UNIT -II

- ✓ **What is knowledge representation?**
- ✓ **Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.**

What to Represent:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term)

Types of knowledge

- **1. Declarative Knowledge:**
- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

Types of knowledge

Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

Types of Knowledge

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

CATEGORIES AND OBJECTS

- In this chapter we address the question of what *content* to put into such an agent's knowledge base-how to represent facts about the world.
- The organization of objects into **categories** is a vital part of knowledge representation.
- There are two chalices for representing categories in first-order logic: predicates and objects.

CATEGORIES AND OBJECTS………

- Categories serve to organize and simplify the knowledge base through **inheritance**.
- Subclass relations organize categories into a **taxonomy, or taxonomic hierarchy**

First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members:

- An object is a member of a category. For example:

  BB9 E *Basketballs*
- A category is a subclass of another category. For example:

  Basketballs C Balls
- All members of a category have some properties. For example:

  x E *Basketballs* + *Round (x)*
- Members of a category can be recognized by some properties. For example:

  Orange (x) A Round (z) A Diameter(x) = 9.5" *A x* E *Balls* + *x* E *Basketballs*
- A category as a whole has some properties. For example:

  Dogs E *DomesticatedSpecies*

CATEGORIES AND OBJECTS………..

- We say that two or more categories are**disjoint** if they have no members in common.
- we say that males and females constitute an **exhaustive decomposition** of the animals.
- A disjoint exhaustive decomposition is known as a **partition.** The following examples illustrate these three concepts:

Disjoint ({Animals, Vegetables))

ExhaustiveDecomposition({Americans, Canadians, Mexicans),NorthAmericans)

Partition ({Males, Females), Animals)

CATEGORIES AND OBJECTS………

- Categories can also be *defined* by providing necessary and sufficient conditions for membership. For example, a bachelor is an unmarried adult male:
- *x* E *Bachelors* -> *Unmarried(x) A x* E *Adults A x E Males* .

Physical composition :*PartOf* relation to say that one thing is part of another

The *PartOf* relation is transitive and reflexive; that is,

- *PartOf(x,* **Y)** A *PartOf (y , z)* + *partof (x, z)* .
- *Part Of (x , x)*

Categories of **composite objects** are often characterized by structural relations among parts

It is also useful to define composite objects with definite parts but no particular structure.

Bunch:

-defines the composite objects.

x E *s* -> *PartOf (x, BunchOf is))* .

ACTIONS, SITUATION AN D EVENTS

- actions are logical terms such as ***Forward*** and *Turn(Right).*
- **Situations** are logical terms consisting of the initial situation (usually called *So)* and all situations that are generated by applying an action to a situation. The function *Result(a, s)* (sometimes called *Do)* names the situation that results when action *a* is executed in situation *s.*
- **Fluents** are functions and predicates that vary from one situation to the next, such as the location of the agent.

Logical Agents

- An Agent can represent knowledge of its world, its goals and current situation.
- LA ha a collection of sentences in logic
- By using Logical sentences the agents decides what to do by inferring knowledge(conclusions).
- Conclusions is achieved by certain action or set of actions, is appropriate to achieve its goals.

Knowledge Base Agents

- The central component of a knowledge-based agent is its knowledge base, or KB.
- KB Contains a set of sentences in a formal Language.
- Sentences are expressed using a knowledge representaion language.
- Two generic functions

TELL- add new sentences (facts) to the KB

“Tell it what it needs to know”

ASK-query what is known from the KB

“Ask what to do next”

Knowledge Base Agents

- **function** _KB-AGENT(percept)ret_**urns** an *action*
- **static:** *KB,* a knowledge base

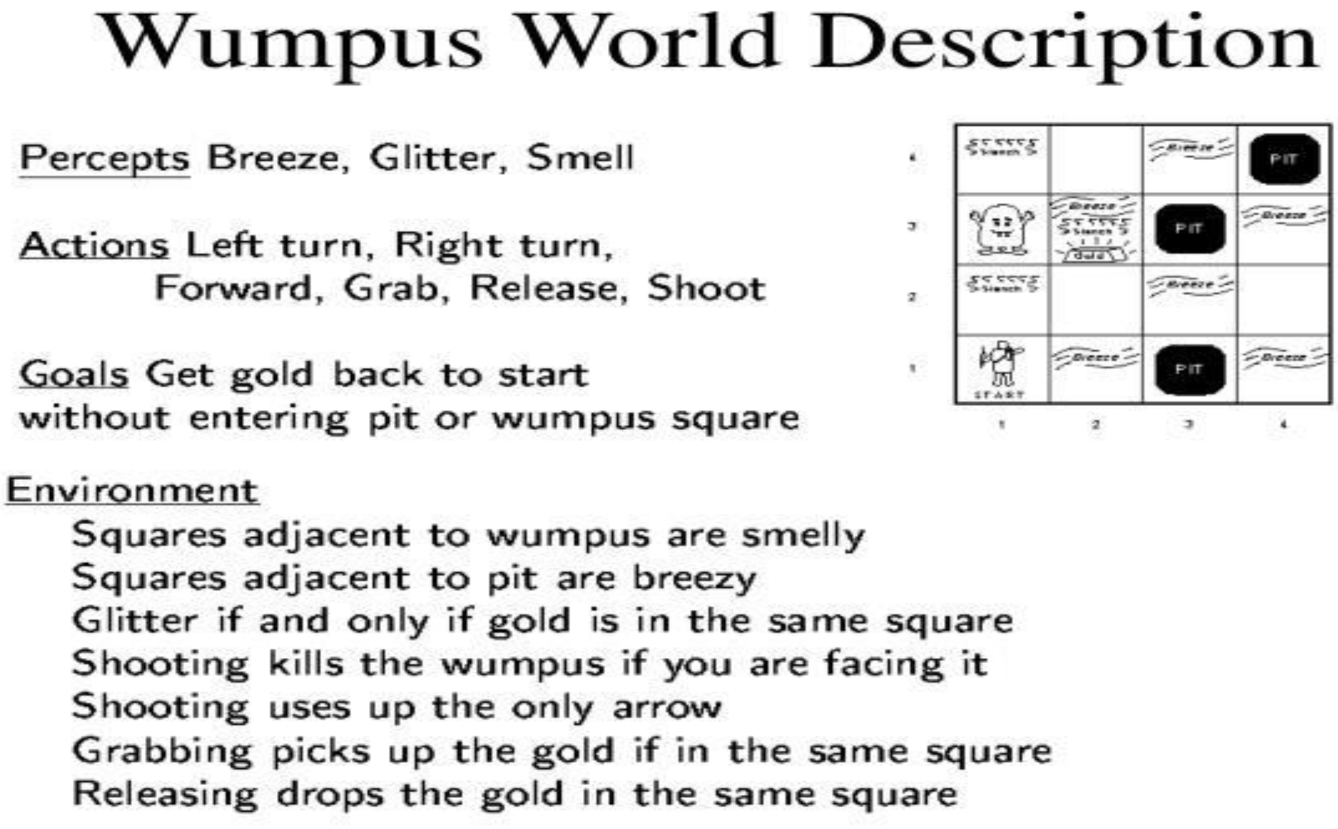
  t, a counter, initially 0, indicating time
- _TELL(KB_M*,* AKE-PERCEPT-SENTENCE(percept)
- *A*
- *ction <*- *AsK(KB,* MAKE-ACTION- QUERY(^))

  TELL(KB,**MAKE-ACTION-ENTENCE**_t)_**(action,t))**

  t->t+1
- **return** *action*

- THE WUMPUS WORLD

The **wumpus world** is a cave consisting of rooms connected by passageways.
Lurking somewhere in the cave is the wumpus, a beast that eats anyone who enters its room.
The wumpus can be shot by an agent, but the agent has only one arrow.
Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the wumpus, which is too big to fall in).
feature of living in this environment is the possibility of finding a heap of gold


4
3
2
1
Stench
Breeze
PIT
Breeze
Stench
Gold
PIT
Breeze
Stench
Breeze
START
Breeze
PIT
Breeze
1
2
3
4

Wumpus World Description

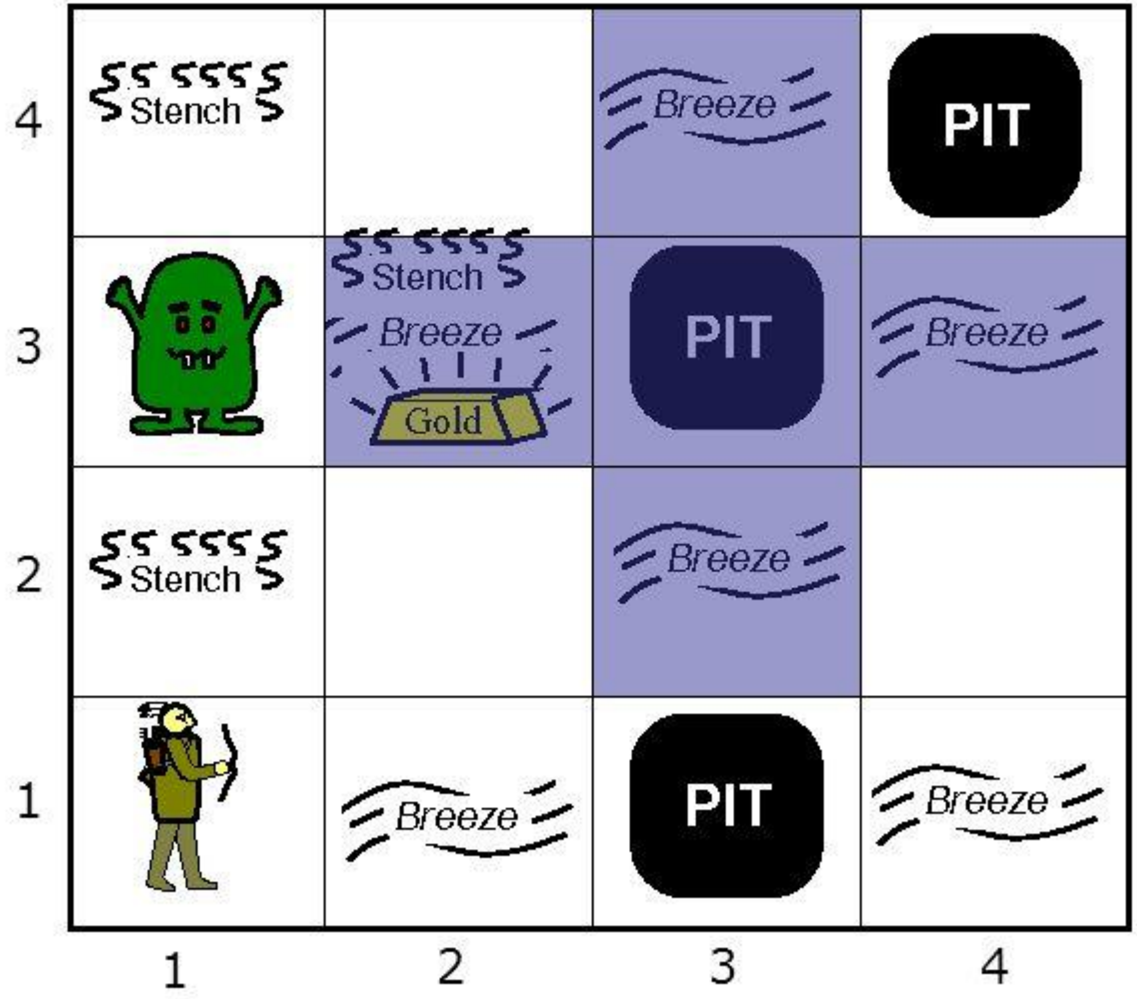
Percepts Breeze, Glitter, Smell

Actions Left turn, Right turn,
Forward, Grab, Release, Shoot

Goals Get gold back to start
without entering pit or wumpus square

Environment
Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter if and only if gold is in the same square
Shooting kills the wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up the gold if in the same square
Releasing drops the gold in the same square

The Wumpus World



Problem 2: Big, bottomless pits where you fall down. You can feel the breeze when you are near them.

The Wumpus World

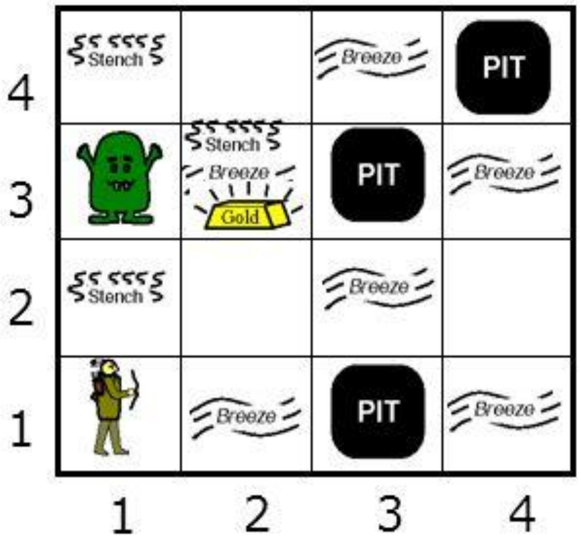
PEAS description

Performance measure:
+1000 for gold
-1000 for being eaten or falling down pit
-1 for each action
-10 for using the arrow

Environment:
4×4 grid of "rooms", each "room" can be empty, with gold, occupied by Wumpus, or with a pit.

Acuators:
Move forward, turn left 90°, turn right 90°
Grab, shoot

Sensors:
Olfactory – stench from Wumpus
Touch – breeze (pits) & hardness (wall)
Vision – see gold
Auditory – hear Wumpus scream when killed



$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} = \begin{pmatrix} forward \\ turn\,left \\ turn\,right \\ grab \\ shoot \end{pmatrix}, \alpha_i \in \{0,1\}$$

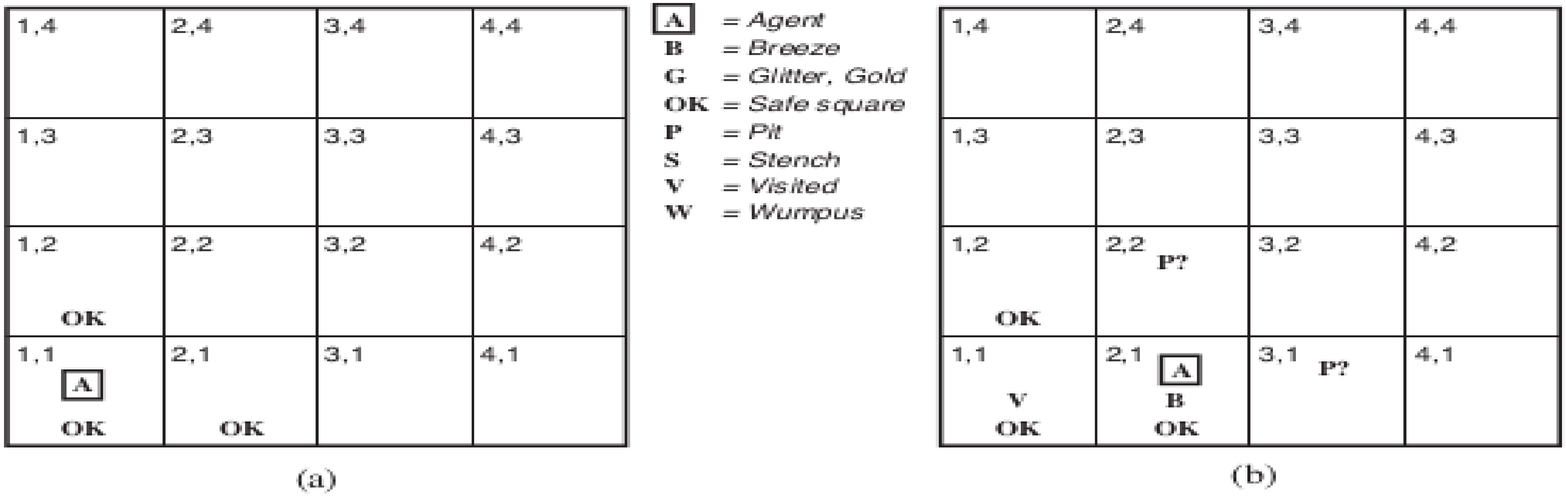


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept $[None, None, None, None, None]$. (b) After one move, with percept $[None, Breeze, None, None, None]$.

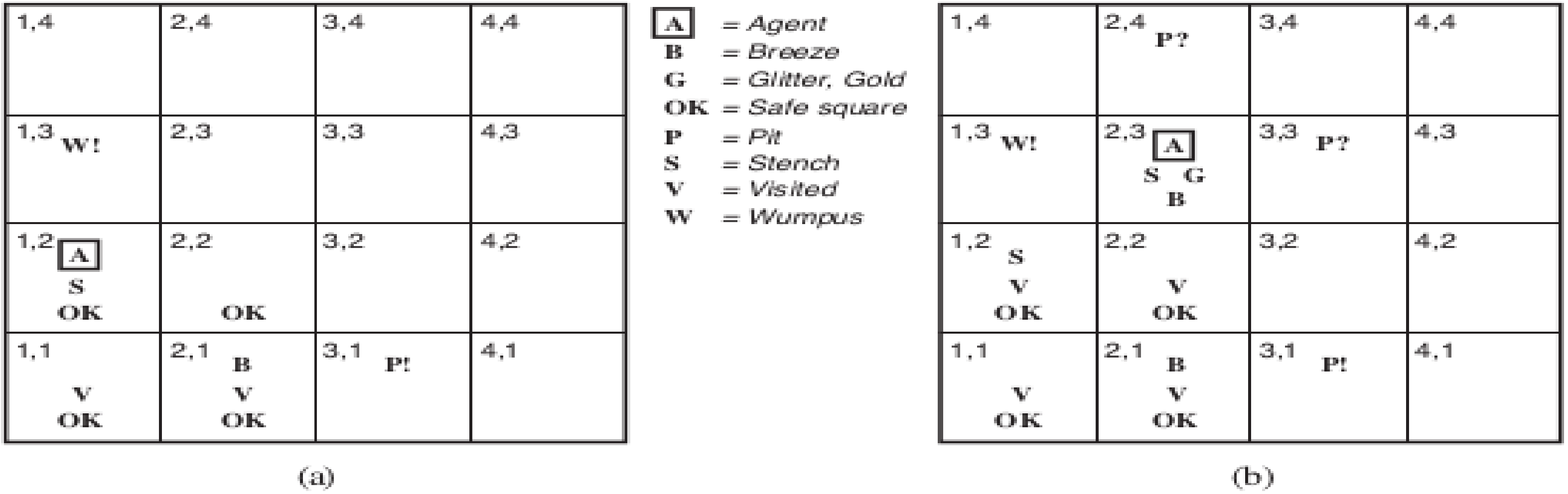


Figure 7.4 Two later stages in the progress of the agent. (a) After the third move, with percept $[Stench, None, None, None, None]$. (b) After the fifth move, with percept $[Stench, Breeze, Glitter, None, None]$.

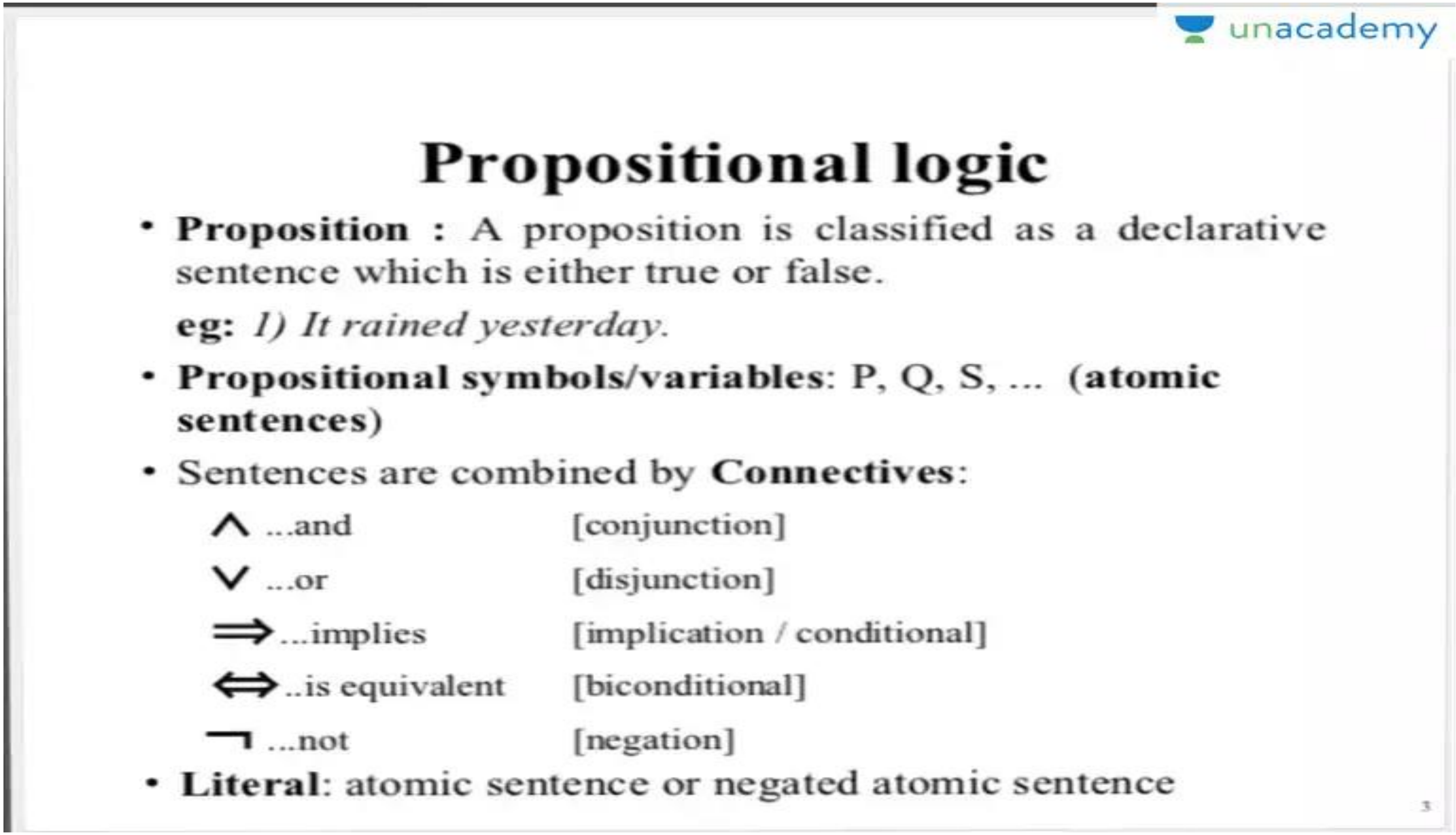
Propositional Logic

- Here the statements are made of propositions.
- The propositional logic is also called as boolean logic
- The sentence /statement is declarative which is either true or false but cannot be both.
- Question,opinion and comma are not allowed in this logic
- Eg.Students are studying in college(True proposition)
- 5+3=8(True proposition)
- What is your name?(not accepted)
- Some students are intelligent(false proposition)

Propositional Logic-syntax

- Syntax-defines the allowable sentences.
- The atomic sentences-the individual syntactic elements-consist od a single proposition symbol.
- Each such symbol stands for propositional that can be true or False.
- We wil use uppercase names for symbols:P,Q,R and so on.
- For eg
- $W_{1,3}$ stand for the proposition that the wumpus is in [1,3]

Complex sentences
logical operators



unacademy

Propositional logic

- **Proposition :** A proposition is classified as a declarative sentence which is either true or false.

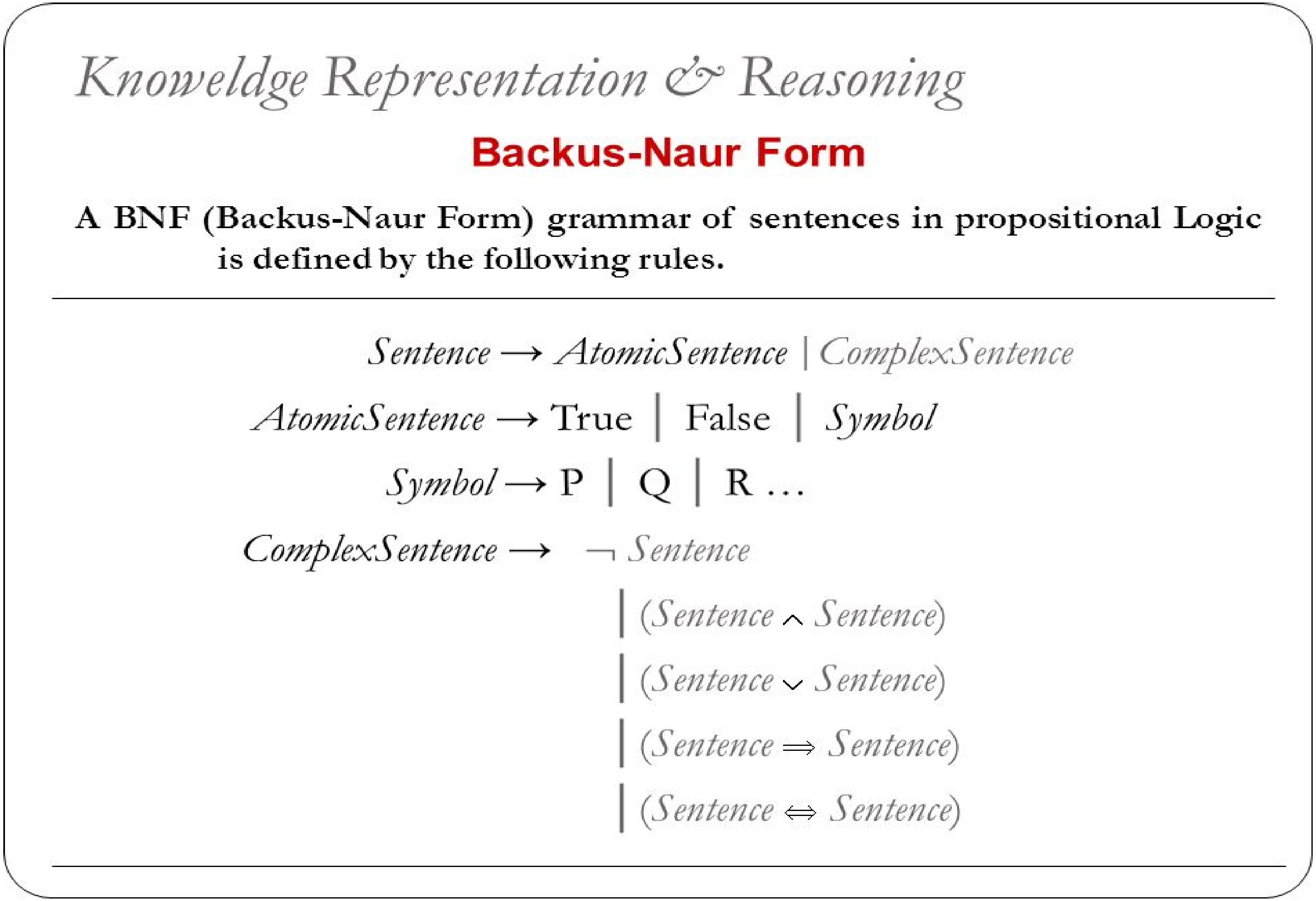
  eg: *1) It rained yesterday.*

- **Propositional symbols/variables**: P, Q, S, ... (**atomic sentences)**
- Sentences are combined by **Connectives**:

$\wedge$ ...and	[conjunction]
$\vee$ ...or	[disjunction]
$\Rightarrow$...implies	[implication / conditional]
$\Leftrightarrow$..is equivalent	[biconditional]
$\neg$ ...not	[negation]

- **Literal**: atomic sentence or negated atomic sentence

A Formal Grammar of Propositional Logic



Knoweldge Representation & Reasoning

Backus-Naur Form

A BNF (Backus-Naur Form) grammar of sentences in propositional Logic is defined by the following rules.

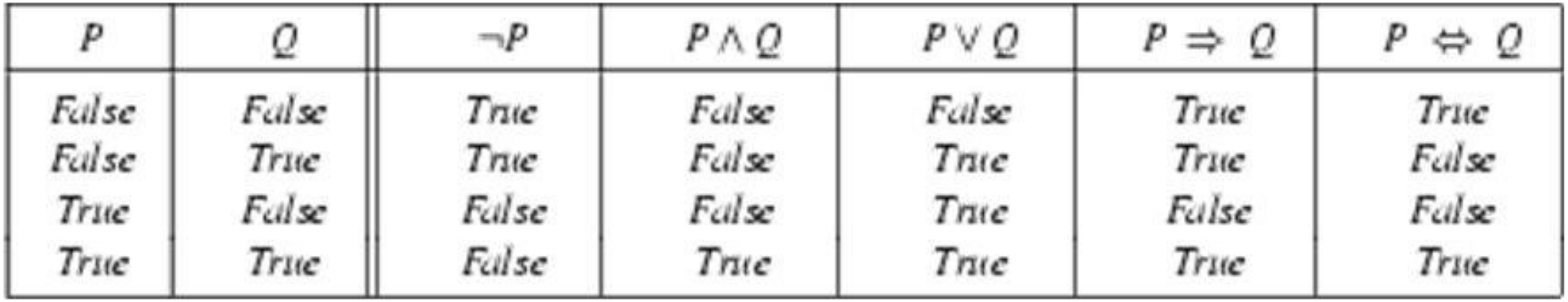
$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathrm{True} \mid \mathrm{False} \mid \mathit{Symbol} \\
\mathit{Symbol} &\rightarrow \mathrm{P} \mid \mathrm{Q} \mid \mathrm{R} \ldots \\
\mathit{ComplexSentence} &\rightarrow \neg\, \mathit{Sentence} \\
&\quad \mid (\mathit{Sentence} \wedge \mathit{Sentence}) \\
&\quad \mid (\mathit{Sentence} \vee \mathit{Sentence}) \\
&\quad \mid (\mathit{Sentence} \Rightarrow \mathit{Sentence}) \\
&\quad \mid (\mathit{Sentence} \Leftrightarrow \mathit{Sentence})
\end{aligned}
$$

Propositional Logic-semantics

- The semantics define the rules for determining the truth of a sentence w.r.to a particular model.
- In propositional logic model simply fixes the truth value true or false-for every propositional symbol.
- m1=(P1,2=false,P2,2=false,P3=true)
- The semantics for this logic must specify how to compute the truth value of any sentence given a model.
- This is done recursively.
- All sentences are constructed form atomic and the five connectives.

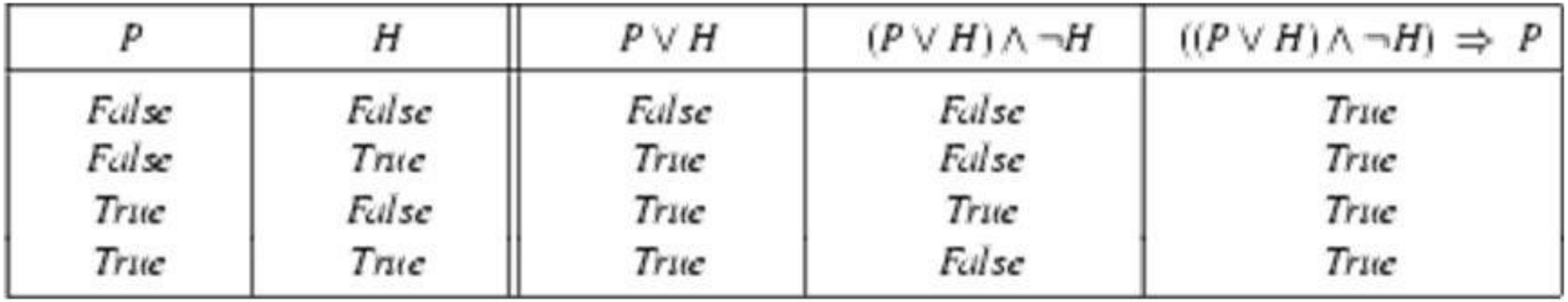
Truth tables II

The five logical connectives:



P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

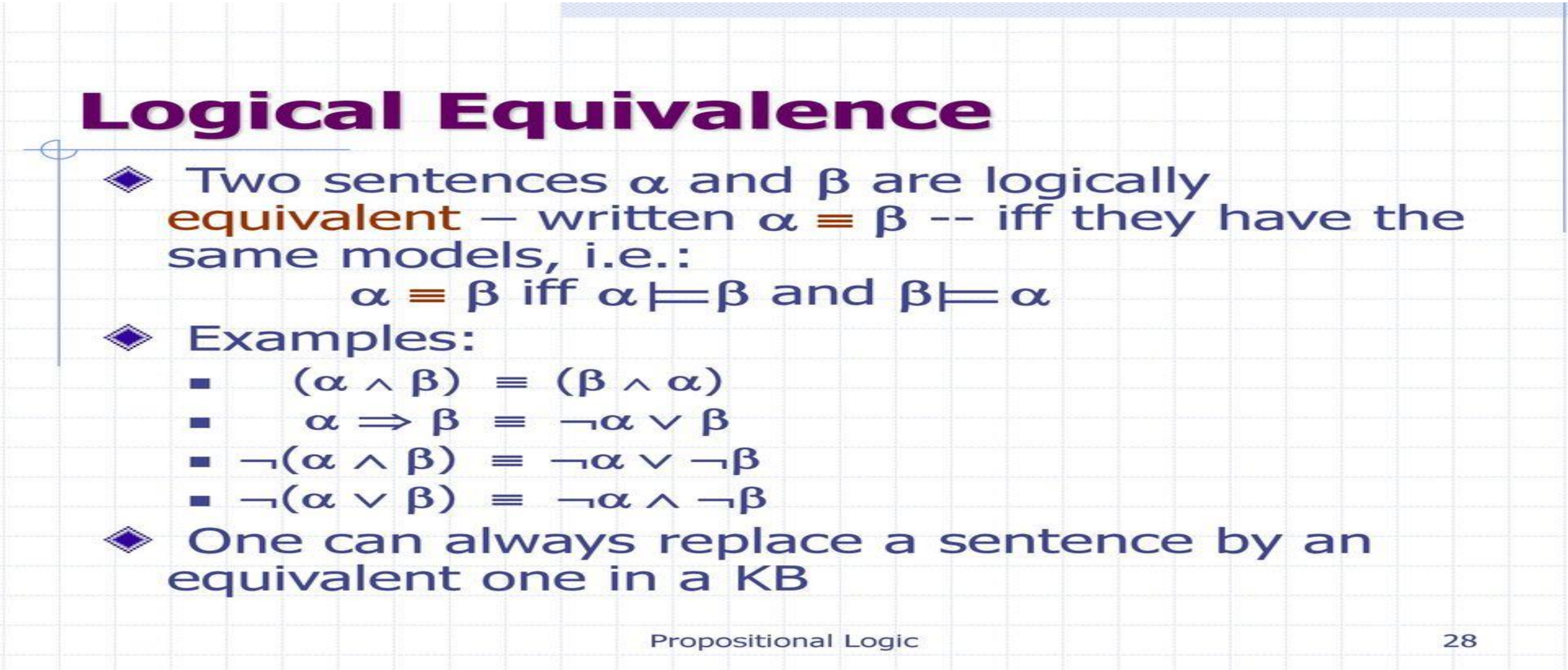
A complex sentence:



P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

Inferences ……….

- **Equivalence, validity, and satisfiability**



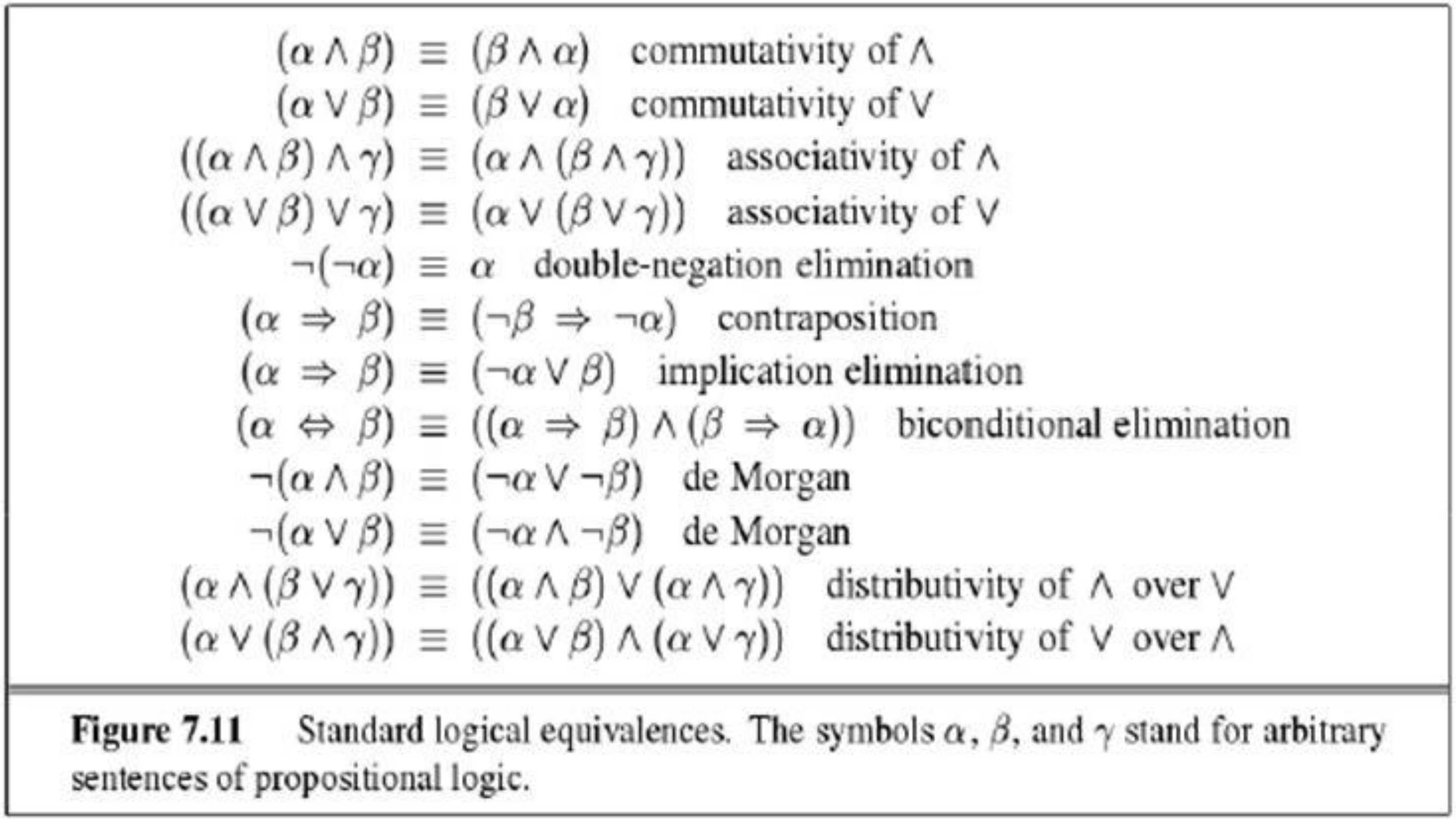
Logical Equivalence

- Two sentences α and β are logically equivalent – written $\alpha \equiv \beta$ -- iff they have the same models, i.e.:

$$\alpha \equiv \beta \text{ iff } \alpha \models \beta \text{ and } \beta \models \alpha$$

- Examples:
  - $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$
  - $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
  - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
  - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- One can always replace a sentence by an equivalent one in a KB

Standard Logical Equivalences



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

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Reasoning pattern in propositional logic

- This section covers standard patterns of inference that can be applied to derive chains of conclusions that lead to the desired goal. These patterns of inference are called **inference rules.**

REASONING PATTERNS IN PROPOSITIONAL LOGIC

- Patterns of inference are called **inference rules**. The best known rule is called **Modus Ponens**

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

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

And-Elimination, which says that ,from a conjunction any of the conjucts can be inferred

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

CONJUNCTIVE NORMAL FORM(CNF)

- A sentence expressed as a conjunction of disjunctions of literals is said to be in conjunctive normal form or CNF. Steps of CNF:

1. Eliminate $\Leftrightarrow$, replacing $(\alpha \Leftrightarrow \beta)$ with $((\alpha \Rightarrow \beta) \wedge ((\beta \Rightarrow \alpha)$
2. Eliminate $\Rightarrow$,replacing $(\alpha \Rightarrow \beta)$ with $(\neg \alpha \vee \beta)$
3. $\neg(\neg a) \equiv a$ a double – negation elimination

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

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

First Order logic/Predicate Logic

- Its like a natural language has well defined syntax and semantics.
- It assumes the world contains
- Object:people,houses,numbers,colors,baseball games, wars……….
- Relations:red,round,prime,brother of, bigger than………
- Functions: father of best friend, one more than, plus……

Properties of FOL

- It has the ability to represent facts about some or all of the objects and relations in the universe.
- Represent law and rules extracted from the real world.
- Useful language for maths, philosophy and AI
- Represent facts in realistic manner rather than just true /false.

Propositional Logic vs. Predicate Calculus

- Propositional Logic
  - The world consists of propositions (sentences) which can be true or false.

- Predicate Calculus (First Order Logic)
  - The world consists of objects, functions and relations between the objects.

Atomic sentences

- Atomic sentences=predicate(term1........termn)
- Term=function(term1,,,,,,,,,,termn)or constant or variable
- Example:

Brother(Richard, John).

Complex sentences

- It is made from atomic sentences using five Logical connectives
- S1 and S2 are two atomic statements then (BCF)

Truth in FOL

- Sentences are true with respect to a model and an interpretation.
- Model contains objects(domain elements) and relations among them.
- Interpretation specifies
  - Constant->objects
  - Predicate->Relations
  - Function->Functional relations

A Model for First Order Logic

E.g. *Sibling(KingJohn,Richard)* $\Rightarrow$ *Sibling(Richard,KingJohn)*

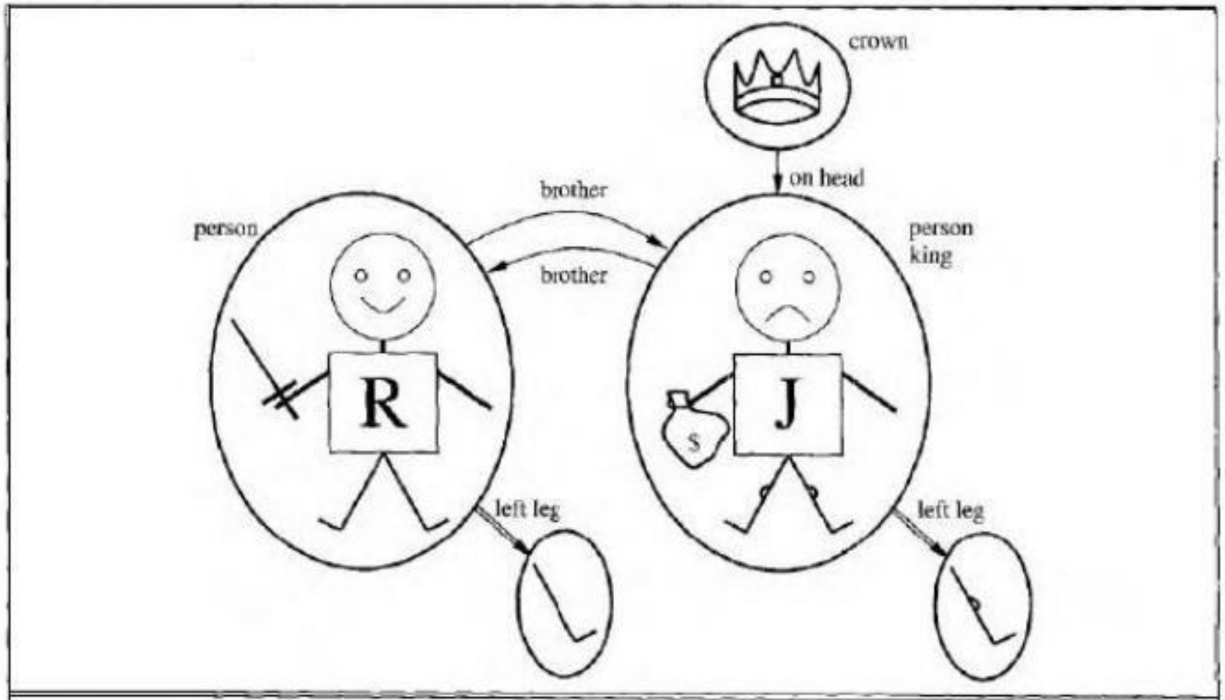
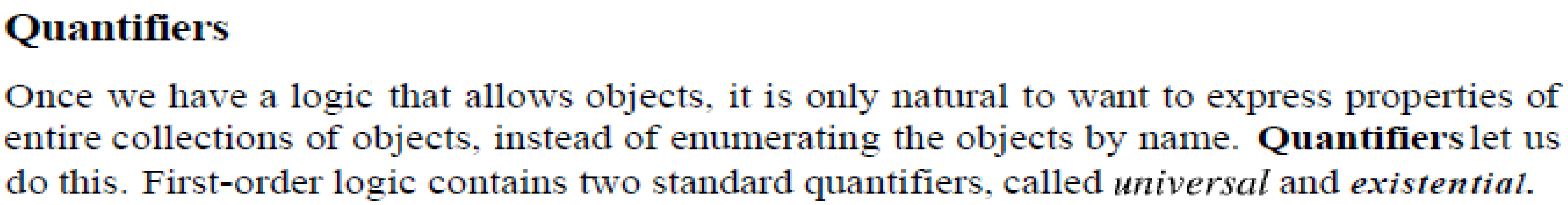


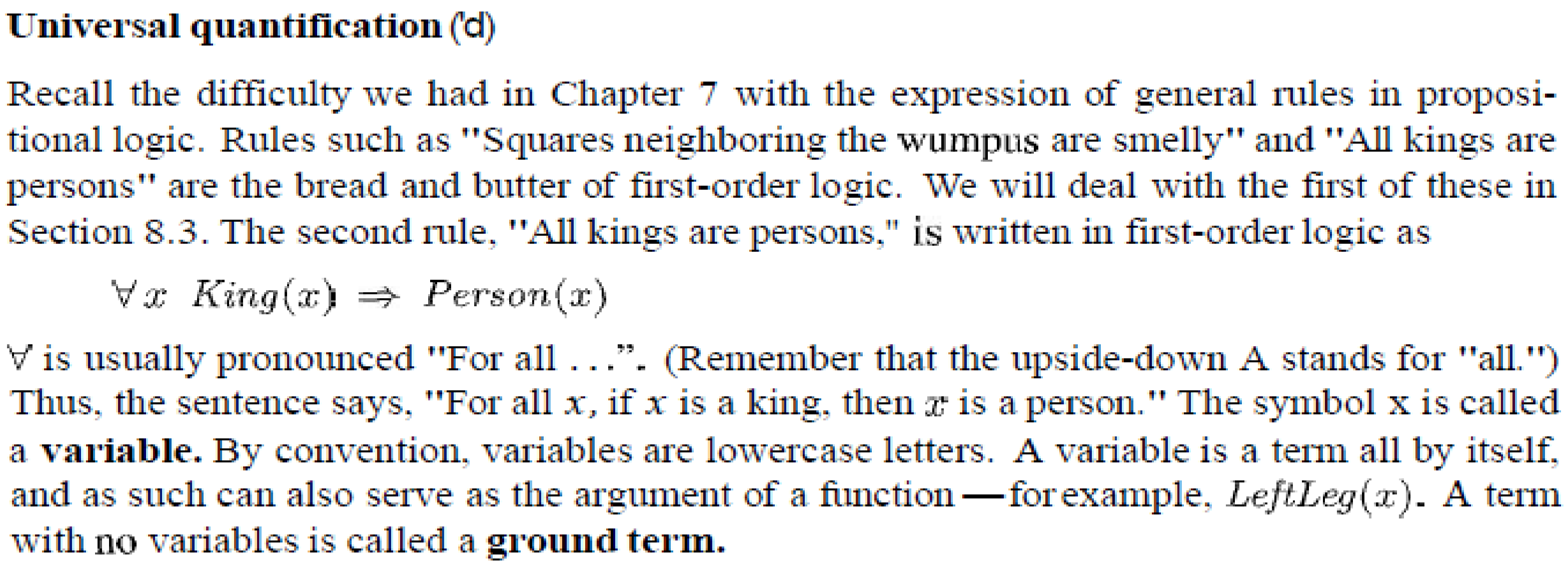
Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

SEMANTICS OF FIRST ORDER LOGIC



Quantifiers

Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. **Quantifiers** let us do this. First-order logic contains two standard quantifiers, called *universal* and *existential*.



Universal quantification ('d)

Recall the difficulty we had in Chapter 7 with the expression of general rules in propositional logic. Rules such as "Squares neighboring the wumpus are smelly" and "All kings are persons" are the bread and butter of first-order logic. We will deal with the first of these in Section 8.3. The second rule, "All kings are persons," is written in first-order logic as

$$\forall x \;\; King(x) \Rightarrow Person(x)$$

$\forall$ is usually pronounced "For all ...". (Remember that the upside-down A stands for "all.") Thus, the sentence says, "For all x, if x is a king, then x is a person." The symbol x is called a **variable.** By convention, variables are lowercase letters. A variable is a term all by itself, and as such can also serve as the argument of a function — for example, $LeftLeg(x)$. A term with no variables is called a **ground term.**

Connections between $\forall$ and $\exists$

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:

$$\forall x \ \neg Likes(x, Parsnips) \quad \text{is equivalent to} \quad \neg\exists x \ Likes(x, Parsnips) \,.$$

We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

$$\forall x \ \ Likes(x , IceCream) \quad \text{is equivalent to} \quad \neg\exists x \ \neg Likes(x, IceCream) \,.$$

FORWARD CHAINING

Inference Engine

- Its is the component of the intelligent system in artificial intelligence which applies logical rules to the Knowledge Base to infer information from known facts.
- The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes,which are:
- Forward chaining
- Backward Chaining

Horn clause and Definite Clause

- Horn clause and definite Clause are the forms of sentences which enables knowledge base to use a more restricted and efficient inference algorithm.
- Logical inference algorithms use forward and backward chaining approaches which require KB in the form of the first-order definite clause.
- Definite clause: A clause which is a disjunction of literals with exactly one positive literal is known as definite clause or strict horn clause
- Horn clause: A clause which is a disjunction of literals with at most one positive literal is known as horn clause .Hence all definite clauses are horn clauses.
- Example(negation p V negation q V k).It has only one positive literal

FORWARD CHAINING

- It is also called as forward deduction or forward reasoning method when using an inference engine.
- The idea is simple: start with the atomic sentences in the knowledge base and apply Modus Ponens in the forward direction, adding new atomic sentences, to extract more data until goal is reached.
- It is a bottom up approach..from bottom to top
- It is a process of making a conclusion based on known facts or data by starting from initial state and reaches the goal state.

- ✓ This algm start with known facts
- ✓ It triggers the entire rules whose premises are satisfied.
- ✓ Adding their conclusions to the known facts.
- ✓ The above process repeats until the query is answered or no new facts are called.

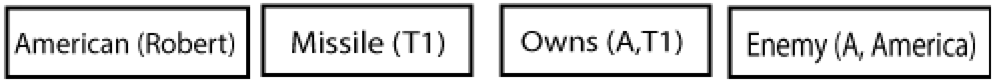
- Example:
- **"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."**
- Prove that **"Robert is criminal."**
- To solve the above problem, first, 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.

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 Owns(A, p) ∧ Missile(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 Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) ....(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) ....(6)
- Country A is an enemy of America.
  Enemy (A, America) ....(7)
- Robert is American
  American(Robert). ....(8)

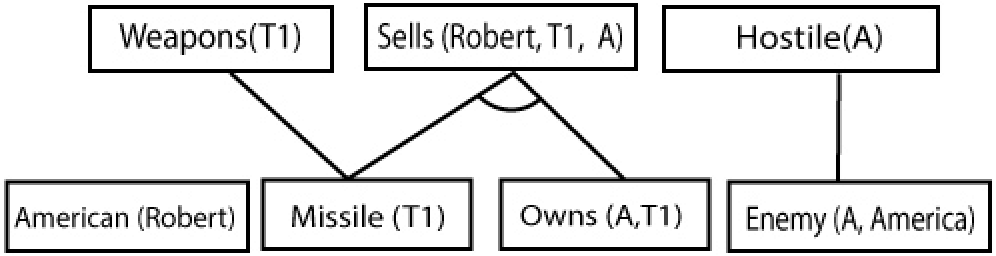
Forward chaining proof:

- **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.



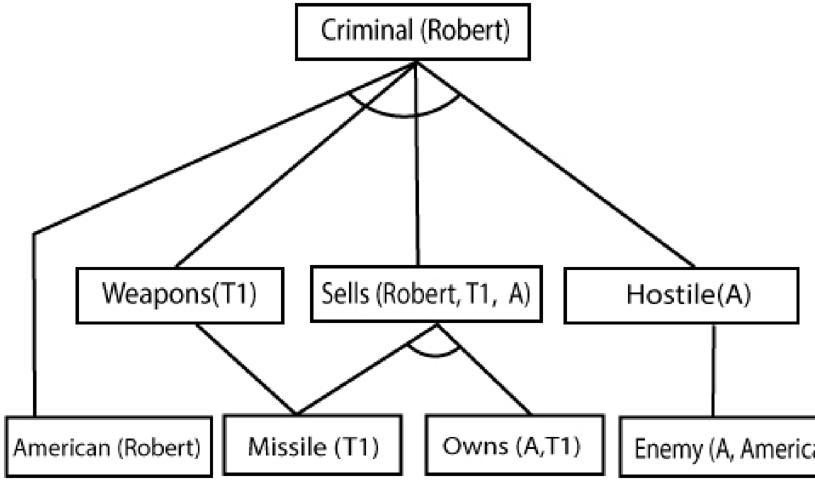
American (Robert) Missile (T1) Owns (A,T1) Enemy (A, America)

- **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).
- Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).
-



- **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

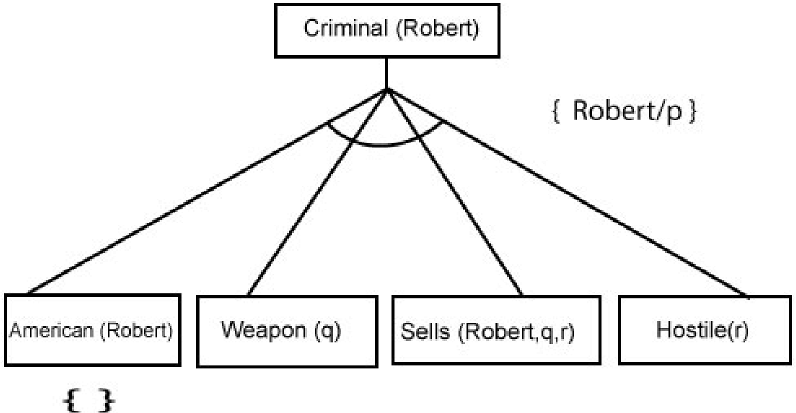
- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine.
- A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.
- It is known as a top-down approach.
- Backward-chaining 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.

Backward-Chaining proof:

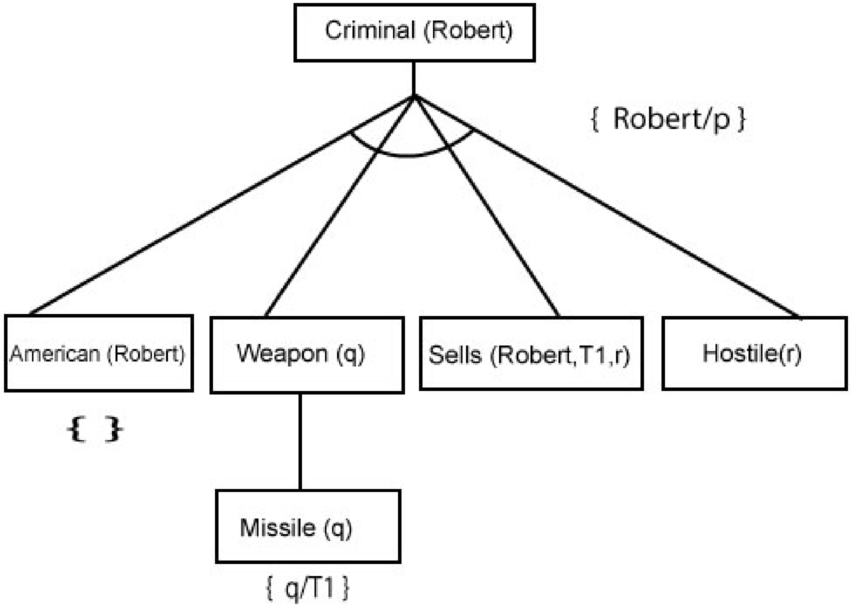
- In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.
- **Step-1:**
- At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is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

Criminal (Robert)

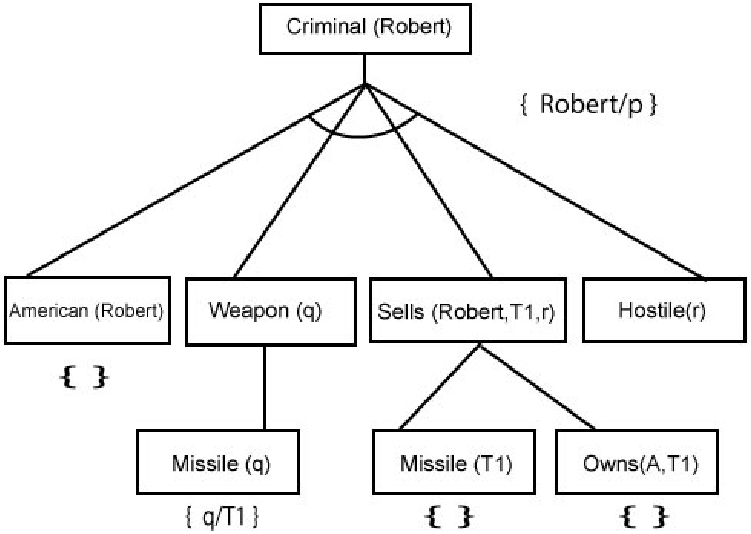
- **Step-2:**
- At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.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



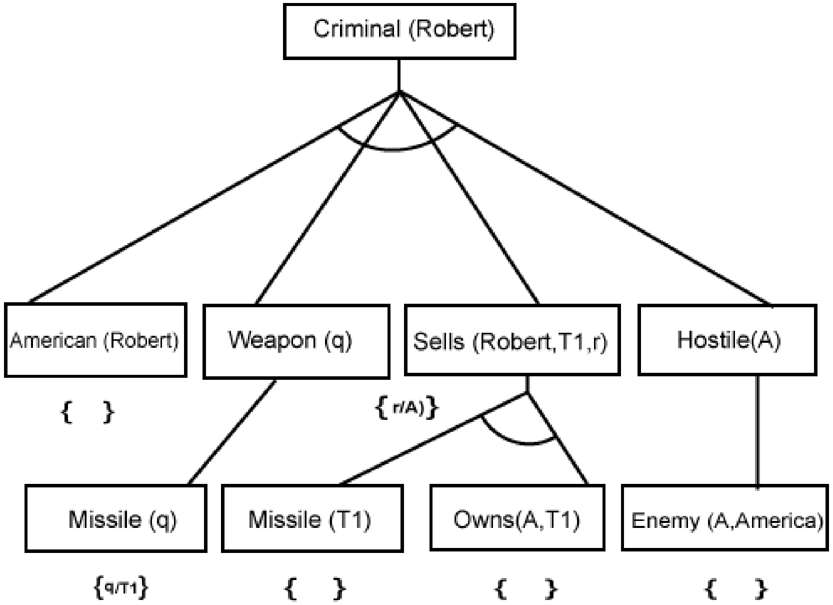
- **Step-3:**t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



- **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.
-



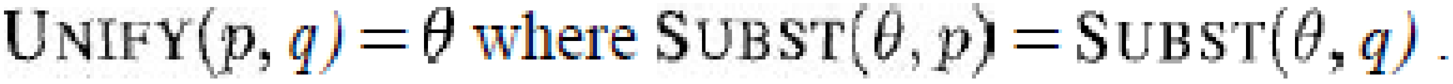
- **Step-5:**
- At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



UNIFICATION

- It is an algorithm for determining the substitutions needed to make two FOL expressions match
- To apply the rule of inference, an inference system must be able to determine when two expressions match, here the unification algorithm used.
- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences.

UNIFICATION



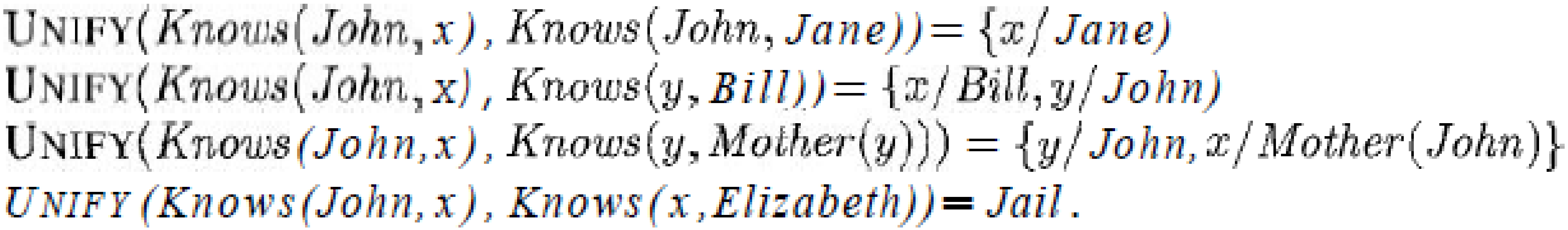
$$\textsc{Unify}(p, q) = \theta \text{ where } \textsc{Subst}(\theta, p) = \textsc{Subst}(\theta, q) \text{ .}$$

- FIND the MGU(Most General Unifier)for

Unify{king(x),king(john)}

Let α1=king(x) , α2=king(john),

Substitution Ө={john/x} is a unifier for those atoms and applying this substitution ,and both expressions will be identical.



$$\textsc{Unify}(Knows(John, x), Knows(John, Jane)) = \{x/Jane)$$

$$\textsc{Unify}(Knows(John, x), Knows(y, Bill)) = \{x/Bill, y/John)$$

$$\textsc{Unify}(Knows(John, x), Knows(y, Mother(y))) = \{y/John, x/Mother(John)\}$$

$$\textsc{Unify}\ (Knows(John, x), Knows(x, Elizabeth)) = Jail \text{ .}$$

Resolution in FOL

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements.
- Unification is a key concept in proofs by resolutions.
- Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

RESOLUTION………

- **Clause**: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.
- **Conjunctive Normal Form**: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Steps for Resolution

- Conversion of facts into first-order logic.
- Convert FOL statements into CNF
- Negate the statement which needs to prove (proof by contradiction)
- Draw resolution graph (unification).

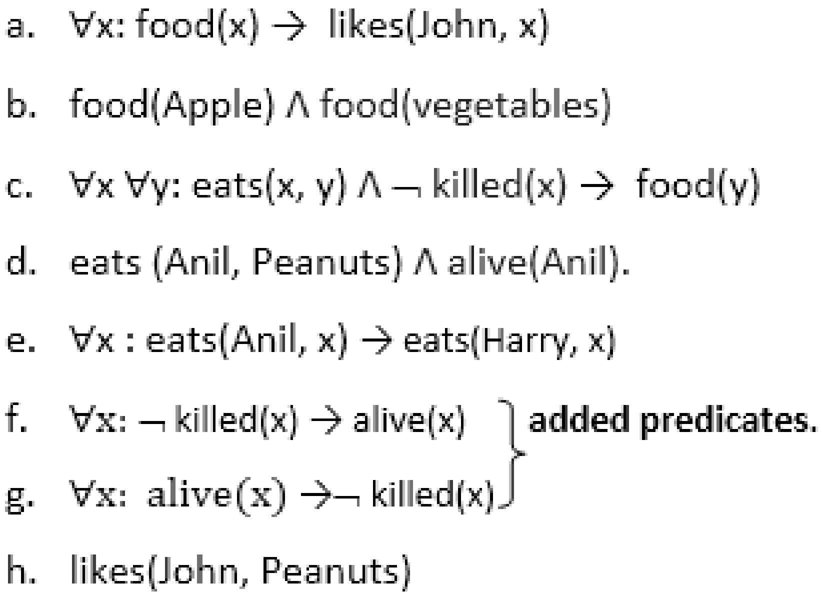
Example:
RESOLUTION

- **John likes all kind of food.**
- **Apple and vegetable are food**
- **Anything anyone eats and not killed is food.**
- **Anil eats peanuts and still alive**
- **Harry eats everything that Anil eats.**

Prove by resolution that:

- **John likes peanuts.**

- **Step-1: Conversion of Facts into FOL**
- In the first step we will convert all the given statements into its first order logic.



a. $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$

b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$

c. $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$

d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$.

e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

f. $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$ **added predicates.**

g. $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$

h. $\text{likes}(\text{John}, \text{Peanuts})$

- **Step-2: Conversion of FOL into CNF**
- In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.
- Eliminate all implication (→) and rewrite
- ∀x ¬ food(x) V likes(John, x)
- food(Apple) Λ food(vegetables)
- ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
- eats (Anil, Peanuts) Λ alive(Anil)
- ∀x ¬ eats(Anil, x) V eats(Harry, x)
- ∀x¬ [¬ killed(x)] V alive(x)
- ∀x ¬ alive(x) V ¬ killed(x)
- likes(John, Peanuts).

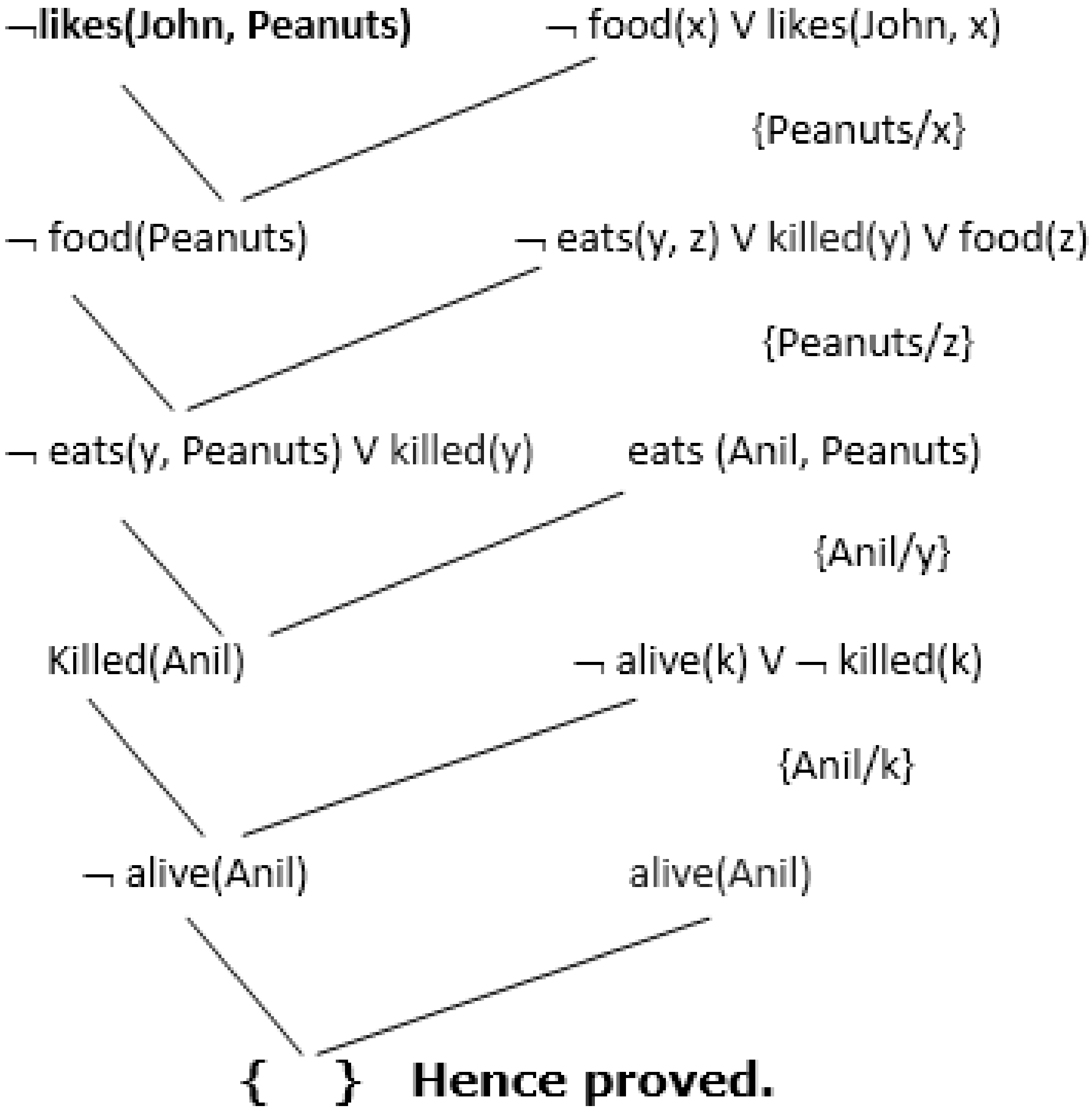
- **Move negation (¬)inwards and rewrite**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀x ¬ eats(Anil, x) V eats(Harry, x)
  - ∀x ¬killed(x)] V alive(x)
  - ∀x ¬ alive(x) V ¬ killed(x)
  - likes(John, Peanuts).

- **Rename variables or standardize variables**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀w¬ eats(Anil, w) V eats(Harry, w)
  - ∀g ¬killed(g)] V alive(g)
  - ∀k ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.
  - ¬ food(x) V likes(John, x)
  - food(Apple)
  - food(vegetables)
  - ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts)
  - alive(Anil)
  - ¬ eats(Anil, w) V eats(Harry, w)
  - killed(g) V alive(g)
  - ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).

- **Step-3: Negate the statement to be proved**
- In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)
- **Step-4: Draw Resolution graph:**
- Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



¬likes(John, Peanuts) ¬ food(x) V likes(John, x)

{Peanuts/x}

¬ food(Peanuts) ¬ eats(y, z) V killed(y) V food(z)

{Peanuts/z}

¬ eats(y, Peanuts) V killed(y) eats (Anil, Peanuts)

{Anil/y}

Killed(Anil) ¬ alive(k) V ¬ killed(k)

{Anil/k}

¬ alive(Anil) alive(Anil)

{ } Hence proved.