Unit-IV
Logical Agents

-Dr. Radhika V. Kulkarni

Associate Professor, Dept. of Computer Engineering,

Vishwakarma Institute of Technology, Pune.

Sources:

1. Stuart Russell & Peter Norvig, "Artificial Intelligence : A Modern Approach", Pearson Education, 2nd Edition.
2. Elaine Rich and Kevin Knight, "Artificial Intelligence" Tata McGraw Hill
3. Deepak Khemani, “A First Course in Artificial Intelligence”, McGraw Hill
4. Saroj Kaushik, “Artificial Intelligence”, Cengage Publication.
5. https://www.javatpoint.com/digital-image-processing-tutorial

DISCLAIMER

This presentation is created as a reference material for the students of TY-CS, VIT (AY 2023-24 Sem-1).

It is restricted only for the internal use and any circulation is strictly prohibited.

Syllabus

Unit-IV Logical Agents

Knowledge based agents, Wumpus world. Propositional Logic: Representation, Inference, Reasoning Patterns, Resolution, Forward and Backward Chaining. First order Logic: Representation, Inference, Reasoning Patterns, Resolution, Forward and Backward Chaining.

Knowledge based Agents

A story

- You roommate comes home; he/she is completely wet
- You know the following things:
  - Your roommate is wet
  - If your roommate is wet, it is because of rain, sprinklers, or both
  - If your roommate is wet because of sprinklers, the sprinklers must be on
  - If your roommate is wet because of rain, your roommate must not be carrying the umbrella
  - The umbrella is not in the umbrella holder
  - If the umbrella is not in the umbrella holder, either you must be carrying the umbrella, or your roommate must be carrying the umbrella
  - You are not carrying the umbrella
- Can you conclude that the sprinklers are on?
- Can AI conclude that the sprinklers are on?

Knowledge base for the story

- RoommateWet
- RoommateWet => (RoommateWetBecauseOfRain OR RoommateWetBecauseOfSprinklers)
- RoommateWetBecauseOfSprinklers => SprinklersOn
- RoommateWetBecauseOfRain => NOT(RoommateCarryingUmbrella)
- UmbrellaGone
- UmbrellaGone => (YouCarryingUmbrella OR RoommateCarryingUmbrella)
- NOT(YouCarryingUmbrella)

- **This is nothing but the LOGIC!**

Mathematical Logic

- In simple words, logic means to reason. This reasoning can be a legal opinion or even a Mathematical confirmation. The reasoning to solve mathematical problems is mathematical logic.
- In mathematical logic we formalize (formulate in a precise mathematical way) notions used informally by mathematicians such as:
  - Property
  - statement (in a given language)
  - Structure
  - truth (what it means for a given statement to be true in a given structure)
  - proof (from a given set of axioms)
  - algorithm
- Classification of mathematical logic:
  - Set Theory
  - Recursion Theory
  - Model Theory
  - Proof Theory
- Mathematical logical operators:
  - Conjunction or (AND : ^)
  - Disjunction or (OR: v)
  - Negation or (NOT: ~)

Knowledge based Agents

- The central component of a knowledge-based agent is its knowledge base, or KB. It is a set of sentences. (Here "sentence" is used as a technical term. It is related but not identical to the sentences of English and other natural languages.)
- Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.
- There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are TELL and ASK, respectively. Both operations may involve inference—that is, deriving new sentences from old.
- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told (or TELLed) to the knowledge base previously.
- The agent maintains a knowledge base, KB, which may initially contain some background knowledge. Each time the agent program is called, it does three things.
  - First, it TELLs the knowledge base what it perceives.
  - Second, it ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.
  - Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action.

Knowledge based Agents (2)

- Figure 7.1 shows the outline of a knowledge-based agent program. Like all our agents, it takes a percept as input and returns an action.

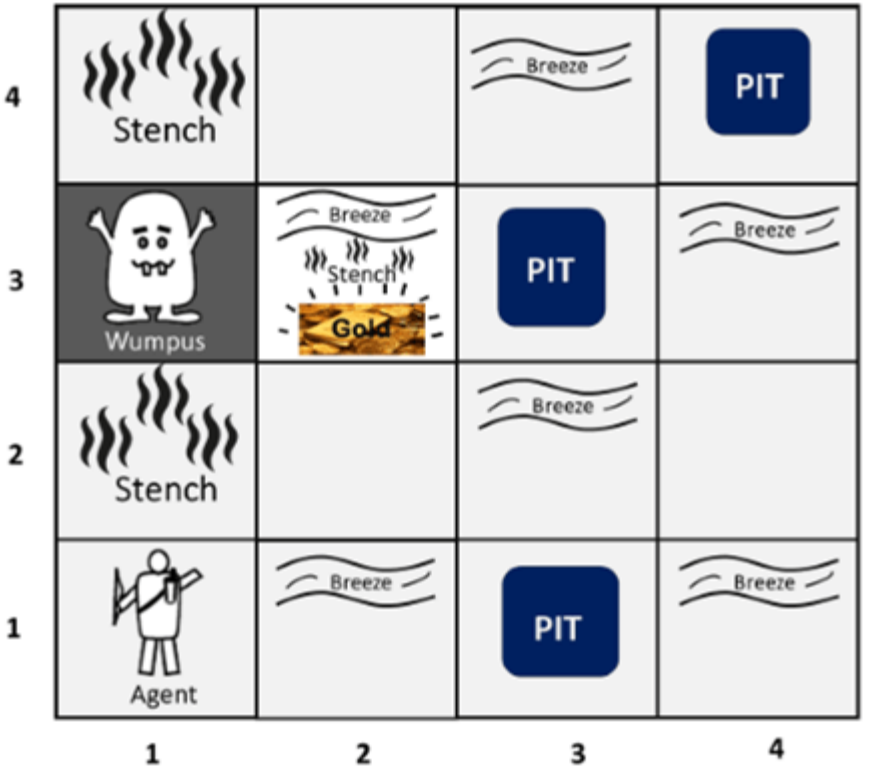
```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  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
```

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

The Wumpus World

The Wumpus World (1)

- The Wumpus World is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation.
- It was inspired by a video game Hunt the Wumpus by Gregory Yob in 1973.
- Problem Description:
- The Wumpus world is a cave which has 4/4 rooms connected with passageways. So, there are total 16 rooms which are connected with each other.
- We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room.
- The Wumpus can be shot by the agent, but the agent has a single arrow.
- In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever.
- The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So, the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus.
- The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.



The Wumpus World (2)

- There are also some components which can help the agent to navigate the cave. These components are given as follows:
  - The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
  - The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
  - There will be glitter in the room if and only if the room has gold.
  - The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

- PEAS description of Wumpus world: To explain the Wumpus world we have given PEAS description as below:
- Performance measure:
  - +1000 reward points if the agent comes out of the cave with the gold.
  - -1000 points penalty for being eaten by the Wumpus or falling into the pit.
  - -1 for each action, and -10 for using an arrow.
  - The game ends if either agent dies or came out of the cave.
- Environment:
  - A 4*4 grid of rooms.
  - The agent initially in room square [1, 1], facing toward the right.
  - Location of Wumpus and gold are chosen randomly except the first square [1,1].
  - Each square of the cave can be a pit with probability 0.2 except the first square.

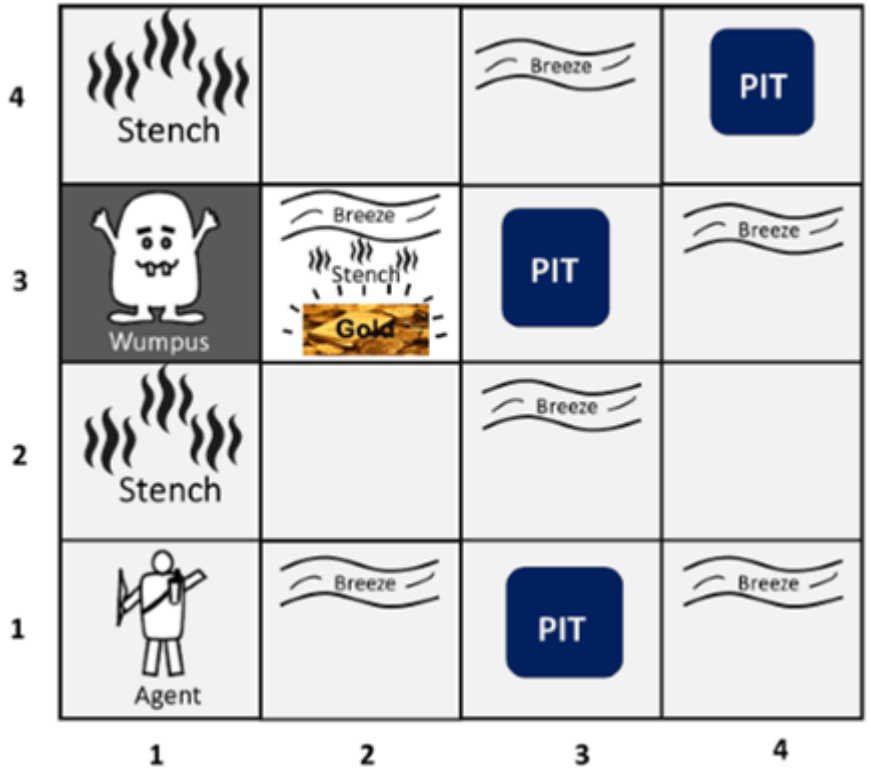
The Wumpus World (3)

- Actuators: Left turn, Right turn, Move forward, Grab, Release, Shoot.
- Sensors:
  - The agent will perceive the stench if he is in the room adjacent to the Wumpus. (Not diagonally).
  - The agent will perceive breeze if he is in the room directly adjacent to the Pit.
  - The agent will perceive the glitter in the room where the gold is present.
  - The agent will perceive the bump if he walks into a wall.
  - When the Wumpus is shot, it emits a horrible scream which can be perceived anywhere in the cave.
  - These percepts can be represented as five element list, in which we will have different indicators for each sensor.
  - **Example:** if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as: [Stench, Breeze, None, None, None].

- The Wumpus world Properties:
  - Partially observable: The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
  - Deterministic: It is deterministic, as the result and outcome of the world are already known.
  - Sequential: The order is important, so it is sequential.
  - Static: It is static as Wumpus and Pits are not moving.
  - Discrete: The environment is discrete.
  - One agent: The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

The Wumpus World (4)

- Exploring the Wumpus world:
- Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.



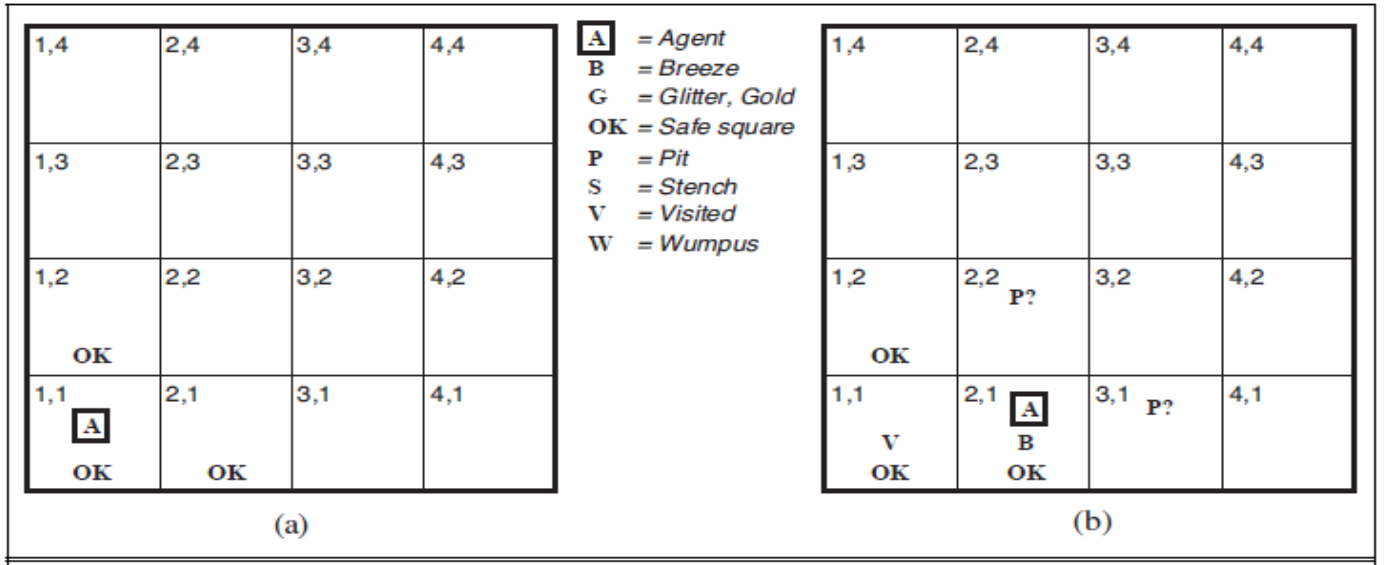


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After one move, with percept [*None*, *Breeze*, *None*, *None*, *None*].

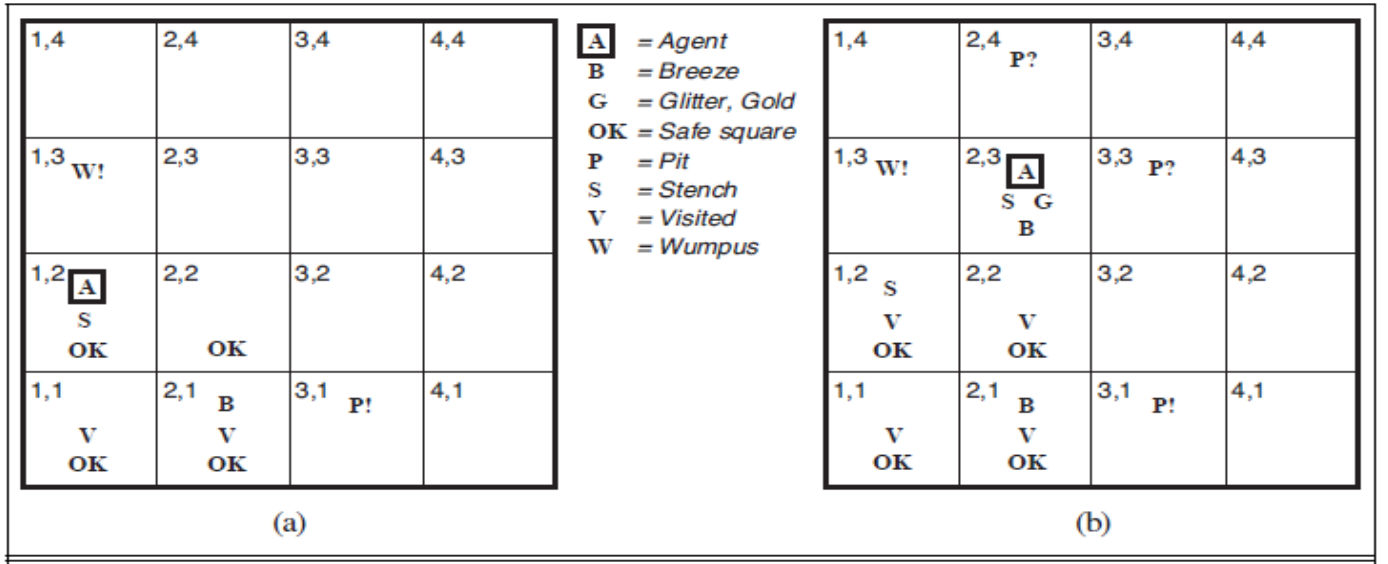


Figure 7.4 Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench*, *None*, *None*, *None*, *None*]. (b) After the fifth move, with percept [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

Propositional Logic

Propositional Logic (1)

- **Proposition (Statement)** is a declarative statement that is either True or False. It is one of the building blocks of Logic. E.g., Delhi is capital of India. Pune is a city in India. VIT is an autonomous institute.
- Exclamatory sentences, questions, commands, opinions are not statements. E.g., What a beautiful morning!! What is your name ? Mute yourselves. China is responsible for Corona Pandemic.

- **Propositional logic** deals with truth values and the logical connectives ¬ (not), ∧ (and), ∨ (or), etc. Most of the concepts in propositional logic have counterparts in first-order logic.

- Propositional Logic is a formal language. Here are the most fundamental concepts.
  - **Syntax** refers to the formal notation for writing assertions. It also refers to the data structures that represent assertions in a computer. It uses a set of propositional symbols P, Q, R, …and logical connectives, → (implies), ↔ (if and only if) to form atomic formulae. E.g. At the level of syntax, 1 + 2 is a string of three symbols, or a tree with a node labelled + and having two children labelled 1 and 2.

  - **Semantics** expresses the meaning of a formula in terms of mathematical or real-world entities. The semantics of a logical statement will typically be true or false. E.g. While 1 + 2 and 2 + 1 are syntactically distinct, they have the same semantics, namely 3.

  - **Proof theory** concerns ways of proving statements, at least the true ones. Typically, we begin with axioms and arrive at other true statements using inference rules. Formal proofs are typically finite and mechanical: their correctness can be checked without understanding anything about the subject matter.

Propositional Logic (2)

Connective	Symbol	Example (p and q are simple statements)
p and q	$\wedge$	$p \wedge q$
p or q	$\vee$	$p \vee q$
not p	$\neg$ or ~	$\neg p$ or ~p
Implies (if then)	$\rightarrow$	$p \rightarrow q$
if and only if (iff)	$\leftrightarrow$	$p \leftrightarrow q$

- The truth value of a proposition is true, denoted by T, if it is a true proposition, and the truth value of a proposition is false, denoted by F, if it is a false proposition.
- It uses truth table to get semantics of the logical formulae.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

  - A → B is known as *modus ponens.*
  - By inspecting the table, we can see that A → B is equivalent to ¬A ∨ B and that A ↔ B is equivalent to (A → B) ∧ (B → A). (The latter is also equivalent to ¬(A ⊕ B), where ⊕ is exclusive-or.)

Propositional Logic (3)

- Syntax of Propositional Logic:
- Let *S* be a set whose elements will be called propositional letters. We'll denote the propositional letters by *p*, *q*, and so on. A propositional logic language *L* with propositional letters *S* is the collection of formulas determined by the following four conditions:
  a) All propositional letters are formulas.
  b) If α is a formula, so is (¬ α).
  c) If α and β are formulas, so are (α ∨ β), (α ∧ β), (α → β), and (α ≡ β).
  d) All formulas are obtained in this manner.
- Some people call formulas well formed formulas, or simply *WFFs*.

- An interpretation, or truth assignment, for a set of formulas is a function from its set of propositional symbols to {1, 0}. An interpretation satisfies a formula if the formula evaluates to 1 under the interpretation.
- In propositional logic, a valid formula is also called a tautology.

Propositional Logic (4)

- Some of the logically equivalent propositions are listed here. They are also called **identities**.
- The symbol ⇔ shows the logical equivalence.

Fundamental Logical Equivalences

Idempotence[a]
$(P \vee P) \Leftrightarrow P$
$(P \wedge P) \Leftrightarrow P$

Domination
$(P \vee \mathbf{T}) \Leftrightarrow \mathbf{T}$
$(P \wedge \mathbf{F}) \Leftrightarrow \mathbf{F}$

Associativity
$[(P \vee Q) \vee R] \Leftrightarrow [P \vee (Q \vee R)]$
$[(P \wedge Q) \wedge R] \Leftrightarrow [P \wedge (Q \wedge R)]$

Identity
$(P \vee \mathbf{F}) \Leftrightarrow P$
$(P \wedge \mathbf{T}) \Leftrightarrow P$

Commutativity
$(P \vee Q) \Leftrightarrow (Q \vee P)$
$(P \wedge Q) \Leftrightarrow (Q \wedge P)$

De Morgan's Laws
$[\neg(P \vee Q)] \Leftrightarrow [(\neg P) \wedge (\neg Q)]$
$[\neg(P \wedge Q)] \Leftrightarrow [(\neg P) \vee (\neg Q)]$

Distributivity ($\wedge$ over $\vee$)
$[P \wedge (Q \vee R)] \Leftrightarrow [(P \wedge Q) \vee (P \wedge R)]$
$[(P \vee Q) \wedge R] \Leftrightarrow [(P \wedge R) \vee (Q \wedge R)]$

Distributivity ($\vee$ over $\wedge$)
$[P \vee (Q \wedge R)] \Leftrightarrow [(P \vee Q) \wedge (P \vee R)]$
$[(P \wedge Q) \vee R] \Leftrightarrow [(P \vee R) \wedge (Q \vee R)]$

Law of the Excluded Middle
$[P \vee (\neg P)] \Leftrightarrow \mathbf{T}$

Law of Contradiction
$[P \wedge (\neg P)] \Leftrightarrow \mathbf{F}$

Law of Double Negation (Involution)
$\neg(\neg P) \Leftrightarrow P$

Law of Addition
$[P \rightarrow (P \vee Q)] \Leftrightarrow \mathbf{T}$

Law of Simplification
$[(P \wedge Q) \rightarrow P] \Leftrightarrow \mathbf{T}$
$[(P \wedge Q) \rightarrow Q] \Leftrightarrow \mathbf{T}$

[a] An idempotent is an algebraic element for which $x^2 = x$.

Propositional Logic (5)

Q. 1 Prove that $(p \vee p) \wedge (p \rightarrow (q \vee q))$ is equivalent to **$p \wedge q$**

Answer: $(p \vee p) \wedge (p \rightarrow (q \vee q)) \Leftrightarrow p \wedge (p \rightarrow q)$

$\Leftrightarrow p \wedge (\neg p \vee q)$

$\Leftrightarrow (p \wedge \neg p) \vee (p \wedge q)$

$\Leftrightarrow F \vee (p \wedge q)$

$\Leftrightarrow (p \wedge q)$

Q.2 $p \rightarrow (q \rightarrow r) \Leftrightarrow$ **$(p \wedge q) \rightarrow r$**

Answer: Let us explore the L.H.S first

$p \rightarrow (q \rightarrow r) \Leftrightarrow \neg p \vee (\neg q \vee r)$

$\Leftrightarrow (\neg p \vee \neg q) \vee r$

$\Leftrightarrow \neg(p \wedge q) \vee r$

$\Leftrightarrow (p \wedge q) \rightarrow r$

Propositional Logic (6)

Q. 3 Show that p⇒(pVq) is a tautology.

Answer:

p	q	pVq	p⇒(pVq)
T	T	T	**T**
T	F	T	**T**
F	T	T	**T**
F	F	F	**T**

Q. 4 Show that the statement p Λ~p is a contradiction.

Answer:

p	~p	p Λ~p
T	F	**F**
F	T	**F**

Propositional Logic (7)

Q. 5 Which of the following is true about the proposition
$p \wedge (\neg p \vee q)$?
a)Tautology
b)Contradiction
c)Logically equivalent to $p \wedge q$
d)None of these

Answer: The proposition can be written as
$(p \wedge \neg p) \vee (p \wedge q) \Leftrightarrow F \vee (p \wedge q)$
$\Leftrightarrow (p \wedge q)$. So the **answer is (c)**

Q. 6 What is the dual value of $(p \wedge q) \vee T$?

Answer: The dual value of any expression is created by replacing $\vee$ with $\wedge$, $\wedge$ with $\vee$, T with F and F with T.
Thus, the dual value of the given expression is **$(p \vee q) \wedge F$**.

Propositional Logic (8)

Q.7 Which of the following proposition is a tautology?

a)$(p \vee q) \rightarrow p$
b) $p \vee (q \rightarrow p)$
c) $p \vee (p \rightarrow q)$
d) $p \rightarrow (p \rightarrow q)$

Answer: There are two ways to solve the problem. The **first** method is by drawing the **Truth-table** for each option given and thereby finding the tautology. This approach sometimes becomes a time and space consuming approach.

The **second** approach is by making the given expression of the form **T $\vee$ (an expression)**. Where T is a known tautology (like $p \vee \neg p$, $q \vee \neg q$ etc..)

Option (a) can be written as $\neg(p \vee q) \vee p$.(There is no tautology).

Option (b) can be written as: $p \vee (\neg q \vee p) \Leftrightarrow (p \vee \neg q) \vee (p \vee p)$

$\Leftrightarrow (p \vee \neg q) \vee p$ (there is no tautology present in this also)

Option (c) can be written as: $p \vee (p \rightarrow q) \Leftrightarrow p \vee (\neg p \vee q)$

$\Leftrightarrow (p \vee \neg p) \vee (p \vee q)$

$\Leftrightarrow T \vee (\text{expression})$

So, the truth value of the statement is always true. So it is a **tautology**. **Ans. is (c)**

Propositional Logic (9)

Q. 8 Let X denotes $(p \vee q) \rightarrow r$ and Y denotes $(p \rightarrow r) \vee (q \rightarrow r)$. Which of the following is a tautology?

a) $X \leftrightarrow Y$ b) $X \rightarrow Y$ c) $Y \rightarrow X$ d) $\neg Y \rightarrow X$

Answer: We need to draw truth tables for all the options given.

p	Q	r	$p \vee q$	$p \rightarrow r$	$q \rightarrow r$	X	Y	$\neg$ Y	$X \rightarrow Y$	$Y \rightarrow X$	$\neg Y \rightarrow X$
F	F	F	F	T	T	T	T	F	T	T	T
F	F	T	F	T	T	T	T	F	T	T	T
F	T	T	T	T	T	T	T	F	T	T	T
T	T	T	T	T	T	T	T	F	T	T	T
T	F	F	T	F	F	F	T	F	T	F	T
T	T	F	T	F	F	F	F	T	T	T	F
F	T	F	T	T	T	F	T	F	T	F	T
T	F	T	T	T	T	T	T	F	T	T	T

A Notion of Truth

Truth (1)

- Semantics:
- A logic must also define the semantics or meaning of sentences. The semantics defines the truth of each sentence with respect to each possible world.
- E.g., the semantics for arithmetic specifies that the sentence “x + y =4” is true in a world where x is 2 and yis 2, but false in a world where x is 1 and y is 1.
- In standard logics, every sentence must be either true or false in each possible world—there is no “in between.” In mathematical abstractions of real environment, the term model in place of “Possible world”.

- Truth Vs Proof:
- Concepts of proof and truth play an important role in metamathematics, especially in the methodology and the foundations of mathematics.
- Truth is the state or quality of being true to someone or something while proof is an effort, process, or operation designed to establish or discover a fact or truth; an act of testing; a test; a trial.
- We have completely separate definitions of "truth" ($\models$) and "provability" ($\vdash$). We would like them to be the same; that is, we should only be able to prove things that are true, and if they are true, we should be able to prove them.

Truth (2)

- Entailment:
- If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α. We use the notation M(α) to mean the set of all models of α.
- Truth involves the relation of logical entailment between sentences—the idea that a sentence follows logically from another sentence.
- It gives semantic consequence and is denoted by the double turnstile (⊨). In mathematical notation, we write α ⊨ β to mean that the sentence α entails the sentence β.
- **The formal definition of entailment**: α ⊨ β if and only if, in every model in which α is true, β is also true.
- One says that *"α logically implies β"*, or *"β is a semantic/logical consequence of α"*, or α ⊨ β, when all possible valuations in which α is true, β is also true.
- Using the notation just introduced, we can write α ⊨ β if and only if M(α) ⊆ M(β). (Note the direction of the ⊆ here: if α ⊨ β, then α is a stronger assertion than β: it rules out more possible worlds.)
- The relation of entailment is familiar from arithmetic; we are happy with the idea that the sentence x = 0 entails the sentence xy = 0. Obviously, in any model where x is zero, it is the case that xy is zero (regardless of the value of y).

Truth (3)

- Entailment: (cont..)
- We can apply the same kind of analysis to the Wumpus-world reasoning example. Consider the situation when the agent has detected nothing in [1,1] and a breeze in [2,1]. These percepts, combined with the agent's knowledge of the rules of the Wumpus world, constitute the KB.
- The agent is interested (among other things) in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits. Each of the three squares might or might not contain a pit, so (for the purposes of this example) there are 2^3 =8 possible models. These eight models are shown in Figure 7.5.

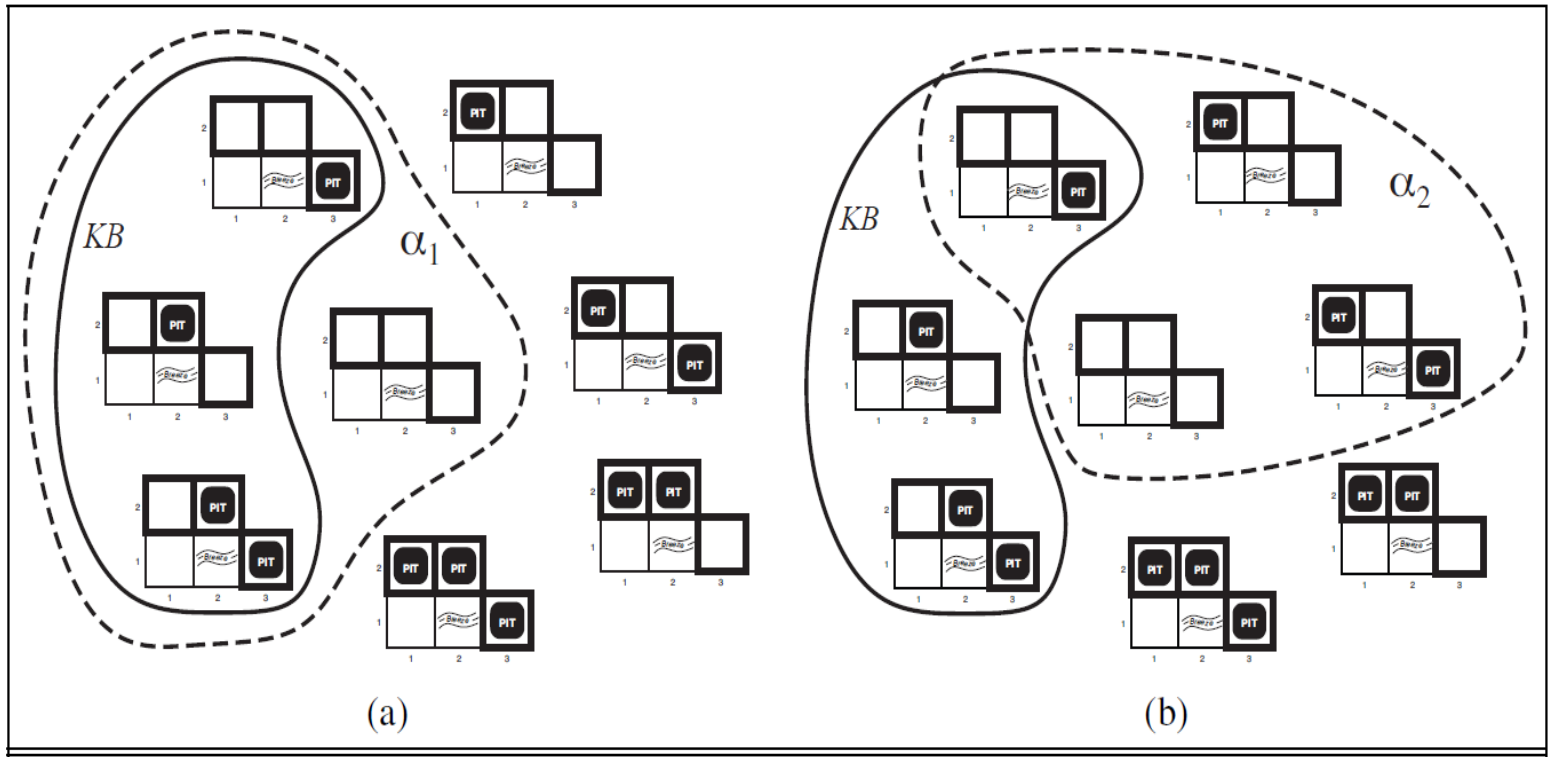


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

Truth (4)

- Entailment: (cont..)
- The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences. The KB is false in models that contradict what the agent knows— for example, the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1].
- There are in fact just three models in which the KB is true, and these are shown surrounded by a solid line in Fig. 7.5.
- Now let us consider two possible conclusions:

  α1 = "There is no pit in [1,2]."

  α2 = "There is no pit in [2,2]."
- We have surrounded the models of α1 and α2 with dotted lines in Figures 7.5(a) and 7.5(b), respectively.
- By inspection, we see the following:

  in every model in which KB is true, α1 is also true.
- Hence, KB $\models$ α1: there is no pit in [1,2]. We can also see that

  in some models in which KB is true, α2 is false.
- Hence, KB $\not\models$ α2: the agent cannot conclude that there is no pit in [2,2]. (Nor can it conclude that there is a pit in [2,2].)

Truth (5)

- Logical Inference:
- The preceding example not only illustrates entailment but also shows how the definition of entailment can be applied to derive conclusions—that is, to carry out logical inference.
- The inference algorithm illustrated in Figure 7.5 is called model checking, because it enumerates all possible models to check that α is true in all models in which KB is true, that is, that M(KB) ⊆ M(α).
- In understanding entailment and inference, it might help to think of the set of all consequences of KB as a haystack and of α as a needle. Entailment is like the needle being in the haystack; inference is like finding it.
- This distinction is embodied in some formal notation: if an inference algorithm i can derive α from KB, we write $KB \vdash_i \alpha$, which is pronounced “α is derived from KB by i” or “i derives α from KB.”
- In proof theory, the turnstile (⊢) is used to denote "provability" or "derivability".
  - It is also referred to as **tee** and is often read as "yields", "proves", "satisfies" or "entails".
  - E.g., if α is a formal theory and β is a particular sentence in the language of the theory then $\alpha \vdash \beta$ *means that* β *is provable from* α .
  - It gives a syntactic consequence.
  - A *modus ponens rule* can be written as a sequent notation as $\alpha \rightarrow \beta, \alpha \vdash \beta$.

Truth (6)

- Soundness & Completeness:
- These two properties are called soundness and completeness.
  - A proof system is sound if everything that is provable is in fact true. In other words, if $\phi_1, \ldots, \phi_n \vdash \psi$ then $\phi_1, \ldots, \phi_n \models \psi$.
  - A proof system is complete if everything that is true has a proof. In other words, if $\phi_1, \ldots, \phi_n \models \psi$ then $\phi_1, \ldots, \phi_n \vdash \psi$.
- If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world.

- The notion of truth for propositional logic and predicate logic are based on truth assignments to the propositional letters and predicates, respectively.
- There are three interrelated truth notions for a formula:
  - Valid: All possible interpretations make the formula true.
  - Satisfiable: Some possible interpretation makes the formula true.
  - Unsatisfiable: No possible interpretations make the formula true.
  - In particular, it follows that $\boldsymbol{\alpha}$ is valid if and only if $(\neg \boldsymbol{\alpha})$ is unsatisfiable.

- The last two notions are extended to a set $\boldsymbol{S}$ of formulas, but the terminology differs:
  - Consistent: Some interpretation makes all the formulas in $\boldsymbol{S}$ simultaneously true.
  - Inconsistent: No interpretation makes all the formulas in $\boldsymbol{S}$ simultaneously true.

Propositional Theorem Proving

Theorem Proving (1)

- We have shown how to determine entailment by model checking: enumerating models and showing that the sentence must hold in all models.
- Entailment can also be done by theorem proving—applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.
- If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

- Logical Equivalence:
- Two sentences α and β are logically equivalent if they are true in the same set of models. We write this as $\alpha \equiv \beta$. For example, we can easily show (using truth tables) that $P \wedge Q$ and $Q \wedge P$ are logically equivalent.
- An alternative definition of equivalence is as follows: any two sentences α and β are equivalent only if each of them entails the other: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$.

- Validity:
- A sentence is valid if it is true in all models. For example, the sentence $P \vee \neg P$ is valid. Valid sentences are also known as tautologies—they are necessarily true.
- From our definition of entailment, we can derive the deduction theorem: For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.

Theorem Proving (2)

- Satisfiability:
- A sentence is satisfiable if it is true in, or satisfied by, some model.
- Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.

- Validity and satisfiability are of course connected: α is valid iff $\neg\alpha$ is unsatisfiable; contrapositively, α is satisfiable iff $\neg\alpha$ is not valid.
- We also have the following useful result:

  $\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable.
- Proving β from α by checking the unsatisfiability of $(\alpha \wedge \neg\beta)$ corresponds exactly to the standard mathematical proof technique of reductio ad absurdum (literally, “reduction to an absurd thing”).
- It is also called proof by refutation or proof by contradiction.
- One assumes a sentence β to be false and shows that this leads to a contradiction with known axioms α.
- This contradiction is exactly what is meant by saying that the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable.

Theorem Proving (3)

- Inference and Proofs:
- The best-known inference rule is called Modus Ponens (Latin for mode that affirms) and is written as

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

- The notation means that, whenever any sentences of the form α ⇒ β and α are given, then the sentence β can be inferred.
- For example, if (WumpusAhead ∧ WumpusAlive) ⇒ Shoot and (WumpusAhead ∧ WumpusAlive) are given, then Shoot can be inferred.

- Another useful inference rule is And-Elimination, which says that, from a conjunction, any of the conjuncts can be inferred:

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

- For example, from (WumpusAhead ∧ WumpusAlive), WumpusAlive can be inferred.
- By considering the possible truth values of α and β, one can show easily that Modus Ponens and And-Elimination are sound once and for all.
- These rules can then be used in any particular instances where they apply, generating sound inferences without the need for enumerating models.

Theorem Proving (4)

- All of the logical equivalences can be used as inference rules.
- For example, the equivalence for biconditional elimination yields the two inference rules:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}.$$

- Monotonicity:
- It says that the set of entailed sentences can only increase as information is added to the knowledge base.
- For any sentences α and β, if KB ⊨ α then KB ∧ β ⊨ α .
- For example, suppose the knowledge base contains the additional assertion β stating that there are exactly eight pits in the world. This knowledge might help the agent draw additional conclusions, but it cannot invalidate any conclusion α already inferred—such as the conclusion that there is no pit in [1,2].
- 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.
- Monotonic reasoning is not useful for the real-time systems, as in real time, facts get changed, so we cannot use monotonic reasoning.
- It is used in conventional reasoning systems, and a logic-based system is monotonic.
- Any theorem proving is an example of monotonic reasoning.

Proof by Resolution

Resolution Rule (1)

- The resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals.
- A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether $\alpha \vDash \beta$.
- A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other (in the following, ¬c is taken to be the complement to c). The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{a_1 \lor a_2 \lor \cdots \lor c, \quad b_1 \lor b_2 \lor \cdots \lor \neg c}{a_1 \lor a_2 \lor \cdots \lor b_1 \lor b_2 \lor \cdots}$$

where

all a_i, b_i, and c are literals,

the dividing line stands for "entails".

The above may also be written as:

$$\frac{(\neg a_1 \land \neg a_2 \land \cdots) \to c, \quad c \to (b_1 \lor b_2 \lor \cdots)}{(\neg a_1 \land \neg a_2 \land \cdots) \to (b_1 \lor b_2 \lor \cdots)}$$

Resolution Rule (2)

- A resolution rule can be schematically written as:

$$\frac{\Gamma_1 \cup \{\ell\} \quad \Gamma_2 \cup \left\{\bar{\ell}\right\}}{\Gamma_1 \cup \Gamma_2} |\ell|$$

We have the following terminology:

- The clauses $\Gamma_1 \cup \{\ell\}$ and $\Gamma_2 \cup \left\{\bar{\ell}\right\}$ are the inference's premises
- $\Gamma_1 \cup \Gamma_2$ (the resolvent of the premises) is its conclusion.
- The literal ℓ is the left resolved literal,
- The literal $\bar{\ell}$ is the right resolved literal,
- $|\ell|$ is the resolved atom or pivot.

- The clause produced by the resolution rule is called the resolvent of the two input clauses. It is the principle of consensus applied to clauses rather than terms.
- When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.
- Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause).

$$\frac{p \rightarrow q, \quad p}{q}$$

is equivalent to

$$\frac{\neg p \vee q, \quad p}{q}$$

Conjunctive Normal Form (1)

- The resolution rule applies only to clauses (that is, disjunctions of literals), so it would seem to be relevant only to knowledge bases and queries consisting of clauses.
- 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.
- E.g. $\neg((\neg A \rightarrow \neg B) \wedge \neg C) \equiv (\neg A \vee C) \wedge (B \vee C)$,
- Given a CNF formula, we can toss each of its clauses into the knowledge base (KB) using following general rules.
  - First, we try to reduce everything to negation, conjunction, and disjunction.
  - Next, we try to push negation inwards so that they sit on the propositional symbols (forming literals). Note that when negation gets pushed inside, it flips conjunction to disjunction, and vice-versa.
  - Finally, we distribute so that the conjunctions are on the outside, and the disjunctions are on the inside.

- Note that each of these operations preserves the semantics of the logical form.

- Also, when we apply a CNF rewrite rule, we replace the old formula with the new one, so there is no blow-up in the number of formulas.

Conjunctive Normal Form (2)

- Conversion to CNF:

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg(f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

Initial formula:

$$(\text{Summer} \rightarrow \text{Snow}) \rightarrow \text{Bizzare}$$

Remove implication ($\rightarrow$):

$$\neg(\neg\text{Summer} \vee \text{Snow}) \vee \text{Bizzare}$$

Push negation ($\neg$) inwards (de Morgan):

$$(\neg\neg\text{Summer} \wedge \neg\text{Snow}) \vee \text{Bizzare}$$

Remove double negation:

$$(\text{Summer} \wedge \neg\text{Snow}) \vee \text{Bizzare}$$

Distribute $\vee$ over $\wedge$:

$$(\text{Summer} \vee \text{Bizzare}) \wedge (\neg\text{Snow} \vee \text{Bizzare})$$

Resolution Algorithm

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\,\}$
  loop do
    for each pair of clauses C_i, C_j **in** $clauses$ **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if $resolvents$ contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

Figure 7.12 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

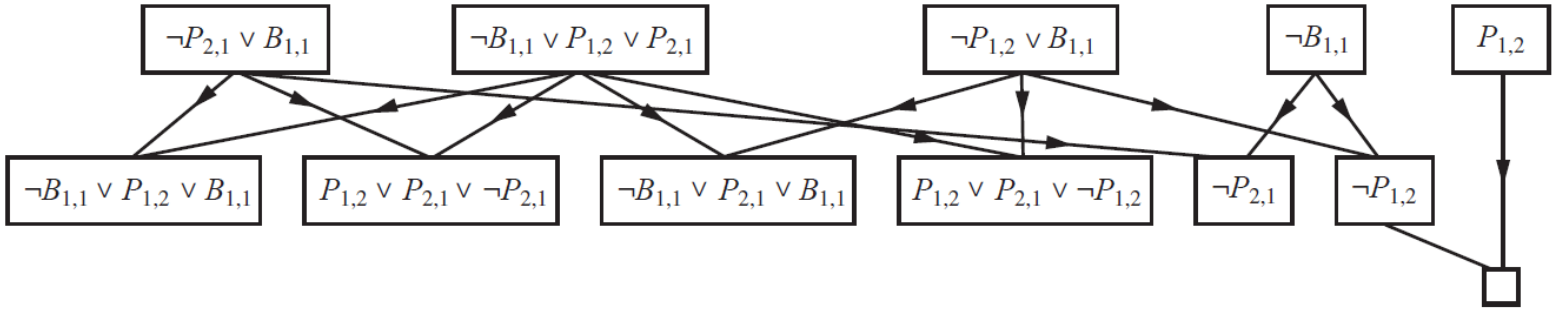


Figure 7.13 Partial application of PL-RESOLUTION to a simple inference in the wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row.

Resolution of Horn Clauses (1)

- Horn clause is clause (a disjunction of literals) with at most one positive (unnegated) literal and zero or more negative literals.
- E.g., $[\neg p, \neg q, \ldots, \neg t, u]$. It is representation for $(\neg p \vee \neg q \vee \ldots \vee \neg t \vee u)$ or $(p \wedge q \wedge \ldots \wedge t) \supset u$. Its implication is $(p \wedge q \wedge \ldots \wedge t) \rightarrow u$.

- The negative literals correspond to the propositional symbols on the left side of the implication, and the positive literal corresponds to the propositional symbol on the right side of the implication.

Written with implication	Written with disjunction (Clausal form)
$A \rightarrow C$	$\neg A \vee C$
$A \wedge B \rightarrow C$	$\neg A \vee \neg B \vee C$

- Conversely, a disjunction of literals with at most one negated literal is called a dual-Horn clause.

Resolution of Horn Clauses (2)

- Types of Horn Clauses :
  - Definite clause / Strict Horn clause – It has exactly one positive literal.
  - Unit clause – Definite clause with no negative literals. A unit clause without variables is a fact.
  - Goal clause – Horn clause without a positive literal. Also referred to as negative clause or integrity constraint.

Type of Horn clause	Disjunction form	Implication form	Read intuitively as
Definite clause or strict Horn clause	$\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$	$u \leftarrow p \wedge q \wedge \ldots \wedge t$	assume that, if p and q and ... and t all hold, then also u holds
Fact	u	$u \leftarrow true$	assume that u holds
Goal clause	$\neg p \vee \neg q \vee \ldots \vee \neg t$	$false \leftarrow p \wedge q \wedge \ldots \wedge t$	show that p and q and ... and t all hold

- All variables in a clause are implicitly universally quantified with the scope being the entire clause.
  - E.g., $\neg$ human(X) $\vee$ mortal(X)

    stands for: $\forall X(\neg$ human(X) $\vee$ mortal(X))

    which is logically equivalent to: $\forall X$ (human(X) $\rightarrow$ mortal(X))

Resolution of Horn Clauses (3)

- Resolution or Generalized inference rule for clauses have any number of literals:
  - Let's try to generalize modus ponens by allowing it to work on general clauses. This generalized inference rule is called resolution, which was invented in 1965 by John Alan Robinson.
  - The idea behind resolution is that it takes two general clauses, where one of them has some propositional symbol ***p*** and the other clause has its negation **¬p**, and simply takes the disjunction of the two clauses with ***p*** and ***¬p*** removed. Here, $f_1, \ldots, f_n, g_1, \ldots, g_m$ are arbitrary literals.
  - We can verify the soundness of resolution by checking its semantic interpretation. Indeed, the intersection of the models of f and g is a subset of models of ***f ∨ g***.

$$\neg A \vee B \vee \neg C \vee D \vee \neg E \vee F$$

Example: resolution inference rule

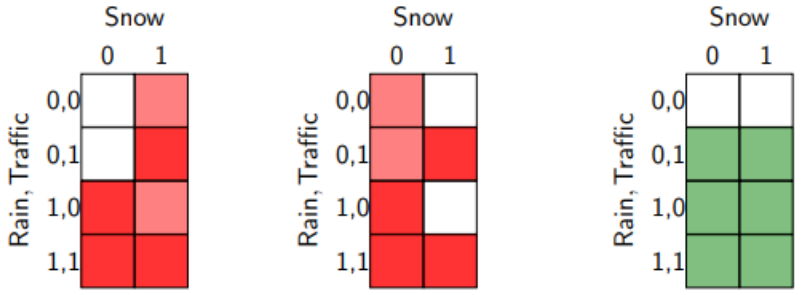
$$\frac{\text{Rain} \vee \text{Snow}, \quad \neg\text{Snow} \vee \text{Traffic}}{\text{Rain} \vee \text{Traffic}}$$

Definition: resolution inference rule

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \quad \neg p \vee g_1 \vee \cdots \vee g_m}{f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m}$$

$$\frac{\text{Rain} \vee \text{Snow}, \quad \neg\text{Snow} \vee \text{Traffic}}{\text{Rain} \vee \text{Traffic}} \text{ (resolution rule)}$$

$\mathcal{M}(\text{Rain} \vee \text{Snow}) \cap \mathcal{M}(\neg\text{Snow} \vee \text{Traffic}) \subseteq ? \mathcal{M}(\text{Rain} \vee \text{Traffic})$



Sound!

Resolution Algorithm (1)

- After we have converted all the formulas to CNF, we can repeatedly apply the resolution rule,

  to check $KB \models f \leftrightarrow KB \cup \{\neg f\}$ is unsatisfiable.

  - Both testing for entailment and contradiction boil down to checking satisfiability. Resolution can be used to do this very thing. If we ever apply a resolution rule (e.g., to premises A and ¬A) and we derive false (which represents a contradiction), then the set of formulas in the knowledge base is unsatisfiable.
  - If we are unable to derive false, that means the knowledge base is satisfiable because resolution is complete. However, unlike in model checking, we don't actually produce a concrete model that satisfies the KB.

- Resolution-based Inference Algorithm:
  - Add $\neg f$ into KB.
  - Convert all formulas into CNF.
  - Repeatedly apply resolution rule.
  - Return entailment iff derive false.

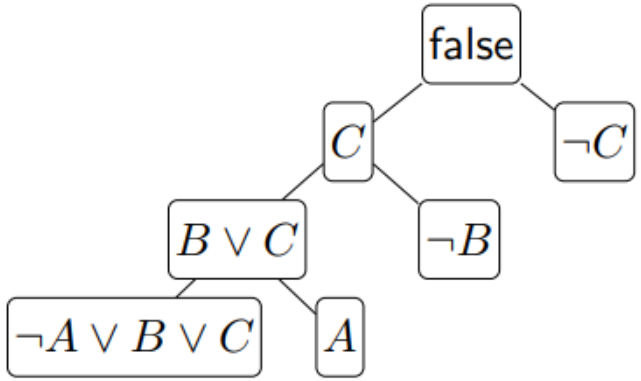
Resolution Algorithm (2)

- Resolution Algorithm Example:
- Here's an example of taking a negation of a formula to a knowledge base, converting it into CNF, and applying resolution. In this case, we derive false, which means that the original knowledge base was unsatisfiable.

  Given: After adding ¬*f* into *KB the resulting* KB' = {A → (B ∨ C), A, ¬B, ¬C}

  Convert to CNF: KB' = {¬A ∨ B ∨ C, A, ¬B, ¬C}

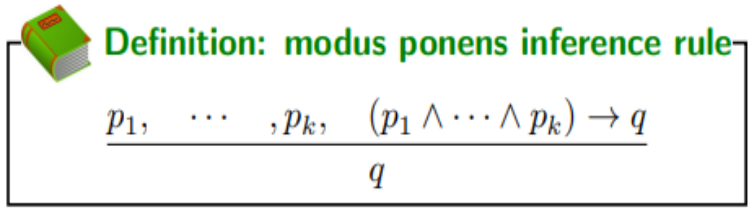
  Repeatedly apply resolution rule:



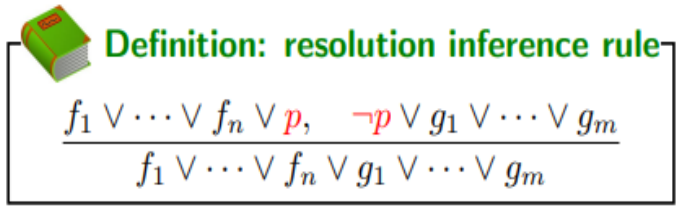
  Conclusion: ***KB*** ⊨ ***f*** (*KB* entails *f*) .

Resolution Algorithm (3)

- Time Complexity:
- We have a sound and complete inference procedure for all of propositional logic (although we didn't prove completeness). But what do we have to pay computationally for this increase?
- If we only have to apply modus ponens, each propositional symbol can only get added once, so with the appropriate algorithm (forward chaining), we can apply all necessary modus ponens rules in linear time.
- But with resolution, we can end up adding clauses with many propositional symbols, and possibly any subset of them! Therefore, this can take exponential time.



- Each rule application adds clause with **one** propositional symbol ⇒ linear time



- Each rule application adds clause with **many** propositional symbols ⇒ exponential time

Resolution Algorithm (4)

- To summarize, we can either content ourselves with the limited expressivity of Horn clauses and obtain an efficient inference procedure (via modus ponens).
- If we wanted the expressivity of full propositional logic, then we need to use resolution and thus pay more.

Horn clauses	Any clauses
modus ponens	resolution
linear time	exponential time
less expressive	more expressive

- Two inference procedures based on modus ponens for Horn KBs:
- **Forward chaining:**
  - Idea: Whenever the premises of a rule are satisfied, infer the conclusion. Continue with rules that became satisfied.
- **Backward chaining (goal reduction):**
  - Idea: To prove the fact that appears in the conclusion of a rule prove the premises of the rule. Continue recursively.
- Both procedures are complete for KBs in the Horn form !!!

Predicate Logic

Predicate Logic (1)

- Propositional Logic can't say:
  - If X is married to Y, then Y is married to X.
  - If X is west of Y, and Y is west of Z, then X is west of Z.
  - And a million other simple things.
  - Fix it: Extend representation: add predicates. Extend operator(resolution): add unification

- Predicate logic or First-order logic (FOL) extends propositional logic to allow reasoning about the members (such as numbers) of some non-empty universe.

- A predicate is a generalization of a propositional variable. It uses the quantifiers ∀ (for all) and ∃ (there exists). Quantifiers are symbols used with propositional functions. There are two types of quantifiers as shown in the table below.

Name	Symbol	Meaning
Universal Quantifier	∀	" for all"
Existential Quantifier	∃	" there exists at least one"

- Eg: If N is a set of all positive numbers, then the following statements are true.
  - $\forall x \in N, (x + 3 > 2)$.
  - $\exists x \in N, (x + 2 < 7)$.

Predicate Logic (2)

- Variables are symbols capable of taking on any constant as value. In fact, a propositional variable is equivalent to a predicate with no arguments, and we shall write p for an atomic formula with predicate name p and zero arguments.
- An atomic formula is a predicate with zero or more arguments. For example, u(X) is an atomic formula with predicate u and one argument, here occupied by the variable X.
- An atomic formula all of whose arguments are constants is called a ground atomic formula.
- A nonground atomic formula can have constants or variables as formula arguments, but at least one argument must be a variable.
- A proposition is a predicate with no arguments, and therefore is a ground atomic formula.
- A literal is either an atomic formula or its negation.
- If there are no variables among the arguments of the atomic formula, then the literal is a ground literal.

- Naming conventions:
  - A variable name will always begin with an upper-case letter. E.g. X, A.
  - Constants are represented either by 1) Character strings beginning with a lower-case letter, 2) Numbers, like 12 or 14.3, or 3) Quoted character strings. E.g. 34, “AIML”, sy.
  - Predicates, like constants, will be represented by character strings beginning with a lower-case letter.
  - There is no possibility that we can confuse a predicate with a constant, since constants can only appear within argument lists in an atomic formula, while predicates cannot appear there.

Predicate Logic (3)

- The Elements of Predicate Logic Language:
- A predicate logic language *L* consists of the following symbols:
  - an infinite set of variables, denoted by uppercase letters;
  - a set of constants, denoted by lowercase letters, usually *a, b*, etc.;
  - a set of predicates, denoted by lowercase letters, usually *p, q,* etc.;
  - a set of functions, denoted by lowercase letters, usually *f, g*, etc.;
  - the connectives ¬, ∨, ∧, →, and ≡;
  - the quantifiers ∃ and ∀; and
  - the parentheses) and (.

- The Syntax of Predicate Logic Terms:
- The terms in *L* are defined recursively as follows:
  a) Every variable and every constant is a term.
  b) If $t_1, \ldots, t_n$ are terms and *f* is a function that takes *n* arguments, then $f(t_1, \ldots, t_n)$ is a term.
  c) Every term is obtained in this manner.
- Terms with no variables are called variable-free terms.

Predicate Logic (4)

- The Syntax of Predicate Logic Formulas:
- The formulas in *L* are defined recursively as follows.
  a) If $t_1, \ldots, t_n$ are terms and p is a predicate that takes n arguments, then $p(t_1, \ldots, t_n)$ is a formula, called an atomic formula.
  b) If α is a formula, so is (¬ α).
  c) If α and β are formulas, so are (α ∨ β), (α ∧ β), (α → β), and (α ≡ β).
  d) If *V* is a variable and α is a formula, then (∀*V* α) and (∃*V* α) are formulas.
  e) Every formula is obtained in this manner.

- Semantics for Predicate Logic:
- The syntactic definitions tell us how to construct everything in the language of first-order predicate calculus.
- As in propositional logic, the syntax tells us nothing about what our formulas "mean."
- To associate meaning with predicate logic formulas, we must know how to interpret them. Unfortunately, this is more complicated than the simple true/false of propositional logic.
- The approach logicians use to define semantics involves the discussion of models.
- Semantics will specify when a formula is true in a recursive manner that parallels the syntactic definition of the formula.

Predicate Logic (5)

- Informal Semantics for Predicate Logic:

  a) If p is a predicate and none of the terms $t_1, \ldots, t_n$ contains variables, then $p(t_1, \ldots, t_n)$ is either true or not according to the interpretation.

  b) If the truths of α and β are known, then the truth of connectives is determined by semantics (using truth tables) of logic formulae $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$, and $(\alpha \equiv \beta)$.

  c) Let V be a variable and α be a formula. If there is some constant c such that replacing every free occurrence of V in α with c gives a true formula, then $(\exists V\ \alpha)$ is true. (The restriction to free V is needed because a quantifier in α might bind some occurrences of a variable that is also called V.) ∃ is called an existential quantifier.

  d) Let V be a variable and α be a formula. If, for every constant c, replacing every free occurrence of V in α with c gives a true formula, then $(\forall V\ \alpha)$ is true. ∀ is called a universal quantifier.

- A formula is called valid or a tautology if and only if it is true for all possible interpretations.

Predicate Logic (6)

- Informal Semantics for Predicate Logic: (cont..)
- The definition of an informal semantics for predicate logic defines the truth and falsity of formulas only when there are no free occurrences of variables; so, don't try to apply it to a formula where a variable occurs freely. Also, as in propositional logic, the definition is often applied in the reverse direction of the definition of syntax—while syntax builds up, semantics tears down.

- E.g., consider $((\forall X\ p(X)) \rightarrow (\exists X\ p(X)))$ ----------(1)
- To determine the truth of a given predicate logic (1) we must first determine the truth of $(\forall X\ p(X))$ and $(\exists X\ p(X))$.
- There are three relevant possibilities for the truth of *p( )*. Here they are, along with their consequences.
  - *p(c)* is true for all *c*. In this case, both $(\forall X\ p(X))$ and $(\exists X\ p(X))$ are true. Thus (1) is true.
  - *p(c)* is false for all *e*. In this case, both $(\forall X\ p(X))$ and $(\exists X\ p(X))$ are true. Thus (1) is true.
  - *p(c)* is true for some *c* and false for some *c*. In this case, $(\forall X\ p(X))$ is false and $(\exists X\ p(X))$ is true. Thus (1) is true.

Predicate Logic (7)

- Quantifiers Examples: Quantifiers can apply to arbitrary expressions, not just to atomic formulas.
  1. "There exists an individual X such that X gets wet." The expression: (∃X)w(X).
  2. "If it rains, then for all individuals X, either X takes an umbrella or X gets wet."
     The expression: r → (∀X) (u(X) OR w(X))
  3. "Either all individuals X stay dry or, at least one individual Y gets wet."
     The expression: ((∀X) NOT w(X)) OR ((∃Y)w(Y))
- Precedence of Operators in Logical Expressions:
  - NOT (highest), AND, OR, →, and ≡ (lowest).
  - However, quantifiers have highest precedence of all.
- Order of Quantifiers:
  - A common logical mistake is to confuse the order of quantifiers.
  - E.g., to think that (∀X)(∃Y) means the same as (∃Y)(∀X), which it does not.
    - E.g., if we informally interpret loves(X, Y) as "X loves Y ," then (∀X)(∃Y)loves(X, Y) means "Everybody loves somebody," that is, for every individual X there is at least one individual Y that X loves.
    - On the other hand, (∃Y)(∀X)loves(X, Y) means that there is some individual Y who is loved by everyone — a very fortunate Y , if such a person exists.

Predicate Logic (8)

Q.9 Identify the following as constants, variables, ground atomic formulas, or nonground atomic formulas, using the conventions of predicate logic.

a) CS205 b) cs205 c) 205 d) “cs205” e) p(X, x) f) p(3, 4, 5)

Answer: a) CS205 : variable
b) cs205 : constant
c) 205 : constant
d) “cs205” : constant
e) p(X, x) : a nonground atomic formula
f) p(3, 4, 5) : a ground atomic formula

Predicate Logic (9)

Q.10 State TRUE or FALSE for the following statements.

1. p(X, a) is an atomic formula and a literal.
2. p(X, a) is a ground atomic formula.
3. NOT p(X, a) is an atomic formula and a literal.
4. NOT p(X, a) is a ground literal.
5. The expressions p(a, b) is a ground atomic formula and ground literal.
6. NOT p(a, b) is a ground atomic formula and ground literal.

Answer:

1. p(X, a) is an atomic formula and a literal. ------TRUE
2. p(X, a) is a ground atomic formula.------FALSE (It is not ground because of the argument X, which is a variable by our convention.)
3. NOT p(X, a) is an atomic formula and a literal. ------- FALSE (It is a literal, but not an atomic formula.)
4. NOT p(X, a) is a ground literal. -------- FALSE (It is not a ground literal.)
5. The expressions p(a, b) is a ground atomic formula and ground literal.------TRUE
6. NOT p(a, b) is a ground atomic formula and ground literal. -------- FALSE (It is not an atomic formula but is a ground literal.)

Predicate Logic (10)

Q. 11 What is the predicate calculus statement equivalent to the following?
"Every teacher is liked by some student"
a)$\forall x$ [teacher(x) → $\exists y$[student(y) → likes(y,x)]]
b)$\forall x$ [teacher(x) → $\exists y$[student(y) $\wedge$ likes(y,x)]]
c)$\exists y \forall x$[teacher(x) → [student(y) $\wedge$ likes(y,x)]]
d)$\forall x$ [teacher(x) $\wedge$ $\exists y$[student(y) → likes(y,x)]]

Answer: The statement given can also be written as "For all x, if x is a teacher, then there exists a student y who likes x", which can also be represented using the quantifiers as follows.
$\forall x$ [teacher(x) → $\exists y$[student(y) $\wedge$ likes(y,x)]].
It is important to note that implication (→) is almost used in conjunction with the quantifier $\forall$. Mostly the quantifier $\exists$ is associated with $\vee$.

Predicate Logic (11)

Q. 12 p(x): x is a human being.
f(x, y): x is father of y.
m(x, y): x is mother of y.
Write the predicate corresponding to
"x is the father of the mother of y"

Answer: If we try to interpret the predicate using three variables x, y, z we can write "z is a human being and x is the father of z and z is the mother of y". This can be represented as **(∃z)(p(z) ∧ f(x,z) ∧ m(z,y))**.

Predicate Logic (12)

- **Normal Forms:**
- Some important points to remember here are,
  1. An atomic proposition is a proposition containing no logical connectives. E.g.: **p, q, r** etc.
  2. A literal is either an atomic proposition or a negation of an atomic proposition. E.g.: **¬p, q, ¬r** etc.
  3. A conjunctive clause is a proposition that contains only literals and the connective ∧. E.g.: **(¬p ∧ q ∧ ¬r).**
  4. A disjunctive clause is a proposition that contains only literals and the connective ∨. E.g.: **(¬p ∨ q ∨ ¬r).**

- The problem of finding whether a given statement is a tautology or contradiction in a finite number of steps is called a decision problem. Constructing truth tables is not a practical way.
- We can therefore consider alternate procedure known as reduction to normal forms. Two such normal forms are:
  1. Disjunctive Normal form(DNF)
  2. Conjunctive Normal form(CNF)

Predicate Logic (13)

- **Disjunctive Normal form (DNF):**
  - A proposition is said to be in disjunctive normal form (DNF) if it is a disjunction of conjunctive clauses and literals.
    E.g.: **$(\neg p \wedge q \wedge \neg r) \vee q \vee (q \wedge r)$.**
  - A proposition is said to be in principal disjunctive normal form if it is a disjunction of conjunctive clauses only.
    E.g.**:** **$(\neg p \wedge q \wedge \neg r) \vee (q \wedge r)$.**
  - **Fundamental conjunction:** A conjunction of statement variables and (or) their negations is called as a fundamental conjunction(min term).

    E.g.: p, ~p, $p \wedge q$, $p \wedge \sim q \wedge \sim p$ are fundamental conjunctions.
  - A statement form which consist of disjunction of fundamental conjunctions is called disjunctive normal form

    E.g.: 1) $(p \wedge q) \vee \sim q$

    2) $(\sim p \wedge q) \vee (p \wedge q) \vee q$

    3) $(p \wedge \sim q) \vee (p \wedge r)$

Predicate Logic (14)

- **Conjunctive Normal form(CNF):**
  - A proposition is said to be in conjunctive normal form (CNF) if it is a conjunction of disjunctive clauses and literals.
    E.g.: **$(\neg p \vee q \vee \neg r) \wedge r \wedge (q \vee r)$.**
  - A proposition is said to be in principal conjunctive normal form if it is a conjunction of disjunctive clauses only.
    E.g.: **$(\neg p \vee q \vee \neg r) \wedge (q \vee r)$.**
  - **Fundamental disjunction:-**A disjunction of statement variables and (or) their negations is called as a fundamental disjunction (max term).
    E.g.: p, ~p, p V q , p V ~q V ~p are fundamental disjunction.
  - A statement form which consist of conjunction of fundamental disjunctions is called conjunctive normal form
    E.g.: 1) $p \wedge q$
    2) $\sim p \wedge (p \vee q)$
    3) $(p \vee q \vee r) \wedge (\sim p \vee r)$

Predicate Logic (15)

Q. 13 What is the disjunctive normal form of $p \wedge (p \rightarrow q)$?

Answer: $(p \wedge \neg p) \vee (p \wedge q)$

Q. 14 Obtain the DNF of the form $(p \rightarrow q) \wedge (\sim p \wedge q)$.

Answer: $(p \rightarrow q) \wedge (\sim p \wedge q) = (\sim p \vee q) \wedge (\sim p \wedge q)$ ……..(as $p \rightarrow q = \sim p \vee q$)

$= (\sim p \wedge \sim p \wedge q) \vee (q \wedge \sim p \wedge q)$

$= (\sim p \wedge q) \vee (q \wedge \sim p)$----------DNF

Q. 15 Obtain the DNF of $(p \wedge (p \rightarrow q)) \rightarrow q$.

Answer: $(p \wedge (p \rightarrow q)) \rightarrow q = \sim(p \wedge (p \rightarrow q)) \vee q$

$= (\sim p \vee \sim(p \rightarrow q)) \vee q$

$= \sim p \vee \sim(\sim p \vee q) \vee q$

$= \sim p \vee (p \wedge \sim q) \vee q$----------DNF

Predicate Logic (16)

Q. 16 What is the conjunctive normal form of $(\sim p \rightarrow r) \wedge (p <-> q)$?

Answer: $(\sim p \rightarrow r) \wedge (p <-> q) = (\sim p \rightarrow r) \wedge ((p \rightarrow q) \wedge (q \rightarrow p))$

$= (\sim(\sim p) \vee r) \wedge ((\sim p \vee q) \wedge (\sim q \vee p))$

$= (p \vee r) \wedge (\sim p \vee q) \wedge (\sim q \vee p)$ ----------CNF

Q. 17 Obtain the CNF of the form $(p \wedge q) \vee (\sim p \wedge q \wedge r)$.

Answer: $(p \wedge q) \vee (\sim p \wedge q \wedge r) = (p \vee (\sim p \wedge q \wedge r)) \wedge (q \vee (\sim p \wedge q \wedge r))$

$= ((p \vee \sim p) \wedge (p \vee q) \wedge (p \vee r)) \wedge ((q \vee \sim p) \wedge (q \vee q) \wedge (q \vee r))$

$= (T \wedge (p \vee q) \wedge (p \vee r)) \wedge ((q \vee \sim p) \wedge (q) \wedge (q \vee r)$

$= (p \vee q) \wedge (p \vee r) \wedge (q \vee \sim p) \wedge q \wedge (q \vee r)$ ----------CNF

Predicate Logic (17)

- Translating from English to predicate logic:
- E.g., A Lewis Carroll Example:
- The Oxford geometer Charles Dodgson is famous for writing *Alice in Wonderland* and *Through the Looking Glass* under the pen name Lewis Carroll.
- Two years before his death, he published a symbolic logic text containing delightful problems. We'll look at one of his problems in this example.

  1) Colored flowers are always scented;
  2) I dislike flowers that are not grown in the open air;
  3) No flowers grown in the open air are colorless.

  How can we recast these in terms of predicate logic?

Predicate Logic (18)

- A Lewis Carroll Example (cont..)

Answer: Our constants will be flowers and our predicates will be as follows:

- c(X): indicates X is colored,
- d(X): indicates I dislike X,
- g(X): indicates X is grown in the open air,
- s(X): indicates X is scented.

- You should be able to verify that the following are translations of the statements:

1) $\forall X\ (c(X) \rightarrow s(X))$
2) $\forall X\ ((\neg\ g(X)) \rightarrow d(X))$
3) $\neg(\exists X\ (g(X) \wedge (\neg\ c(X))))$

- Other translations are possible, but these are the most direct and also reflect the meaning of the English.
- In this case, one possible conclusion is $\forall X\ ((\neg\ s(X)) \rightarrow d(X))$; that is, I dislike all flowers that are not scented.

Predicate Logic (19)

Q. 18 Translate the following English sentences into predicate logic or first-order-logic.

1. Every kid eats every chocolate.
2. Anyone who eats some chocolate is not a nutrition fanatic.
3. Anyone who eats a watermelon is a nutrition fanatic.
4. Anyone who buys any watermelon either carves it or eats it.
5. Seema is a kid.
6. Seema buys a watermelon.

Answer:

1. $\forall X, \forall Y: ((kid(X) \wedge chocolate(Y)) \rightarrow eats(X,Y))$
2. $\forall X, \exists Y: ((chocolate(Y) \wedge eats(X,Y)) \rightarrow \neg$ nutrition-fanatic(X)) …{Conclusion from 1 & 2:- $\forall X:(kid(X) \rightarrow \neg$ nutrition-fanatic(X))}
3. $\forall X, \exists Y : ((watermelon(Y) \wedge eats(X,Y)) \rightarrow$ nutrition-fanatic(X))
4. $\forall X, \forall Y : ((watermelon(Y) \wedge buys(X,Y)) \rightarrow (carves(X,Y) \vee eats(X, Y)))$
5. kid(Seema) ……{Conclusion from 1, 2 & 5:- $\neg$ nutrition-fanatic(Seema)}
6. $\exists Y : (watermelon(Y) \rightarrow buys(Seema, Y))$ ……{Conclusion from 4 & 5:- $(carves(Seema, Y) \vee eats(Seema, Y)$}

Predicate Logic (20)

Q.19 Translate the following English sentences into predicate logic or first-order-logic.

1. There is a barber in town who shaves all men in town who do not shave themselves.
2. There is a barber in town who shaves only and all men in town who do not shave themselves.

Answer:

1. $\exists x\ (Barber(x) \wedge InTown(x) \wedge \forall y\ (Man(y) \wedge InTown(y) \wedge \neg Shave(y,y) \Rightarrow Shave(x,y)))$
2. $\exists x\ (Barber(x) \wedge InTown(x) \wedge \forall y\ (Man(y) \wedge InTown(y) \wedge \neg Shave(y,y) \Leftrightarrow Shave(x,y)))$

Predicate Logic (21)

- **Connections between ∀ and ∃:**
- The two quantifiers are actually intimately connected with each other, through negation.
- Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa: $\forall x \neg Likes(x,Parsnips) \equiv \neg \exists x\ Likes(x,Parsnips)$.
- We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream: $\forall x\ Likes(x, IceCream) \equiv \neg \exists x \neg Likes(x, IceCream)$.
- Because ∀ is really a conjunction over the universe of objects and ∃ is a disjunction, it should not be surprising that they obey De Morgan's rules.
- The De Morgan rules for quantified and unquantified sentences are as follows:

  $\forall x \neg P \equiv \neg \exists x\ P$ $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$

  $\neg \forall x\ P \equiv \exists x \neg P$ $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

  $\forall x\ P \equiv \neg \exists x \neg P$ $P \wedge Q \equiv \neg(\neg P \vee \neg Q)$

  $\exists x\ P \equiv \neg \forall x \neg P$ $P \vee Q \equiv \neg(\neg P \wedge \neg Q)$.
- Thus, we do not really need both ∀ and ∃, just as we do not really need both ∧ and ∨.

Predicate Logic (22)

- Let α be a formula in which any occurrences of X and Y are free and let β be a formula with no free X. Let * be either $\vee$ or $\wedge$. Then the following formulas are tautologies:

  $(\forall X(\forall Y\ \alpha)) \equiv (\forall Y(\forall X\ \alpha))$ and $(\exists X(\exists Y\ \alpha)) \equiv (\exists Y/(\exists X\ \alpha))$.....(1)

  $(\neg(\forall X\ \alpha)) \equiv (\exists X\ \neg\alpha)$ and $(\neg(\exists X\ \alpha)) \equiv (\forall X\ \neg\alpha)$.....(2)

  $((\forall X\ \alpha) * \beta) \equiv (\forall X(\alpha * \beta)$ and $((\exists X\ \alpha) * \beta) \equiv (\exists X\ (\alpha * \beta))$.....(3)

- **Proof:** Abuse notation and write $\alpha(X)$ to indicate the free occurrences of X in α. Use $\leftrightarrow$ to stand for the phrase "if and only if." Let's prove (2) here and leave the rest as an exercise.
- By the definition of the semantics,

  $\neg(\forall X\ \alpha(X))$ is true $\leftrightarrow \forall X\ \alpha(X)$ is false

  $\leftrightarrow$ there is some constant c such that $\alpha(c)$ is false

  $\leftrightarrow (\exists X \neg\ \alpha(X))$ is true.

- The right side of (2) follows similarly.

Unification

Unification (1)

- To apply inference rules inference system must be able to determine when two expressions are the same or match.
- In propositional calculus, this is trivial: two expressions match if and only if they are syntactically identical.
- In predicate calculus, the process of matching two sentences is complicated by the existence of variables in the expressions.
- Universal instantiation allows universally quantified variables to be replaced by terms from the domain.
- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let p and q be two atomic sentences and σ be a unifier such that, $p\sigma = q\sigma$, then it can be expressed as UNIFY(p, q).
- In order to match antecedents to existing literals in the KB, need a pattern matching routine.

Unification (2)

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms. It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.
- Examples:

1. UNIFY(Parent(x,y), Parent(Tom, John)) = {x/Tom, y/John}
2. UNIFY(Parent(Tom,x), Parent(Tom, John)) = {x/John})
3. UNIFY(Likes(x,y), Likes(z,FOL)) = {x/z, y/FOL}
4. UNIFY(Likes(Tom,y), Likes(z,FOL)) = {z/Tom, y/FOL}
5. UNIFY(Likes(Tom,y), Likes(y,FOL)) = fail
6. UNIFY(Likes(Tom,Tom), Likes(x,x)) = {x/Tom}
7. UNIFY(Likes(Tom,Fred), Likes(x,x)) = fail

Unification (3)

Q. 20 Find the MGU for Unify {King(x), King(John)}

Answer:

- Let p = King(x), q = King(John),
- Substitution σ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

Q. 21 Find the MGU for Unify { P(x,y), P(a,f(z)}

Answer:

- n this example, we need to make both above statements identical to each other. For this, we will perform the substitution: 1) P(x, y), 2) P(a, f(z)).
- Substitute x with a, and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set (MGU) will be: [a/x, f(z)/y].

Unification (4)

- Conditions for Unification:
  - Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
  - Number of arguments in both expressions must be identical.
  - Unification will fail if there are two similar variables present in the same expression.
  - All variables should be Universally Quantified. This allows full freedom in computing substitutions.
  - Existentially quantified variables may be eliminated from sentences in the database by replacing them with the constants that make the sentence true.

- Implementation of the Algorithm:

Step 1: Initialize the substitution set to be empty.

Step 2: Recursively unify atomic sentences:

a) Check for Identical expression match.

b) If one expression is a variable v_i and the other is a term t_i which does not contain variable v_i, then:

   i. Substitute t_i / v_i in the existing substitutions.

   ii. Add t_i / v_i to the substitution setlist.

   iii. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

Unification (5)

- Exact variable names used in sentences in the KB should not matter.
  - But if Likes(x, FOL) is a formula in the KB, it does not unify with Likes(John, x) but does unify with Likes(John, y).
- To avoid such conflicts, one can standardize apart one of the arguments to UNIFY to make its variables unique by renaming them.
  - Likes(x, FOL) → Likes(x1, FOL)
  - UNIFY(Likes(John, x), Likes(x1,FOL)) = {x1/John, x/FOL}
- There are many possible unifiers for some atomic sentences.
  - UNIFY (Likes(x,y), Likes (z,FOL)) = {x/z, y/FOL}
  - {x/John, z/John, y/FOL}
  - {x/Fred, z/Fred, y/FOL}
- UNIFY should return the most general unifier which makes the least commitment to variable values.
- Inference using unification:

$$\frac{\forall x.\ \neg P(x) \vee Q(x) \qquad P(A)}{Q(A)}$$

  - For universally quantified variables, find MGU {x/A} and proceed as in propositional resolution.

Skolemization

Skolemization (1)

- Conversion of sentences FOL to CNF requires skolemization.
- Skolemization: remove existential quantifiers by introducing new function symbols.
- Special case: introducing constants (trivial functions: no previous universal quantifier). Variables bound by existential quantifiers which are not inside the scope of universal quantifiers can simply be replaced by a Skolem term or constants. E.g. $\exists x\,[x < 3]$ can be changed to $c < 3$, with c a suitable constant.
- When the existential quantifier is inside a universal quantifier, the bound variable must be replaced by a Skolem function of the variables bound by universal quantifiers. Replace any existentially quantified variable $\exists x$ that is in the scope of universally quantified variables $\forall y_1 \ldots \forall y_n$ with a new function $F(y_1,\ldots,y_n)$ (a Skolem function). Thus $\forall x\,[x=0 \vee \exists y\,[x = y+1]]$ becomes $\forall x[x=0 \vee x= f(x)+1]$.
- For each existentially quantified variable introduce a n-place function where n is the number of previously appearing universal quantifiers.
- In general, the functions and constants symbols are new ones added to the language for the purpose of satisfying these formulas, and are often denoted by the formula they realize, for instance $c_{\exists x\phi(x)}$.

Skolemization (2)

- ∀x ∀y ∃w ∀z Q(x, y, w, z, G(w, x)) is equivalent to ∀x ∀y ∀z Q(x, y, P(x, y), z, G(P(x, y), x)) where P is the Skolem function for w.

- Examples:
  - Every philosopher writes at least one book.
    $\forall x[Philo(x) \rightarrow \exists y[Book(y) \wedge Write(x, y)]]$
  - Eliminate Implication:
    $\forall x[\neg Philo(x) \vee \exists y[Book(y) \wedge Write(x, y)]]$
  - Skolemize: substitute y by $g(x)$
    $\forall x[\neg Philo(x) \vee [Book(g(x)) \wedge Write(x, g(x))]]$

  - All students of a philosopher read one of their teacher's books.
    $\forall x \forall y[Philo(x) \wedge StudentOf(y, x) \rightarrow$
    $\exists z[Book(z) \wedge Write(x, z) \wedge Read(y, z)]]$
  - Eliminate Implication:
    $\forall x \forall y[\neg Philo(x) \vee \neg StudentOf(y, x) \vee \exists z[Book(z) \wedge$
    $Write(x, z) \wedge Read(y, z)]]$
  - Skolemize: substitute z by $h(x, y)$
    $\forall x \forall y[\neg Philo(x) \vee \neg StudentOf(y, x) \vee [Book(h(x, y)) \wedge$
    $Write(x, h(x, y)) \wedge Read(y, h(x, y))]]$

Resolution of Predicate Logic

Resolution of Predicate Logic (1)

- Conjunctive normal form for first-order logic:
- As in the propositional case, first-order resolution requires that sentences be in conjunctive normal form (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.

- Literals can contain variables, which are assumed to be universally quantified.
- E.g., the sentence $\forall x \text{ American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \rightarrow \text{Criminal}(x)$ becomes, in CNF,

$$\neg\text{American}(x) \lor \neg\text{Weapon}(y) \lor \neg\text{Sells}(x, y, z) \lor \neg\text{Hostile}(z) \lor \text{Criminal}(x).$$

- **Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.**

- In particular, the CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable, so we have a basis for doing proofs by contradiction on the CNF sentences.

- To carry out a resolution proof for FOL we must rewrite a formula so that all the quantifiers are universal and on the outside while the formula inside the quantifiers is in clausal form.

Resolution of Predicate Logic (2)

- This can be done as follows:
  - Adapt the conversion rules to produce a clausal form containing quantifiers.

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg(f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

  - Use eq.(2) to move negation inward through quantifiers. …{(¬(∀X α)) ≡ (∃X ¬α) and (¬(∃X α)) ≡ (∀X ¬α)…(2)}
  - Assign unique names to all quantified (=bound) variables and then use eq.(3) to move the quantifiers to the left side of the formula. The result is said to be in *prenex form*. …{((∀X α) * β) ≡ (∀X(α * β) and ((∃X α) * β) ≡ (∃X (α * β))…(3)}
  - Use "Skolemization" to eliminate existential quantifiers.
- Once this normal form has been achieved, resolution can begin; however, it's complicated by the need for "unification." The unification must be done so as not to impose any equalities that are not absolutely required—the "most general unification."

Resolution of Predicate Logic (3)

We illustrate the procedure by translating the sentence "Everyone who loves all animals is loved by someone," or

$$\forall x \ [\forall y \ \ Animal(y) \ \Rightarrow \ Loves(x, y)] \ \Rightarrow \ [\exists y \ \ Loves(y, x)] \ .$$

The steps are as follows:

- **Eliminate implications**:

  $$\forall x \ [\neg\forall y \ \neg Animal(y) \vee Loves(x, y)] \vee [\exists y \ \ Loves(y, x)] \ .$$

- **Move $\neg$ inwards**: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

  $$\begin{array}{lll} \neg\forall x \ p & \text{becomes} & \exists x \ \neg p \\ \neg\exists x \ p & \text{becomes} & \forall x \ \neg p \ . \end{array}$$

  Our sentence goes through the following transformations:

  $$\begin{array}{l} \forall x \ [\exists y \ \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y \ \ Loves(y, x)] \ . \\ \forall x \ [\exists y \ \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)] \ . \\ \forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)] \ . \end{array}$$

  Notice how a universal quantifier ($\forall y$) in the premise of the implication has become an existential quantifier. The sentence now reads "Either there is some animal that x doesn't love, or (if this is not the case) someone loves x." Clearly, the meaning of the original sentence has been preserved.

Resolution of Predicate Logic (4)

- **Standardize variables**: For sentences like $(\exists x\, P(x)) \vee (\exists x\, Q(x))$ which use the same variable name twice, change the name of one of the variables. This avoids confusion later when we drop the quantifiers. Thus, we have

$$\forall x \;\; [\exists y \;\; Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \;\; Loves(z, x)] \; .$$

- **Skolemize**: **Skolemization** is the process of removing existential quantifiers by elimination. In the simple case, it is just like the Existential Instantiation rule of Section 9.1: translate $\exists x\, P(x)$ into $P(A)$, where A is a new constant. However, we can't apply Existential Instantiation to our sentence above because it doesn't match the pattern $\exists v\, \alpha$; only parts of the sentence match the pattern. If we blindly apply the rule to the two matching parts we get

$$\forall x \;\; [Animal(A) \wedge \neg Loves(x, A)] \vee Loves(B, x) \; ,$$

which has the wrong meaning entirely: it says that everyone either fails to love a particular animal A or is loved by some particular entity B. In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus, we want the Skolem entities to depend on x and z:

$$\forall x \;\; [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(z), x) \; .$$

Here F and G are **Skolem functions**. The general rule is that the arguments of the Skolem function are all the universally quantified variables in whose scope the existential quantifier appears. As with Existential Instantiation, the Skolemized sentence is satisfiable exactly when the original sentence is satisfiable.

Resolution of Predicate Logic (5)

- **Drop universal quantifiers**: At this point, all remaining variables must be universally quantified. Moreover, the sentence is equivalent to one in which all the universal quantifiers have been moved to the left. We can therefore drop the universal quantifiers:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(z), x) \ .$$

- **Distribute $\vee$ over $\wedge$**:

$$[Animal(F(x)) \vee Loves(G(z), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(z), x)] \ .$$

This step may also require flattening out nested conjunctions and disjunctions.

The Resolution Inference Rule

- Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.
- Propositional literals are complementary if one is the negation of the other; first-order literals are complementary if one unifies with the negation of the other. Thus, we have

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

where $\text{UNIFY}(\ell_i, \neg m_j) = \theta$. For example, we can resolve the two clauses

$$[Animal(F(x)) \vee Loves(G(x), x)] \quad \text{and} \quad [\neg Loves(u, v) \vee \neg Kills(u, v)]$$

by eliminating the complementary literals $Loves(G(x), x)$ and $\neg Loves(u, v)$, with unifier $\theta = \{u/G(x), v/x\}$, to produce the **resolvent** clause

$$[Animal(F(x)) \vee \neg Kills(G(x), x)] \ .$$

This rule is called the **binary resolution** rule because it resolves exactly two literals. The binary resolution rule by itself does not yield a complete inference procedure. The full resolution rule resolves subsets of literals in each clause that are unifiable. An alternative approach is to extend **factoring**—the removal of redundant literals—to the first-order case. Propositional factoring reduces two literals to one if they are *identical*; first-order factoring reduces two literals to one if they are *unifiable*. The unifier must be applied to the entire clause. The combination of binary resolution and factoring is complete.

Forward Chaining (1)

- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. It is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to derive all consequences until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.
- Inferences cascade to draw deeper and deeper conclusions.
- To avoid looping and duplicated effort, must prevent addition of a sentence to the KB which is the same as one already present.
- Must determine all ways in which a rule (Horn clause) can match existing facts to draw new conclusions.

- Properties of Forward-Chaining:
  - It is a bottom-up approach, as it moves from bottom to top.
  - It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
  - It is also called as data-driven as we reach to the goal using available data.
  - It is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Forward Chaining (2)

- Example 1: Assume in KB:
  1) parent(X,Y) ∧ male(X) ⇒ father(X,Y)
  2) father(X,Y) ∧ father(X,Z) ⇒ sibling(Y,Z)
- Add to KB: 3) parent(Tom, John)
- Rule 1) tried but can't "fire"
- Add to KB: 4) male(Tom)
- Rule 1: now satisfied and triggered and adds:
  5) father(Tom, John)
- Rule 2: now triggered and adds:
  6) sibling(John, John) {X/Tom, Y/John, Z/John}
- Add to KB: 7) parent(Tom, Fred)
- Rule 1: triggered again and adds:
  8) father(Tom, Fred)
- Rule 2: triggered again and adds:
  9) sibling(Fred, Fred) {X/Tom, Y/Fred, Z/Fred}
- Rule 2: triggered again and adds:
  10) sibling(John, Fred) {X/Tom, Y/John, Z/Fred}
- Rule 2: triggered again and adds:
  11) sibling(Fred, John) {X/Tom, Y/Fred, Z/John}

procedure FORWARD-CHAIN(KB, p)

if there is a sentence in KB that is a renaming of p **then return**
Add p to KB
for each $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ **in** KB such that for some i, UNIFY$(p_i, p) = \theta$ succeeds **do**
  FIND-AND-INFER$(KB, [p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n], q, \theta)$
end

---

procedure FIND-AND-INFER($KB, premises, conclusion, \theta$)

if $premises = [\,]$ **then**
  FORWARD-CHAIN$(KB,$ SUBST$(\theta, conclusion))$
else for each p' **in** KB such that UNIFY$(p',$ SUBST$(\theta,$ FIRST$(premises))) = \theta_2$ **do**
  FIND-AND-INFER$(KB,$ REST$(premises), conclusion,$ COMPOSE$(\theta, \theta_2))$
end

Forward Chaining (3)

- Example2:**"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."** Prove that **"Robert is criminal."**
- **Solution:**
- To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

A) Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say P, Q, and R are variables): American (P) ∧ weapon(Q) ∧ sells (P, Q, R) ∧ hostile(R) → criminal(P) ...(1)
- Country A has some missiles: ∃P missile(P) ∧ owns(A, P) . It can be written in two definite clauses by using Existential Instantiation, introducing new constant t1: owns(A, t1) .....(2), missile(t1) .....(3)
- All of the missiles were sold to country A by Robert. ∃P missile(P) ∧ owns(A, P) → sells (Robert, P, A) .....(4)
- Missiles are weapons: missile(P) → weapon(P) .....(5)
- Enemy of America is known as hostile: enemy(A, America) →hostile(P) .....(6)
- Country A is an enemy of America: enemy (A, America) .....(7)
- Robert is American: American(Robert) .....(8)

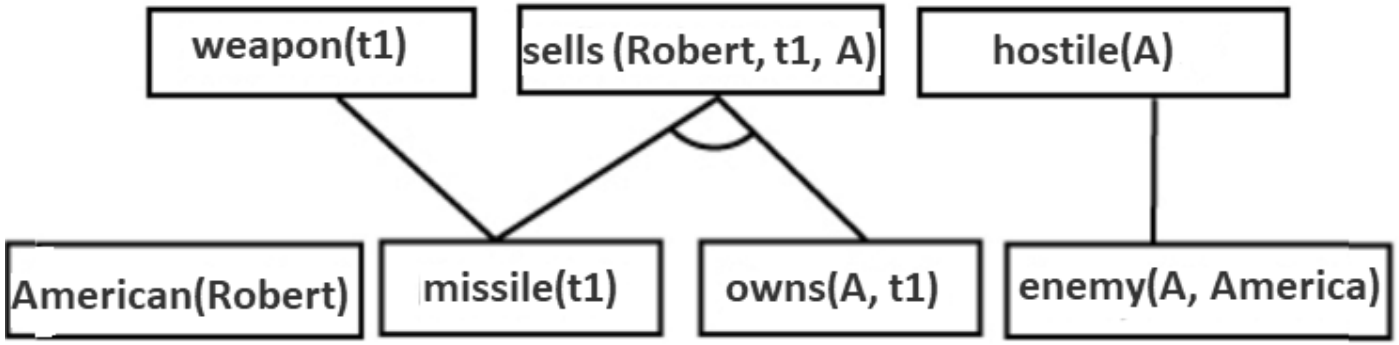
Forward Chaining (4)

- Example2 Solution (cont..):

B) Forward Chaining Proof:

- **Step-1:** In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), enemy(A, America), owns(A, t1), and missile(t1)**. All these facts will be represented as below.

- **Step-2:** At the second step, we will see those facts which infer from available facts and with satisfied premises.
  - Rule-(1) does not satisfy premises, so it will not be added in the first iteration.
  - Rule-(2) and (3) are already added.
  - Rule-(4) satisfy with the substitution {p/t1}, so sells (Robert, t1, A) is added, which infers from the conjunction of Rule (2) and (3).

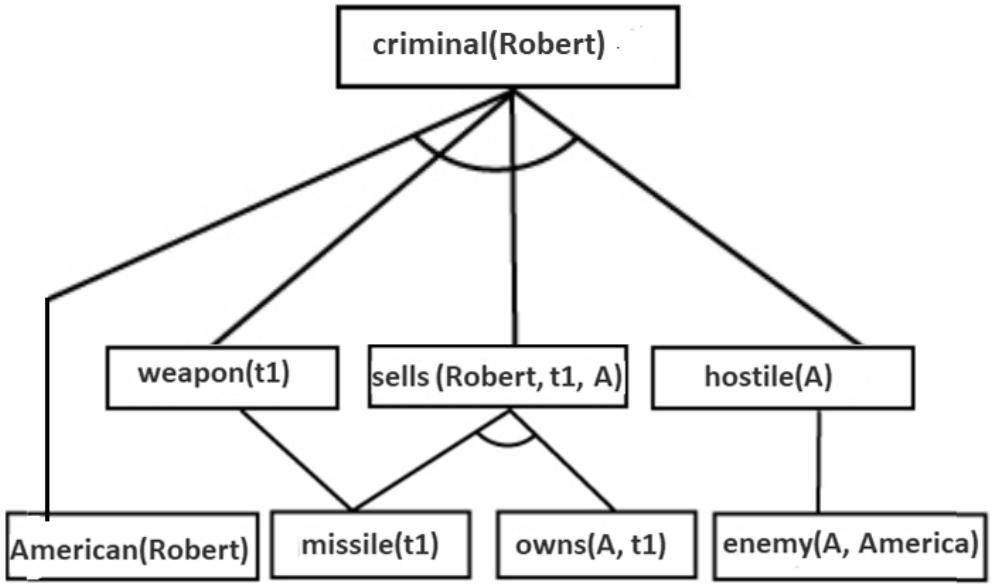


Forward Chaining (4)

- Example2 Solution (cont..):

B) Forward Chaining Proof:

- **Step-3:** At step-3, as we can check Rule-(1) is satisfied with the substitution {P/Robert, Q/t1, R/A}, so we can add criminal(Robert) which infers all the available facts. And hence we reached our goal statement.



- **Hence it is proved that Robert is Criminal using forward chaining approach.**

Backward Chaining (1)

- Backward chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.
- Given a conjunction of queries, first get all possible answers to the first conjunct and then for each resulting substitution try to prove all of the remaining conjuncts.
- Assume variables in rules are renamed (standardized apart) before each use of a rule.

- Properties of Forward-Chaining:
  - It is known as a top-down approach.
  - It is based on modus ponens inference rule.
  - In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
  - It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
  - It is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
  - The backward-chaining method mostly used a depth-first search strategy for proof.

Backward Chaining (2)

- Example 1:
- KB: 1) parent(X,Y) ∧ male(X) ⇒ father(X,Y)
  2) father(X,Y) ∧ father(X,Z) ⇒ sibling(Y,Z)
  3) parent(Tom, John)
  4) male(Tom)
  7) parent(Tom, Fred)

- Query: parent(Tom, X)
  Answers: ({X/John}, {X/Fred})

- Query: father(Tom, S)
  Subgoal: Parent(Tom, S) ∧ male(Tom)
  {S/John}
  Subgoal: Male(Tom)
  Answer: {S/John}
  {S/Fred}
  Subgoal: Male(Tom)
  Answer: {S/Fred}
  Answers: ({S/John}, {S/Fred})

function BACK-CHAIN(KB, q) **returns** a set of substitutions

BACK-CHAIN-LIST(KB, $[q]$, $\{\}$)

---

function BACK-CHAIN-LIST(KB, $qlist$, θ) **returns** a set of substitutions
inputs: KB, a knowledge base
$qlist$, a list of conjuncts forming a query (θ already applied)
θ, the current substitution
static: *answers*, a set of substitutions, initially empty

if $qlist$ is empty **then return** $\{\theta\}$
$q \leftarrow$ FIRST($qlist$)
for each q'_i **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
Add COMPOSE(θ, θ_i) to *answers*
end
for each sentence $(p_1 \wedge \ldots \wedge p_n \Rightarrow q'_i)$ **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
answers $\leftarrow$ BACK-CHAIN-LIST(KB, SUBST(θ_i, $[p_1 \ldots p_n]$), COMPOSE(θ, θ_i)) $\cup$ *answers*
end
return the union of BACK-CHAIN-LIST(KB, REST($qlist$), θ) for each $\theta \in$ *answers*

Backward Chaining (3)

(Example1 Cont..)

- Query: father(F, S)

  Subgoal: parent(F, S) ∧ male(F)

  {F/Tom, S/John}

  Subgoal: male(Tom)

  Answer: {F/Tom, S/John}

  {F/Tom, S/Fred}

  Subgoal: male(Tom)

  Answer: {F/Tom, S/Fred}

  Answers: ({F/Tom, S/John}, {F/Tom, S/Fred})

(Example1 Cont..)

- Query: sibling(A, B)

  Subgoal: father(F, A) ∧ father(F, B)

  {F/Tom, A/John}

  Subgoal: father(Tom, B)

  {B/John}

  Answer: {F/Tom, A/John, B/John}

  {B/Fred}

  Answer: {F/Tom, A/John, B/Fred}

  {F/Tom, A/Fred}

  Subgoal: father(Tom, B)

  {B/John}

  Answer: {F/Tom, A/Fred, B/John}

  {B/Fred}

  Answer: {F/Tom, A/Fred, B/Fred}

  Answers: ({F/Tom, A/John, B/John},{F/Tom, A/John, B/Fred}

  {F/Tom, A/Fred, B/John}, {F/Tom, A/Fred, B/Fred})

Backward Chaining (4)

- Example2:**"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."** Prove that **"Robert is criminal."**
- **Solution:**
- To solve the above problem, by backward-chaining algorithm we will write all the rules in FOL.

A) Facts Conversion into FOL:

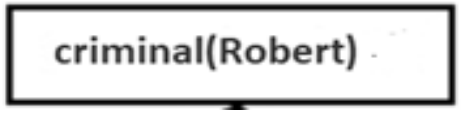
- It is a crime for an American to sell weapons to hostile nations. (Let's say P, Q, and R are variables): American (P) ∧ weapon(Q) ∧ sells (P, Q, R) ∧ hostile(R) → criminal(P) ...(1)
- Country A has some missiles: ∃P missile(P) ∧ owns(A, P) . It can be written in two definite clauses by using Existential Instantiation, introducing new constant t1: owns(A, t1) .....(2), missile(t1) .....(3)
- All of the missiles were sold to country A by Robert. ∃P missile(P) ∧ owns(A, P) → sells (Robert, P, A) .....(4)
- Missiles are weapons: missile(P) → weapon(P) .....(5)
- Enemy of America is known as hostile: enemy(A, America) →hostile(P) .....(6)
- Country A is an enemy of America: enemy (A, America) .....(7)
- Robert is American: American(Robert) .....(8)

Backward Chaining (5)

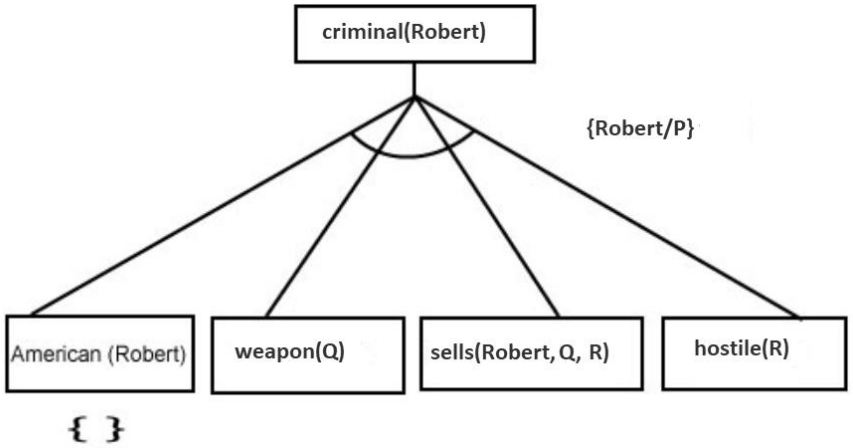
- Example2 Solution (cont..):

B) Backward Chaining Proof:

- In Backward chaining, we will start with our goal predicate, which is criminal(Robert), and then infer further rules.
- **Step-1:** At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So, our goal fact is "Robert is Criminal," so following is the predicate of it.



- **Step-2:** At the second step, we will infer other facts form goal fact which satisfies the rules. So, as we can see in Rule-1, the goal predicate criminal (Robert) is present with substitution {Robert/P}. So, we will add all the conjunctive facts below the first level and will replace P with Robert. Here we can see American(Robert) is a fact, so it is proved here.

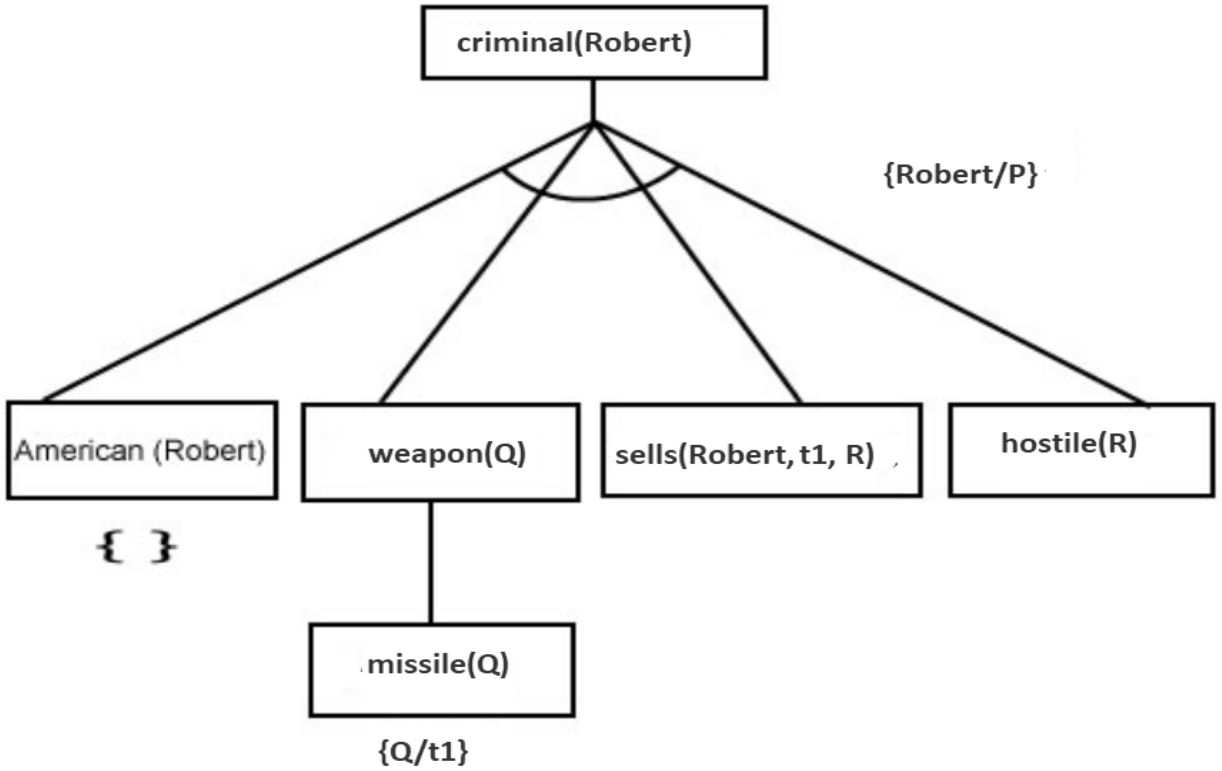


Backward Chaining (6)

- Example2 Solution (cont..):

B) Backward Chaining Proof:

- **Step-3:** At step-3, we will extract further fact missile(Q) which infer from weapon(Q), as it satisfies Rule-(5). weapon(Q) is also true with the substitution of a constant t1 at Q.

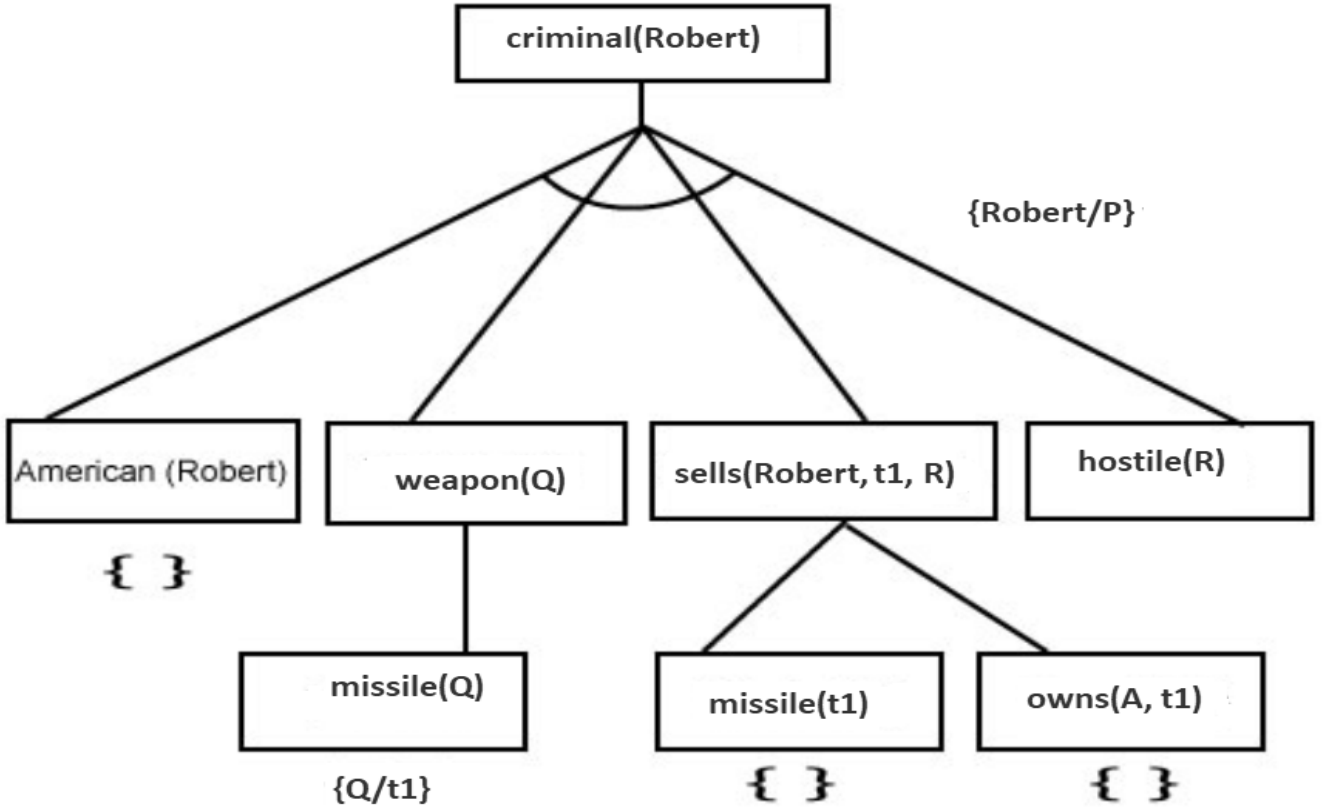


Backward Chaining (7)

- Example2 Solution (cont..):

B) Backward Chaining Proof:

- **Step-4:** At step-4, we can infer facts missile(t1) and owns(A, t1) form sells(Robert, t1, R) which satisfies the Rule- 4, with the substitution of A in place of R. So, these two statements are proved here.

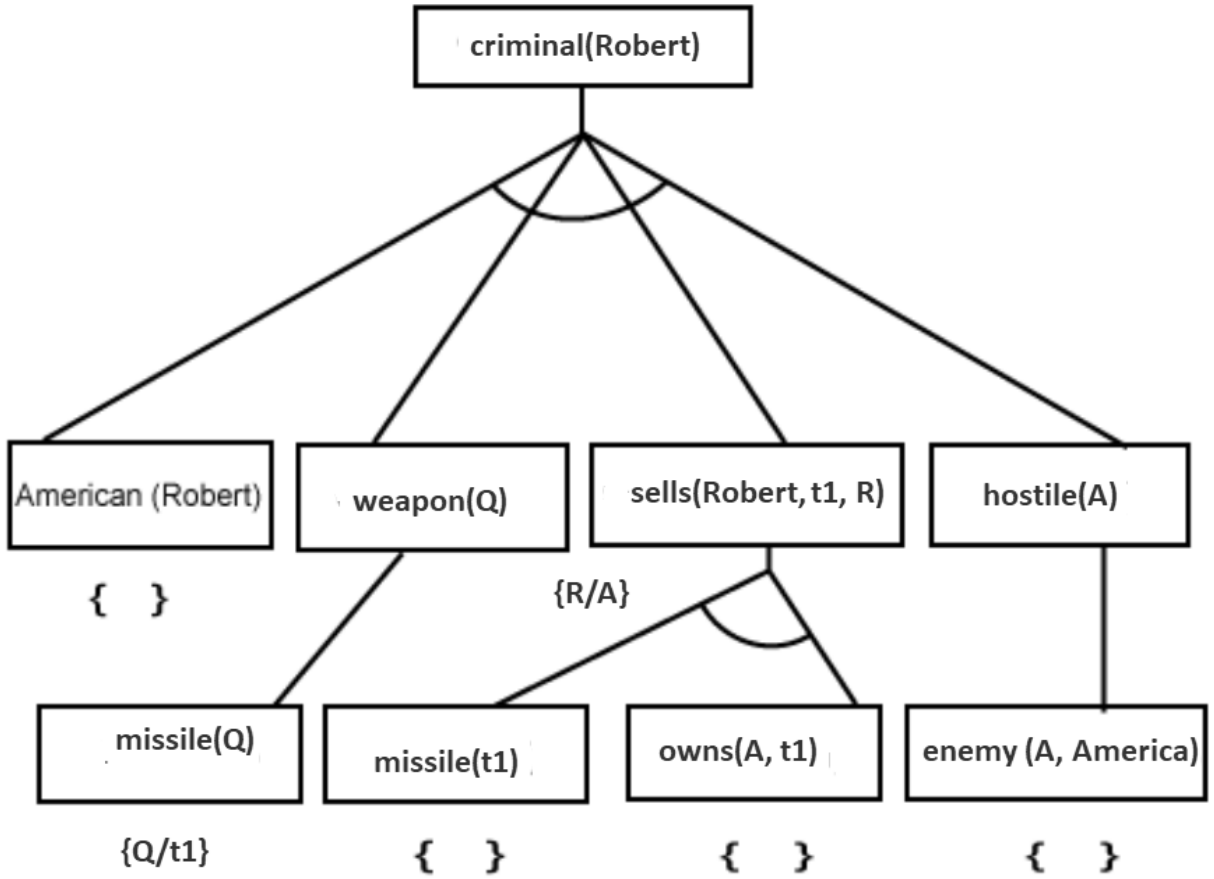


Backward Chaining (8)

- Example2 Solution (cont..):

B) Backward Chaining Proof:

- **Step-5:** At step-5, we can infer the fact enemy(A, America) from hostile(A) which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Thank you!