Propositional Logics

Unit-IV

Propositional Logic: Proof by Resolution

- Proof by Resolution: In proof by inference, if we removed the biconditional elimination rule, the proof would not go through.
- Here a single inference rule, known as resolution, that yields a complete inference algorithm when coupled with any complete search algorithm.
- This method is basically used for proving the satisfiability of a sentence. Resolution is also called **Proof by Refutation (contradiction)** .
- Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal.
- As a result, we have to conclude that the original goal is true.
- We begin by using a simple version of the resolution rule in the wumpus world.

Propositional Logic: Proof by Resolution

- Let us consider the steps: the agent returns from [2, 1] to [1, 1] and then goes to [1, 2], where it perceives a stench, but no breeze.
- We add the following facts to the knowledge base:

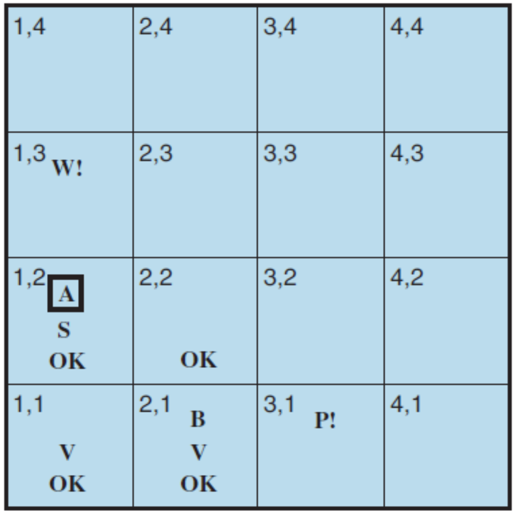
$$R_{11}: \quad \neg B_{1,2}.$$
$$R_{12}: \quad B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}).$$

- We can now derive the absence of pits in [2, 2] and [1, 3]

$$R_{13}: \quad \neg P_{2,2}.$$
$$R_{14}: \quad \neg P_{1,3}.$$

- We can also apply biconditional elimination to R_3, followed by Modus Ponens with R_5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]:

$$R_{15}: \quad P_{1,1} \vee P_{2,2} \vee P_{3,1}.$$



(a)

Figure 1. Wampus world for random state

Propositional Logic: Proof by Resolution

- Apply resolution rule:
- The literal $\neg P_{2,2}$ in R_{13} resolves with the literal $P_{2,2}$ in R_{15} to give the resolvent

$$R_{16}: \quad P_{1,1} \vee P_{3,1}.$$

- i.e., R_{15} says -- there is a pit in one of [1, 1], [2, 2] and [3, 1]. R_{13} says -- there is no pit in [2, 2].
- From these two, R_{16} says -- the pit is in [1, 1] or [3, 1].
- Now, the literal $\neg P_{1,1}$ (in R_1) resolves with the literal $P_{1,1}$ in R_{16} to give

$$R_{17}: \quad P_{3,1}.$$

- i.e., if there is a pit in [1, 1] or [3, 1] and it is not in [1, 1], then it is in [3, 1].
- These last two inference steps are examples of the unit resolution inference rule.

Propositional Logic: Proof by Resolution

- The last two inference steps are examples of the *Unit Resolution Inference Rule:*

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

- where each ***'l'*** is a literal and $\boldsymbol{l_i}$ and **m** are complementary literals (i.e., one is the negation of the other).
- Thus, the unit resolution *takes a clause (a disjunction of literals) and a literal and produces a new clause*.
- Note that a single literal can be viewed as a disjunction of one literal, also known as a unit clause.

Propositional Logic: Proof by Resolution

- The unit resolution rule can be generalized to the *full resolution rule*.

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

- where `l_i and m_j are complementary literals. This says that resolution takes two clauses and produces a new clause containing all the literals of the two original clauses except the two complementary literals.
- For example, we have

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}} .$$

- You can resolve only one pair of complementary literals at a time. For example, we can resolve P and ¬ P to deduce

$$\frac{P \vee \neg Q \vee R, \qquad \neg P \vee Q}{\neg Q \vee Q \vee R} ,$$

Propositional Logic: Proof by Resolution

- But you can't resolve on both P and Q at once to infer R.
- There is one more technical aspect of the resolution rule: the resulting clause should contain only one copy of each literal.
- **Factoring:** The removal of multiple copies of literals.
- Ex: if we resolve $(A \lor B)$ with $(A \lor \neg B)$, we obtain $(A \lor A)$, which is reduced to just A.
- The soundness of the resolution rule can be seen easily by considering the literal l_i that is complementary to literal m_j in the other clause.

Basics for Resolution Methods in AI

- Conjunctive Normal Form (CNF)
- In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. There are following steps used to convert into CNF:

1) Eliminate bi-conditional implication by replacing

   $A \Leftrightarrow B$ with $(A \rightarrow B) \wedge (B \rightarrow A)$

1) Eliminate implication by replacing $A \rightarrow B$ with $\neg A \vee B$.
2) In CNF, negation(¬) appears only in literals, therefore we move it inwards as:

   $\neg(\neg A) \equiv A$ (double-negation elimination)

   $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$ (De Morgan)

   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ (De Morgan)

4) Finally, using distributive law on the sentences, and form the CNF as:

   $(A1 \vee B1) \wedge (A2 \vee B2) \wedge \ldots. \wedge (An \vee Bn)$.

Basics for Resolution Methods in AI

- The <u>resolution rule applies only to disjunction of literals</u> (relevant to KBs and queries consisting of such disjunctions).
- In other words, a sentence expressed as a <u>*conjunction of disjunction of literals*</u> is said to be <u>in CNF (i.e.,</u> AND of ORS).

- Disjunctive Normal Form (DNF):
- This is a reverse approach to CNF. The process is similar to CNF with the following difference:

  (A1 Λ B1) V (A2 Λ B2) V,…,V (An Λ Bn).

- In DNF, it is OR of ANDs, i.e., sum of products, or a cluster concept, whereas, in CNF, it is ANDs of ORs, i.e., product of sums.

Propositional Logic: Proof by Resolution

- Conjunctive Normal Form (CNF):
- A sentence expressed as a conjunction of clauses is said to be in conjunctive normal form or CNF shown in given figure.
-

$$
\begin{aligned}
\mathit{CNFSentence} &\rightarrow \mathit{Clause}_1 \wedge \cdots \wedge \mathit{Clause}_n \\
\mathit{Clause} &\rightarrow \mathit{Literal}_1 \vee \cdots \vee \mathit{Literal}_m \\
\mathit{Fact} &\rightarrow \mathit{Symbol} \\
\mathit{Literal} &\rightarrow \mathit{Symbol} \mid \neg \mathit{Symbol} \\
\mathit{Symbol} &\rightarrow P \mid Q \mid R \mid \ldots \\
\mathit{HornClauseForm} &\rightarrow \mathit{DefiniteClauseForm} \mid \mathit{GoalClauseForm} \\
\mathit{DefiniteClauseForm} &\rightarrow \mathit{Fact} \mid (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{Symbol} \\
\mathit{GoalClauseForm} &\rightarrow (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{False}
\end{aligned}
$$

Figure 2. A grammar for conjunctive normal form, Horn clauses, and definite clauses. A CNF clause such as ¬A∨¬B∨C can be written in definite clause form as A∧B=>C.

Propositional Logic: Proof by Resolution

- A procedure for converting to CNF. We illustrate the procedure by converting the sentence $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ into CNF. The steps are as follows:

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. CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following equivalences from Figure 7.11:

$$\neg(\neg\alpha) \equiv \alpha \quad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$

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

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

4. Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law from Figure 7.11, distributing $\vee$ over $\wedge$ wherever possible.

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

The original sentence is now in CNF, as a conjunction of three clauses.

Propositional Logic: Proof by Resolution

- A Resolution Algorithm:
- This uses the principle of 'proof by contradiction'. That is to show that KB $\models \alpha$, we show that (KB $\wedge \neg \alpha$) is unsatisfiable.
- We do this by proving a contradiction.
- First, (KB $\wedge \neg \alpha$) is converted into CNF. Then, the resolution rule is applied to the resulting clauses.
- Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
- The process continues until one of two things happens:
  - There are no new clauses that can be added, in which case KB does not entail ; or,
  - Two clauses resolve to yield the empty clause, in which case KB entails .

Propositional Logic: Proof by Resolution

- It means consider the sentence (query) to be proved, negate it and try to bring a contradiction.
- If a *contradiction exists*, we can say that *given sentence is proved* (KB entails α).
- *Otherwise*, the *inferred sentence will be added to the KB*.
- **Or**
- The process for resolution method contains the below steps:
  - Convert the given axiom into clause form, i.e., CNF.
  - Apply and proof the given goal using negation rule.
  - Use those literals which are needed to prove.
  - Solve the clauses together and achieve the goal.

Example of Propositional Resolution

- Consider the following Knowledge Base:
  1. The humidity is high or the sky is cloudy.
  2. If the sky is cloudy, then it will rain.
  3. If the humidity is high, then it is hot.
  4. It is not hot.
- **Goal:** It will rain.

Example of Propositional Resolution

- **Solution:** Let's construct propositions of the given sentences one by one:

1) Let,

   P: Humidity is high.

   Q: Sky is cloudy.

   R: It will rain.

   S: It is hot.

- It will be represented as **P V Q.**

2) Q: Sky is cloudy. …**from(1)**

- It will be represented as **Q → R.**

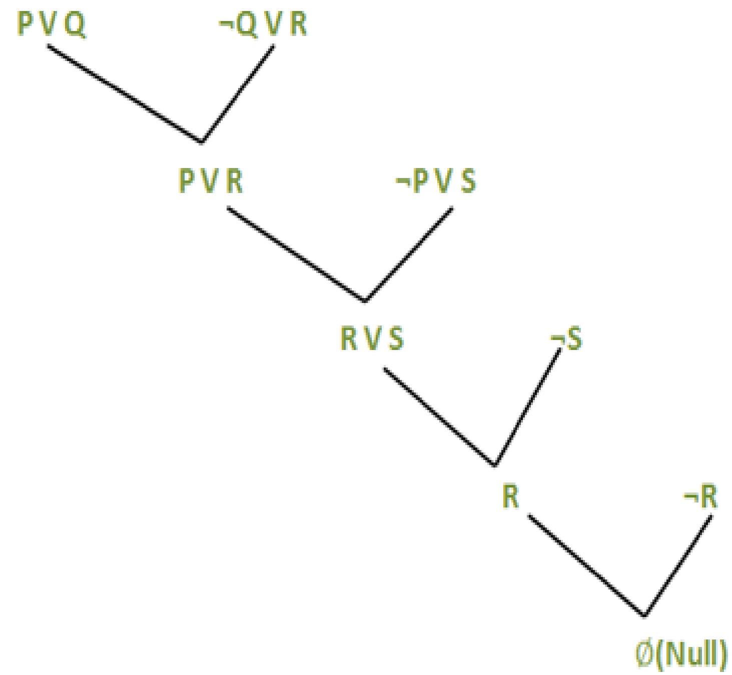
3) P: Humidity is high. …**from(1)**

- It will be represented as **P → S.**

4) ¬S: It is not hot.

Example of Propositional Resolution

- **Applying a resolution:**
- In (2), Q → R will be converted as (¬Q V R)
- In (3), P → S will be converted as (¬P V S)
- **Negation of Goal (¬R):** It will not rain.
- Finally, apply the rule as shown below:
- **After** applying **Proof by Refutation** (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause (Ø).**
- Hence, the goal is achieved.
- Thus, it is not raining.



Resolution Graph

Propositional Logic: Proof by Resolution

- <u>Algorithm:</u>
-

```
function PL-RESOLUTION(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   while true do
      for each pair of clauses Ci, Cj in clauses do
         resolvents ← PL-RESOLVE(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

Figure 3. A simple resolution algorithm for propositional logic. PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

Propositional Logic: Proof by Resolution

- Completeness of Resolution: PL-RESOLUTION is said to be complete. To do this, we introduce the **resolution closure RC(S)** of a set of clauses S, which is the set of all clauses derivable by repeated application of the resolution rule to clauses in S or their derivatives.
- *The resolution closure is what PL-RESOLUTION computes as the final value of the variable clauses*.
- It is easy to see that RC(S) must be finite, because there are only finitely many distinct clauses that can be constructed out of the symbols $P_1,\ldots,P_k$ that appear in S.
- Notice that this would not be true without the factoring step that removes multiple copies of literals.
- Hence, PL-RESOLUTION always terminates.

Propositional Logic: Proof by Resolution

- The completeness theorem for resolution in propositional logic is called the ground resolution theorem:
- If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Propositional Logic: Proof by Resolution

- Horn Clauses and Definite Clauses: Real-world KBs often contain only clauses of a restricted kind called '*Horn Clauses*'.
- Horn Clause -- a disjunction of literals of which <u>at most one is positive</u>
- <u>Ex:</u> $\neg L_{1,1} \vee \neg Breeze \vee B_{1,1}$ (Horn clause) ($L_{1,1}$-- agent's location is [1, 1])
- $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$ (Not a Horn clause) is not, because it has two positive clauses.
- Horn clauses are important because of three reasons:
- (i) Every horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a positive literal.
- <u>Ex:</u> The horn clause $\neg L_{1,1} \vee \neg Breeze \vee B_{1,1}$ can be written as $(L_{1,1} \wedge Breeze) \rightarrow B_{1,1}$ (easy to read)

Propositional Logic: Proof by Resolution

- *Definite clauses (or Goal clauses)* -- horn clauses with exactly one positive literal (the above)
- Head -- the positive literal
- Body -- the negative literals (A → B, A – body, B – head)
- *Fact* -- a definite clause with no negative literals
- A *horn clause with no positive literals* can be written as an implication whose conclusion is the literal false.
- Ex: $\neg W_{1,1} \vee \neg W_{1,2}$ is equivalent to $W_{1,1} \wedge W_{1,2} \rightarrow \text{False}$
- (ii) Inference with Horn clauses can be done through the *forward chaining* and *backward chaining* algorithms.
- (iii) Deciding entailment with Horn clauses can be done in time that is *linear* in the size of the KB.

PL: Forward and Backward Chaining

- Forward Chaining: The forward-chaining algorithm PL-FC-ENTAILS?(KB, q) 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.
- The FC is based on data driven approach.
- Procedure:
- Always the set of pre-conditions (LHS of an implication) is considered.
- If all the pre-conditions (premises) are satisfied, then the consequent (RHS) can be inferred.
- Repeat this process until the goal is satisfied (true).

PL: Forward and Backward Chaining

- The forward-chaining algorithm is shown in given Figure which runs in linear time.

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*
  inputs: KB, the knowledge base, a set of propositional definite clauses
    q, the query, a proposition symbol
  $count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
  $inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
  $queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

  while *queue* is not empty **do**
    $p \leftarrow$ POP($queue$)
    if $p = q$ **then return** *true*
    if $inferred[p] = false$ **then**
      $inferred[p] \leftarrow true$
      for each clause c in KB where p is in c.PREMISE **do**
        decrement $count[c]$
        if $count[c] = 0$ **then** add c.CONCLUSION to *queue*
  return *false*

PL: Forward and Backward Chaining

- In the above, the forward-chaining algorithm for propositional logic,
- The queue keeps track of symbols known to be true but not yet "processed."
- The count table keeps track of how many premises of each implication are not yet proven.
- Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda.
- Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again.
- This avoids redundant work and prevents loops caused by implications such as P=>Q and Q=>P.

PL: Forward and Backward Chaining

- For example 1, if $L_{1,\,1}$ and Breeze
- are known and
- $(L_{1,1} \wedge Breeze) \Rightarrow B_{1;1}$ is in the knowledge base, then $B_{1,\,1}$ can be added.
- This process continues until the query q is added or until no further inferences can be made.

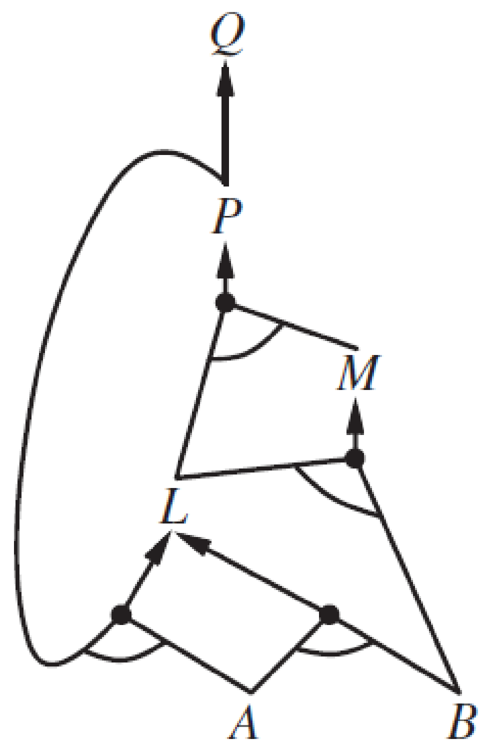
- Example 2: Consider the following KB:
  - 1. P → Q
  - 2. L Λ M → P
  - 3. B Λ L → M
  - 4. A Λ P → L
  - 5. A Λ B → L
  - 6. A
  - 7. B
- 1 to 5: are Rules, Query: Prove Q.
- 6, 7 : are facts
- One way (a sequence) to prove Q is:
- 6 → 7 → 5 → 3 → 2 → 1
- (*explain this process clearly while writing*)

PL: Forward and Backward Chaining…

Horn Clauses and the corresponding AND-OR graph:

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

(a)



(b)

PL: Forward and Backward Chaining

- Backward Chaining:
- Backward chaining is a form of goal-directed reasoning.
  - We start with the goal here.
  - If it is not directly given, we look for it in the RHS of an implication.
  - In the corresponding clause, the premise(s) (LHS) form(s) sub-goal(s).
  - We try to prove these sub-goals by finding them again in the RHS of any clause.
  - This process is repeated.
- At any stage, once the sub-goals are satisfied, the corresponding consequent can be considered as true and from there we go back to the clause from which we have come.

PL: Forward and Backward Chaining

- <u>Ex:</u> The KB: 1. $P \rightarrow Q$
- 2. $L \wedge M \rightarrow P$
- 3. $B \wedge L \rightarrow M$
- 4. $A \wedge P \rightarrow L$
- 5. $A \wedge B \rightarrow L$
- 6. A
- 7. B
- 1 to 5: are Rules, 6, 7 : are facts, <u>Query:</u> Prove Q.
- One way (a sequence) to prove Q is:
- 1 → 2 → 5 → 6 → 7 → 3 and then from 2, P is proved.
- From 1, we can infer Q. (*explain this process clearly*)
-