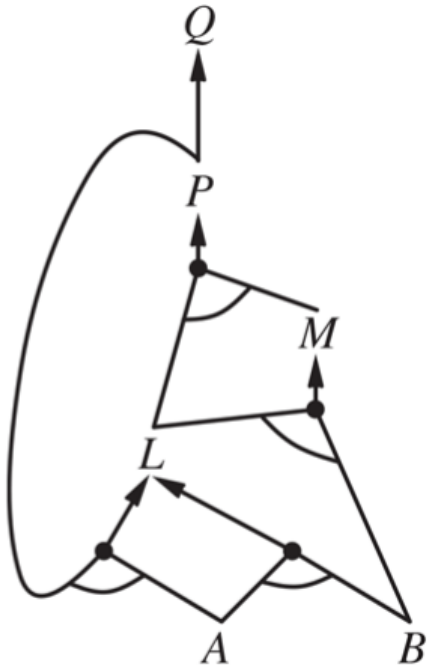
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

(a)



(b)

(a) A set of Horn clauses. (b) The corresponding AND–OR graph.

Backward-chaining

- Backward-chaining algorithm works backward from the query
- If the query q is known to be true, then no work is needed.
- Otherwise, the algorithm finds those implications in the knowledge base whose conclusion is q.
- If all the premises of one of those implications can be proved true (by backward chaining), then q is true.
- When applied to the query Q in given example, it works back down the graph until it reaches a set of known facts, A and B, that forms the basis for a proof..

Example: Backward-chaining

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

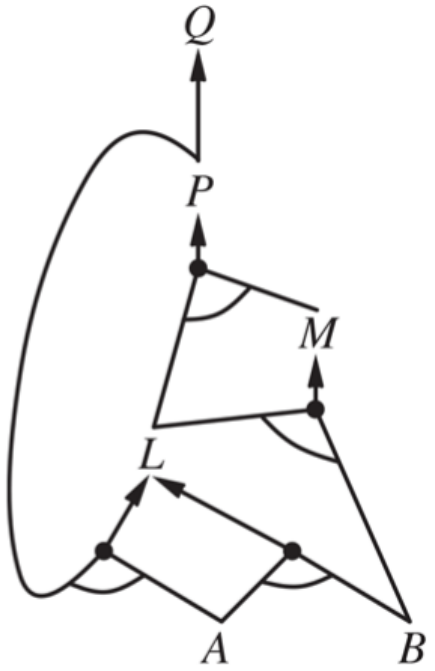
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

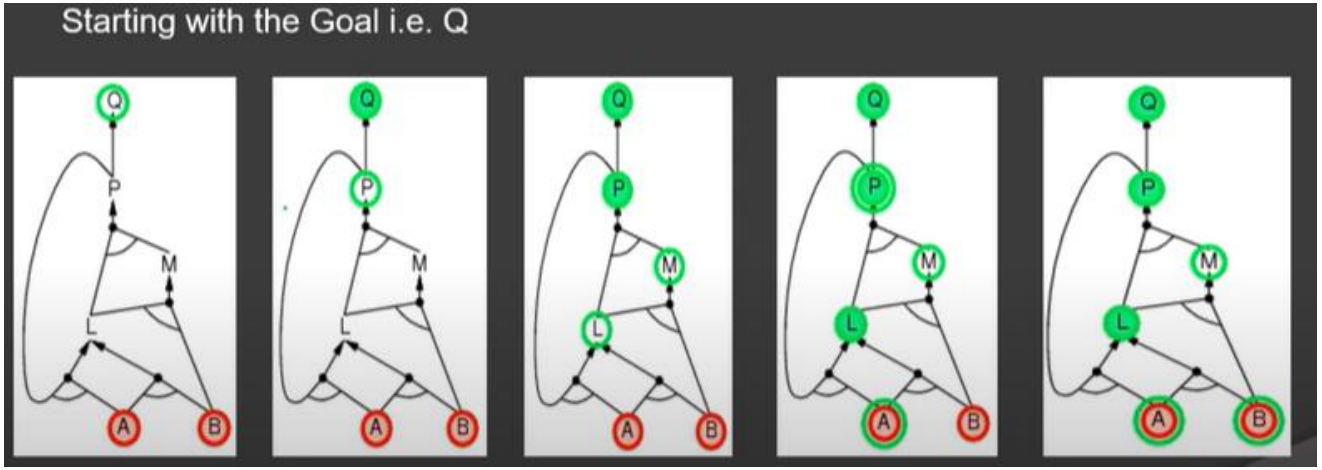
(a)



(b)

(a) A set of Horn clauses. (b) The corresponding AND–OR graph.

Backward-chaining…



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

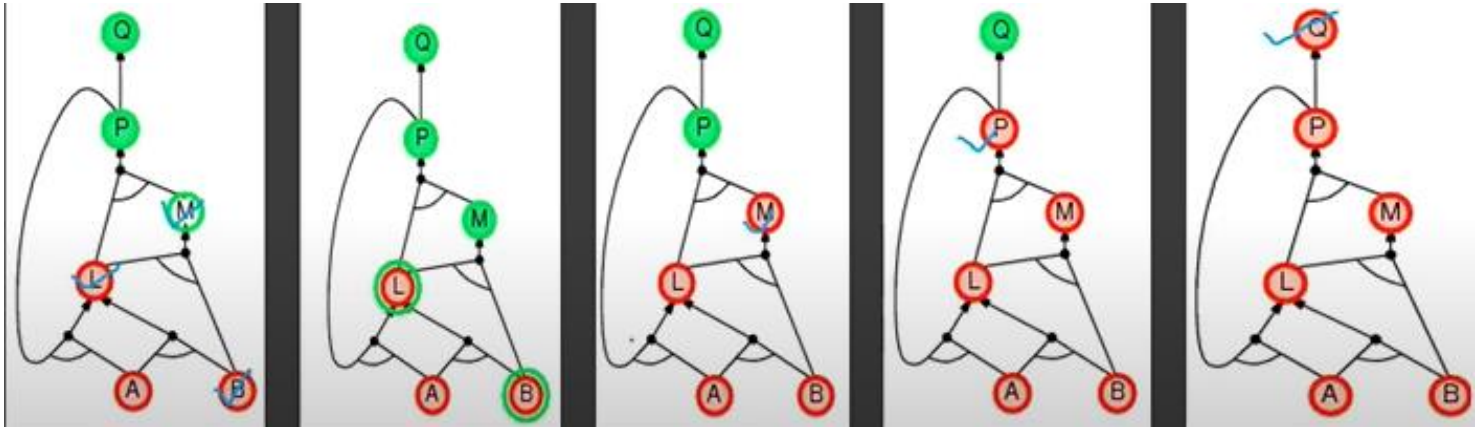
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward-chaining…



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Discussions

- As with forward chaining, an efficient implementation runs in linear time.
- Backward chaining is a form of goal-directed reasoning. It is useful for answering specific questions such as “What shall I do now?” and “Where are my keys?”
- Often, the cost of backward chaining is much less than linear in the size of the knowledge base, because the process touches only relevant facts.