CST401 ARTIFICIAL INTELLIGENCE

MODULE 4

Course Outcomes: After the completion of the course the student will be able to

CO#	CO
CO1	Explain the fundamental concepts of intelligent systems and their architecture. **(Cognitive Knowledge Level: Understanding)**
CO2	Illustrate uninformed and informed search techniques for problem solving in intelligent systems. **(Cognitive Knowledge Level: Understanding)**
CO3	Solve Constraint Satisfaction Problems using search techniques. **(Cognitive Knowledge Level: Apply)**
CO4	Represent AI domain knowledge using logic systems and use inference techniques for reasoning in intelligent systems. **(Cognitive Knowledge Level: Apply)**
CO5	Illustrate different types of learning techniques used in intelligent systems **(Cognitive Knowledge Level: Understand)**

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	✓											
CO2	✓	✓										✓
CO3	✓	✓	✓	✓								✓
CO4	✓	✓	✓	✓								✓
CO5	✓	✓			✓							✓

Abstract POs defined by National Board of Accreditation			
PO#	**Broad PO**	**PO#**	**Broad PO**
PO1	Engineering Knowledge	PO7	Environment and Sustainability
PO2	Problem Analysis	PO8	Ethics
PO3	Design/Development of solutions	PO9	Individual and team work
PO4	Conduct investigations of complex problems	PO10	Communication
PO5	Modern tool usage	PO11	Project Management and Finance
PO6	The Engineer and Society	PO12	Life long learning

****

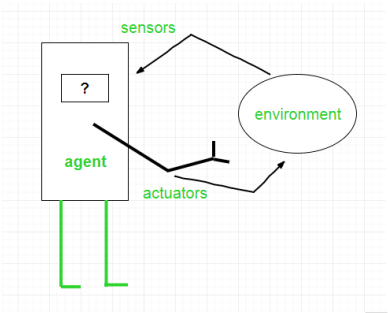
SYLLABUS- MODULE 5 (Knowledge Representation and Reasoning)

Logical Agents – Knowledge based agents, Logic, Propositional Logic, Propositional Theorem proving, Agents based on Propositional Logic. First Order Predicate Logic – Syntax and Semantics of First Order Logic, Using First Order Logic, Knowledge representation in First Order Logic. Inference in First Order Logic – Propositional Vs First Order inference, Unification and Lifting, Forward chaining, Backward chaining, Resolution.

****

Logical Agents

The idea is that an agent can represent knowledge of its world, its goals and the current situation by sentences in logic and decide what to do by inferring that a certain action or course of action is appropriate to achieve its goals.

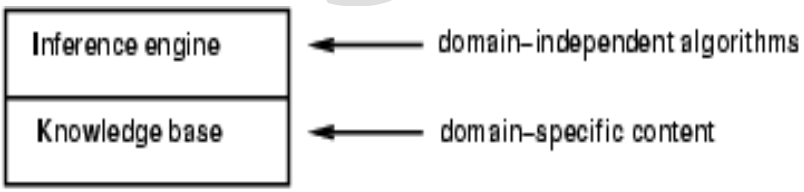


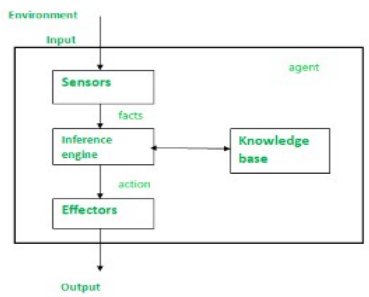
Knowledge-Based Agents

- Central component of a Knowledge-Based Agent is a Knowledge-Base
- It is a set of sentences in a formal language
- Sentences are expressed using a knowledge representation language
- Two generic functions:

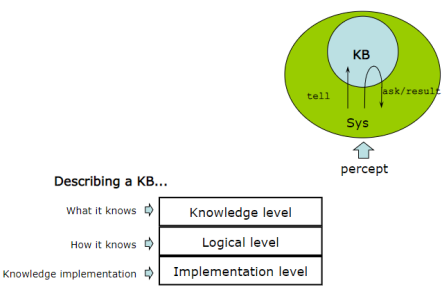
  – TELL - add new sentences (facts) to the KB • "Tell it what it needs to know"

  – ASK - query what is known from the KB • "Ask what to do next"





Knowledge Based Agents



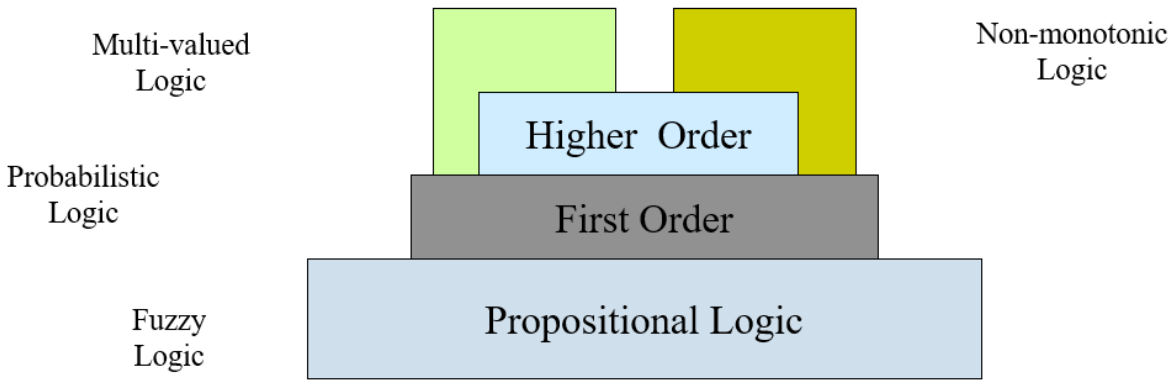
PEAS of Knowledge Based Agent

- **Performance Measure:** Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precepts.
- **Environment**: Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion.
- **Actuator**: An actuator is a part of the agent that delivers the output of action to the environment.
- **Sensor**: Sensors are the receptive parts of an agent that takes in the input for the agent.

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Logic



Logics are formal languages for representing information such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences; define truth of each sentence with respect to each possible world

E.g., the language of arithmetic

x+2 ≥ y is a sentence

x2+y > {} is not a sentence

x+2 ≥ y is true iff the number x+2 is no less than the number y

x+2 ≥ y is true in a world where x = 7, y = 1

Model- “possible world, mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence

Possible worlds- real environments that the agent might or might not be in

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

Entailment

Entailment means a sentence follows logically from another : to mean that the sentence α entails the sentence β

α |= β if and only if, in every model in which α is true, β is also true

α |= β if and only if M(α) ⊆ M(β) .

Knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true

Eg. the KB containing “the Phillies won” and “the Reds won” entails “Either the Phillies won or the Reds won”

Eg. x+y = 4 entails 4 = x+y

Entailment is a relationship between sentences (i. e. , syntax) that is based on semantics

Inference and Entailment

Inference is a procedure that allows new sentences to be derived from a knowledge base. Understanding inference and entailment: think of Set of all consequences of a KB as a haystack, α as the needle, Entailment is like the needle being in the haystack and Inference is like finding it.

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.”

M is a model of a sentence α if α is true in M

M(α) is the set of all models of α

- **Entailment**
  - *KB* entails a sentence *s*: $KB \models s$
  - *KB* derives (proves) a sentence *s*: $KB \vdash s$
- **Soundness and Completeness**
  - Soundness: $KB \vdash s \Rightarrow KB \models s$, for all *s*
  - Completeness : $KB \models s \Rightarrow KB \vdash s$, for all *s*

Validity: true under all interpretations

Satisfiability: true under some interpretation, i.e., there is at least one model

18 (a) Prove or find a counter example to the following assertion in propositional logic: **(6)**

If $\alpha \models (\beta \wedge \gamma)$ then $\alpha \models \beta$ and $\alpha \models \gamma$.

$\alpha \models (\beta \wedge \gamma) \equiv \alpha \rightarrow (\beta \wedge \gamma) \equiv \neg\alpha \vee (\beta \wedge \gamma)$

$\alpha \models \beta$ and $\alpha \models \gamma \equiv (\neg\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma)$

$(\alpha \models \beta) \wedge (\alpha \models \gamma) \equiv (\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma)$

If $\alpha \models (\beta \wedge \gamma)$ then $\alpha \models \beta$ and $\alpha \models \gamma$, can be written as

$\alpha \models (\beta \wedge \gamma) = (\alpha \models \beta) \wedge (\alpha \models \gamma)$, since $p \models q \equiv p \rightarrow q$

$\alpha \rightarrow (\beta \wedge \gamma) = (\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma)$, since $p \rightarrow q \equiv \neg p \vee q$

$\neg\alpha \vee (\beta \wedge \gamma) = (\neg\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma)$, Apply distributive law

$(\neg\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma) = (\neg\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma)$

Hence LHS = RHS

Sound and Truth Preserving Inference

An inference algorithm that derives only entailed sentences is called sound or truth preserving. Soundness is a highly desirable property. An unsound inference procedure essentially makes things up as it goes along—it announces the discovery of nonexistent needles. It is easy to see that model checking, when it is applicable is a sound procedure. The property of completeness is also desirable: an inference algorithm is complete if it can derive any sentence that is entailed

A reasoning process whose conclusions are guaranteed to be true in any world in which the premises are true; in particular, 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

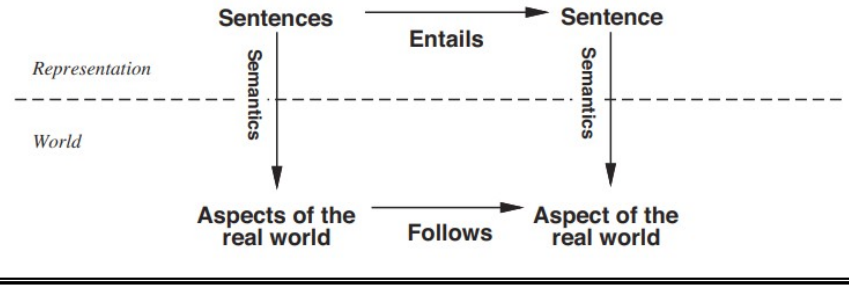


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

Propositional Logic: A Very Simplest Logic

Syntax of PL: defines the allowable sentences or propositions.

Definition (Proposition): A proposition is a declarative statement (True or False).

A fact, like "the Sun is hot." The Sun cannot be both hot and not hot at the same time. This declarative statement could also be referred to as a proposition.

Atomic proposition: single proposition symbol. Each symbol is a proposition. Notation: upper case letters and may contain subscripts.

Compound proposition: constructed from atomic propositions using parentheses and logical connectives.

Examples of atomic propositions:

- 2+2=4 is a true proposition
- W1,3 is a proposition. It is true if there is a Wumpus in [1,3]
- "If there is a stench in [1,2] then there is a Wumpus in [1,3]" is a proposition
- "How are you?" or "Hello!" are not propositions. In general, statement that are questions, commands, or opinions are not propositions.

Examples of compound/complex propositions

Let p, p1, and p2 be propositions

- Negation ¬p is also a proposition.
- A literal is either an atomic proposition or its negation. E.g., W1,3 is a positive literal, and ¬W1,3 is a negative literal.
- Conjunction p1 ∧ p2. E.g., W1,3 ∧ P3,1
- Disjunction p1 ∨ p2 E.g., W1,3 ∨ P3,1

- Implication p1 → p2. E.g., W1,3 ∧ P3,1 → ¬W2,2
- If and only if p1 ↔ p2. E.g., W1,3 ↔ ¬W2,2

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \mid [\textit{Sentence}] \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Truth Table

The semantics define the rules to determine the truth of a sentence.

- Semantics can be specified by truth tables.
- Boolean values domain: T,F , n-tuple: (x1, x2, ..., xn)
- Operator on n-tuples : g(x1 = v1, x2 = v2, ..., xn = vn)
- A truth table defines an operator g on n- tuples by specifying a Boolean value for each tuple.
- Number of rows in a truth table? R = 2n

Negation

p	$\neg p$
T	F
F	T

Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

If and only if

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Exclusive OR

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Precedence of operators

1. Expressions in parentheses are processed (inside to outside)
2. Negation

3. AND
4. OR
5. Implication
6. Biconditional
7. Left to right

Use parentheses whenever you have any doubt!

Logical Equivalence

Two propositions p and q are logically equivalent if and only if the columns in the truth table giving their truth values agree. We write this as p ⇔ q or p ≡ q.

p→q is equivalent to ¬pvq

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties

Commutativity:

p ∧ q = q ∧ p

p ∨ q = q ∨ p

Associativity:

p ∧ q) ∧ r = p ∧ (q ∧ r)

(p ∨ q) ∨ r = p ∨ (q ∨ r)

Identity element:

p ∧ T rue = p

p ∨ T rue = T rue

¬(¬p) = p

p ∧ p = p p ∨ p = p

Distributivity:

p ∧ (q ∨ r) = (p ∧ q) ∨ (p ∧ r)

p ∨ (q ∧ r) = (p ∨ q) ∧ (p ∨ r)

p ∧ (¬p) = False and p ∨ (¬p) = T rue

DeMorgan's laws:

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

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

Tautology and contradiction

Tautology is a proposition which is always true. Contradiction is a proposition which is always false. Contingency is a proposition which is neither a tautology or a contradiction

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Contrapositive and Inverse

Given an implication p → q

- The converse is: q → p
- The contrapositive is: ¬q → ¬p
- The inverse is: ¬p → ¬q

Conditional statement: "If it is raining, then the grass is wet." The first step is to identify the hypothesis and conclusion statements. Hypothesis, p: it is raining. Conclusion, q: grass is wet

- Converse statement would be: "If the grass is wet, then it is raining."
- Inverse statement would be: "If it is NOT raining, then the grass is NOT wet."
- Contrapositive statement would be: "If the grass is NOT wet, then it is NOT raining."

Inference (Modus Ponens)

- Assume you are given the following two statements:
  - "you are in this class"
  - "if you are in this class, you will get a grade"

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$$

- Let p = "you are in this class"
- Let q = "you will get a grade"
- By Modus Ponens, you can conclude that you will get a grade

$$\frac{p \qquad p \rightarrow q}{q}$$

For example, if (WumpusAhead ∧WumpusAlive) ⇒ Shoot and (WumpusAhead ∧ WumpusAlive) are given, then Shoot can be inferred

- Consider $(p \wedge (p \rightarrow q)) \rightarrow q$

p	q	p→q	p∧(p→q))	(p∧(p→q)) → q
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

$$\begin{array}{l} p \\ \underline{p \rightarrow q} \\ \therefore q \end{array}$$

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Modus Tollens

Modus tollens is a valid argument form in propositional calculus in which p and q are propositions. If p implies q, and q is false, then p is false. Also known as an indirect proof or a proof by contrapositive.

$$\frac{p \Rightarrow q, \neg q}{\therefore \neg p}$$

For example, if being the king implies having a crown, not having a crown implies not being the king.

- Assume that we know: ¬q and p → q
  - Recall that p → q = ¬q → ¬p
- Thus, we know ¬q and ¬q → ¬p
- We can conclude ¬p

$$\begin{array}{l} \neg q \\ \underline{p \rightarrow q} \\ \therefore \neg p \end{array}$$

- Assume you are given the following two statements:
  - "you will not get a grade"
  - "if you are in this class, you will get a grade"

$$\begin{array}{l} \neg q \\ \underline{p \rightarrow q} \\ \therefore \neg p \end{array}$$

- Let p = "you are in this class"
- Let q = "you will get a grade"
- By Modus Tollens, you can conclude that you are not in this class

And Elimination, Unit Resolution

And-Elimination : (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

From a conjunction, any of the conjuncts can be inferred. For example, from (WumpusAhead ∧ WumpusAlive), WumpusAlive can be inferred.

Unit Resolution : (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \ \neg \beta}{\alpha}$$

DNF and CNF

DNF: Disjunctive Normal Form, OR of ANDs (terms)

e.g. (p∧¬q) ∨ (¬p∧¬r)

CNF: Conjunctive Normal Form

"Every sentence of propositional logic is logically equivalent to a conjunction of clauses". AND of ORs (clauses)

e.g. (p∨¬q) ∧ (¬p∨¬r)

Procedure for converting to CNF

Convert the sentence B1,1 ⇔ (P1,2 ∨ P2,1) into CNF

1. Eliminate ⇔, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$
2. Eliminate ⇒, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) .$$
3. CNF requires ¬ to appear only in literals, so we "move ¬ inwards" by repeated application of the following equivalences from Figure 7.11:
$$\neg(\neg\alpha) \equiv \alpha \quad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$
In the example, we require just one application of the last rule:
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) .$$
4. Now we have a sentence containing nested ∧ and ∨ operators applied to literals. We apply the distributivity law from Figure 7.11, distributing ∨ over ∧ wherever possible.
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) .$$

Definite clauses

- A **definite clause** is of the form

$$h \leftarrow a_1 \wedge \ldots \wedge a_m.$$

where h is an atom, the **head** of the clause, and each a_i is an atom. It can be read "h if a_1 and ...and a_m".

If $m > 0$, the clause is called a **rule**, where $a_1 \wedge \ldots \wedge a_m$ is the **body** of the clause.

If $m = 0$ the arrow can be omitted and the clause is called an **atomic clause** or **fact**. The clause has an **empty body**.

- A **knowledge base** is a set of definite clauses.

Horn Clause

Horn clause = at most one +ve literal in clause

- positive / definite clause = exactly one +ve literal
  e.g. $[\neg p_1, \neg p_2, ..., \neg p_n, q]$
- negative clause = no +ve literals (also, referred to as integrity constraints)
  e.g. $[\neg p_1, \neg p_2, ..., \neg p_n]$ and also [ ]

Types of Horn Clauses :

- **Definite clause / Strict Horn clause –**
  It has exactly one positive literal.
- **Unit clause –**
  Definite clause with no negative literals.
- **Goal clause –**
  Horn clause without a positive literal.

$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}
$$

Figure 7.14 A grammar for conjunctive normal form, Horn clauses, and definite clauses. A clause such as $A \wedge B \Rightarrow C$ is still a definite clause when it is written as $\neg A \vee \neg B \vee C$, but only the former is considered the canonical form for definite clauses. One more class is the k-CNF sentence, which is a CNF sentence where each clause has at most k literals.

Prove, or find a counter example to, the following assertion:
If $\alpha \models \gamma$ or $\beta \models \gamma$ (or both) then $(\alpha \wedge \beta) \models \gamma$

Ans. $\alpha \models \gamma \equiv \alpha \rightarrow \gamma \equiv \neg\alpha \vee \gamma$

$\beta \models \gamma \equiv \beta \rightarrow \gamma \equiv \neg\beta \vee \gamma$

$$
\begin{aligned}
(\alpha \vdash \gamma) \vee (\beta \vdash \gamma) &\equiv (\neg\alpha \vee \gamma) \vee (\neg\beta \vee \gamma) \\
&\equiv (\neg\alpha \vee \neg\beta) \vee \gamma \\
&\equiv \neg(\alpha \vee \beta) \vee \gamma \\
&\equiv (\alpha \vee \beta) \rightarrow \gamma \\
&\equiv (\alpha \vee \beta) \vdash \gamma
\end{aligned}
$$

Hence Proved

AGENTS BASED ON PROPOSITIONAL LOGIC: Wumpus world – Knowledge Base

Atomic propositions

- Let $\mathbf{P_{i,j}}$ be true if there is a Pit in the room [i, j].
- Let $\mathbf{B_{i,j}}$ be true if agent perceives breeze in [i, j], (dead or alive).
- Let $\mathbf{W_{i,j}}$ be true if there is wumpus in the square[i, j].
- Let $\mathbf{S_{i,j}}$ be true if agent perceives stench in the square [i, j].
- Let $\mathbf{V_{i,j}}$ be true if that square[i, j] is visited.
- Let $\mathbf{G_{i,j}}$ be true if there is gold (and glitter) in the square [i, j].
- Let $\mathbf{OK_{i,j}}$ be true if the room is safe.

1,4	2,4 P?	3,4	4,4
1,3 W?	2,3 S G B	3,3	4,3
1,2	2,2 V P?	3,2	4,2
1,1 A ok	2,1 B V ok	3,1 P?	4,1

- Let $P_{i,j}$ be true if there is a pit in $[i,j]$
- Let $B_{i,j}$ be true if there is a breeze in $[i,j]$

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

- "A square is breezy if and only if there is an adjacent pit"

R_2: $B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$

R_3: $B_{2,1} \Leftrightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$

R_4: $\neg B_{1,1}$

R_5: $B_{2,1}$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(R1) $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

(R2) $\neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

(R3) $\neg S_{12} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

(R4) $S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

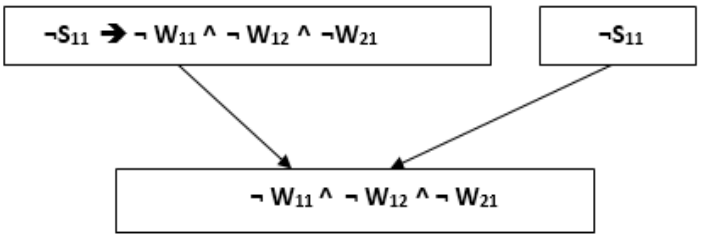
$\neg W_{11}$	$\neg S_{11}$	$\neg P_{11}$	$\neg B_{11}$	$\neg G_{11}$	V_{11}	OK_{11}
$\neg W_{12}$	----	$\neg P_{12}$	-----	----	$\neg V_{12}$	OK_{12}
$\neg W_{21}$	$\neg S_{21}$	$\neg P_{21}$	B_{21}	$\neg G_{21}$	V_{21}	OK_{21}

Room[1,1], room does not have wumpus(¬ W11), no stench (¬S11), no Pit(¬P11), no breeze(¬B11), no gold (¬G11), visited (V11), and the room is Safe(OK11).

Inference

Prove that Wumpus is in the room (1, 3) using propositional rules which have been derived for the Wumpus world and using inference rule.

- **Apply Modus Ponens with ¬S11 and R1:**



Apply And-Elimination Rule:

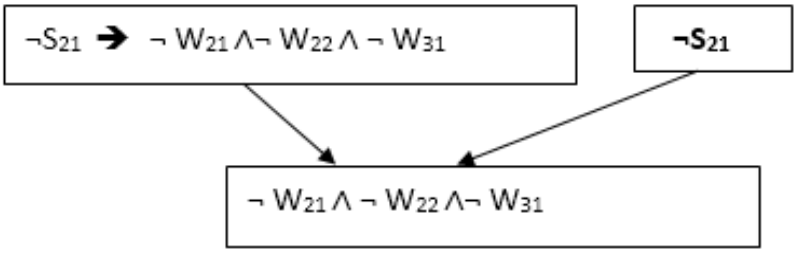
After applying And-elimination rule to ¬ W11 ∧ ¬ W12 ∧ ¬ W21, we will get three statements:

¬ W11, ¬ W12, and ¬W21.

- **Apply Modus Ponens to $\neg S_{21}$, and R2:**

Now we will apply Modus Ponens to $\neg S_{21}$ and R2 which is $\neg S_{21} \rightarrow \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$, which will give the Output as **$\neg W_{21} \wedge \neg W_{22} \wedge$**

$\neg W_{31}$

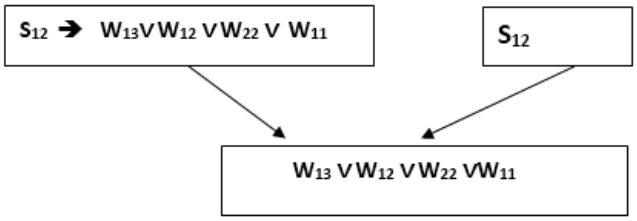


- **Apply And -Elimination rule:**

Now again apply And-elimination rule to **¬ W_{21} ∧ ¬ W_{22} ∧¬ W_{31}**, We will get three statements:
¬ W_{21}, ¬ W_{22}, and ¬ W_{31}.

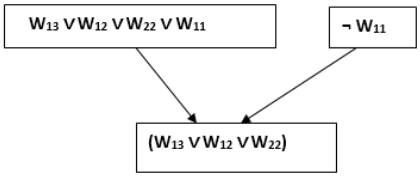
- **Apply Modus Pones to S_{12} and R4:**

Apply Modus Ponens to **S_{12}** and **R_4** which is **S_{12} → W_{13} ∨. W_{12} ∨. W_{22} ∨.W_{11}**, we will get the output as **W_{13}∨ W_{12} ∨ W_{22} ∨.W_{11}**.



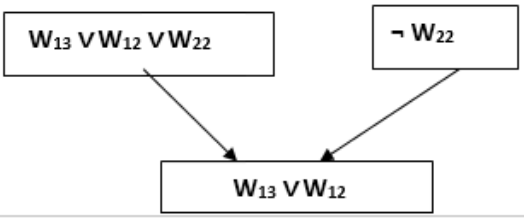
- **Apply Unit resolution on W_{13} ∨ W_{12} ∨ W_{22} ∨W_{11} and ¬ W_{11} :**

After applying Unit resolution formula on W_{13} ∨ W_{12} ∨ W_{22} ∨W_{11} and ¬ W_{11} we will get W_{13} ∨ W_{12} ∨ W_{22}.



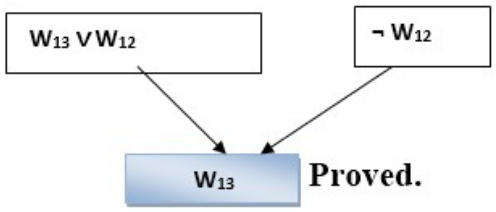
- **Apply Unit resolution on W_{13} ∨ W_{12} ∨ W_{22} and ¬ W_{22} :**

After applying Unit resolution on **W_{13} ∨ W_{12} ∨ W_{22}**, and **¬W_{22}**, we will get **W_{13} ∨ W_{12}** as output.



- **Apply Unit Resolution on W_{13} ∨ W_{12} and ¬ W_{12} :**

After Applying Unit resolution on **W_{13} ∨ W_{12} and ¬ W_{12}**, we will get **W_{13}** as an output, hence it is proved that the Wumpus is in the room [1, 3].



Problems on Propositional Logic

- Tom is hardworking

P: Hardworking(Tom)

- Tom is an intelligent student

Q:Intelligent(Tom)

- If Tom is hardworking and intelligent then Tom scores high marks

R: Hardworking(Tom) ∧Intelligent(Tom) →ScoresHighMarks(Tom)

What about other students that are hardworking and intelligent?

All students that are hardworking and intelligent score high marks

Let all students be represented by variable 'x'

Student(x) ˆ Hardworking(x) ˆIntelligent(x) →ScoresHighMarks(x)

Represent the following assertion in propositional logic:

"A person who is radical (R) is electable (E) if he/she is conservative (C), but otherwise is not electable."

$$R \Rightarrow (E \Leftrightarrow C)$$

Demerit of Propositional Logic

Propositional logic can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

"Some humans are intelligent", or

"Sachin likes cricket."

First-Order Logic (FOL)

It is an extension to propositional logic. FOL is sufficiently expressive to represent the natural language statements in a concise way. First-order logic is also known as Predicate logic or First-order predicate logic.

First-order logic (FOL) models the world in terms of

Objects, which are things with individual identities

Properties of objects that distinguish them from other objects

Relations that hold among sets of objects

Functions, which are a subset of relations where there is only one “value” for any given “input”

Examples:

Objects: Students, lectures...

Relations: Brother-of, bigger-than, outside..

Properties: blue, oval, even, large, ...

Functions: father-of, best-friend, second-half, one-more-than ...

FOL is also called as Predicate Logic. It is a generalization of Propositional Logic that allows us to express and infer arguments in infinite models, Eg,

Some birds can fly

All men are mortal

At least one student has course registered

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} & \rightarrow & (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
 & \mid & \neg\ \textit{Sentence} \\
 & \mid & \textit{Sentence} \wedge \textit{Sentence} \\
 & \mid & \textit{Sentence} \vee \textit{Sentence} \\
 & \mid & \textit{Sentence} \Rightarrow \textit{Sentence} \\
 & \mid & \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
 & \mid & \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
 & \mid & \textit{Constant} \\
 & \mid & \textit{Variable} \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{array}
$$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1060 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

Terms

A term is a logical expression that refers to an object. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object. For example, in English we might use the expression “King John’s left leg” rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use LeftLeg(John).

Consider a term f(t1,...,tn). The function symbol f refers to some function in the model (call it F); the argument terms refer to objects in the domain (call them d1,...,dn); and the term as a whole refers to the object that is the value of the function F applied to d1,...,dn.

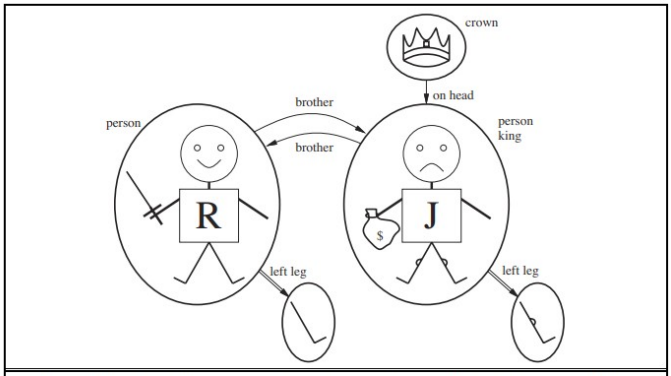


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

Atomic Sentences in FOL

Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms. Atomic sentences are represented as Predicate (term1, term2, ..., term n).

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

Complex sentences

Complex sentences are made by combining atomic sentences using connectives. We can use logical connectives to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the model of Figure 8.2 under our intended interpretation:

Brother (Richard, John) $\wedge$ Brother (John, Richard)

King(Richard) $\vee$ King(John)

$\neg$King(Richard) $\Rightarrow$ King(John)

Quantifiers

Quantifiers express properties of entire collections of objects. First-order logic contains two standard quantifiers, called

- universal and
- existential.

Universal quantification ($\forall$)

It is expression of general rules in propositional logic. The second rule, "All kings are persons," is written in first-order logic as

$\forall$ x King(x) $\Rightarrow$ Person(x) // "For all x, if x is a king, then x is a person."

$\forall$ is usually pronounced "For all ...".

The symbol x is called a variable. 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.

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

would be equivalent to asserting

Richard the Lionheart is a king $\wedge$ Richard the Lionheart is a person,
King John is a king $\wedge$ King John is a person,
Richard's left leg is a king $\wedge$ Richard's left leg is a person,

Existential quantification ($\exists$)

Universal quantification makes statements about every object. Similarly, we can make a statement about some object in the universe without naming it, by using an existential quantifier. To say, for example, that King John has a crown on his head, we write

$\exists$ x Crown(x) $\wedge$ OnHead(x, John)

$\exists$x is pronounced "There exi

sts an x such that ..." or "For some x...". The sentence $\exists$ x P says that P is true for at least one object x.

∃ x P is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element

∃ x Crown(x) ⇒ OnHead(x, John)

Nested quantifiers

We want to express more complex sentences using multiple quantifiers. For example, "Brothers are siblings" can be written as

∀ x ∀ y Brother (x, y) ⇒ Sibling(x, y) .

Consecutive quantifiers of the same type can be written as one quantifier with several variables. For example, to say that siblinghood is a symmetric relationship, we can write

∀ x, y Sibling(x, y) ⇔ Sibling(y, x) .

In other cases we will have mixtures. "Everybody loves somebody" means that for every person, there is someone that person loves:

∀ x ∃ y Loves(x, y) .

On the other hand, to say "There is someone who is loved by everyone," we write

∃ y ∀ x Loves(x, y) .

says that everyone has a particular property, namely, the property that they love someone. On the other hand, ∃ y (∀ x Loves(x, y)) says that someone in the world has a particular property, namely the property of being loved by everybody

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:

∀ x ¬Likes(x,Parsnips) is equivalent to ¬∃ x Likes(x,Parsnips) .

"Everyone likes ice cream" means that there is no one who does not like ice cream:

∀ x Likes(x,IceCream) is equivalent to ¬∃ x ¬Likes(x,IceCream)

De Morgan rules for quantified and unquantified sentences

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

Equality

We can use the equality symbol to signify that two terms refer to the same object.

For example, Father (John) = Henry

says that the object referred to by Father (John) and the object referred to by Henry are the same. Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.

To say that Richard has at least two brothers, we would write

$$\exists x, y \;\; Brother(x, Richard) \wedge Brother(y, Richard) \wedge \neg(x = y) .$$

FOL Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Examples of FOL

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:
∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y),**" **where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y),**" **where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

8.10 Consider a vocabulary with the following symbols:

$Occupation(p, o)$: Predicate. Person p has occupation o.
$Customer(p1, p2)$: Predicate. Person $p1$ is a customer of person $p2$.
$Boss(p1, p2)$: Predicate. Person $p1$ is a boss of person $p2$.
$Doctor$, $Surgeon$, $Lawyer$, $Actor$: Constants denoting occupations.
$Emily$, Joe: Constants denoting people.

Use these symbols to write the following assertions in first-order logic:

a. Emily is either a surgeon or a lawyer.
b. Joe is an actor, but he also holds another job.
c. All surgeons are doctors.
d. Joe does not have a lawyer (i.e., is not a customer of any lawyer).
e. Emily has a boss who is a lawyer.
f. There exists a lawyer all of whose customers are doctors.
g. Every surgeon has a lawyer.

a) Emily is either a surgeon or a lawyer.

Occupation(Emily, Surgeon) ∨ Occupation(Emily, Lawyer)

or

Occupation(Emily, Surgeon) ⇔ ¬Occupation(Emily, Lawyer)

b) Joe is an actor, but he holds another job.

Occupation(Joe, Actor) ∧ ∃ o [Occupation(Joe, o) ∧ ¬ (o = Actor)]

or

Occupation(Joe, Actor) ∧ [Occupation(Joe, Doctor) ∨ Occupation(Joe, Surgeon) ∨ Occupation(Joe, Lawyer)]

c) All surgeons are doctors.

$\forall$ p [Occupation(p, Surgeon) $\Rightarrow$ Occupation(p, Doctor)]

d) Joe does not have a lawyer (i.e., Joe is not a customer of any lawyer).

Joe does not have a lawyer (i.e., Joe is not a customer of any lawyer).

$\forall$ p [Occupation(p, Lawyer) $\Rightarrow$ $\neg$ Customer(Joe, p)]

Or

$\neg$ $\exists$ p [Occupation(p, Lawyer) $\wedge$ Customer(Joe, p)]

or

$\forall$ p [Customer(Joe, p) $\Rightarrow$ $\neg$ Occupation(p, Lawyer)]

e) Emily has a boss who is a lawyer.

$\exists$ p [Boss(p, Emily) $\wedge$ Occupation(p, Lawyer)]

f) There exists a lawyer all of whose clients are doctors (i.e., all of whose customers are doctors).

$\exists$ p1 $\forall$ p2 Occupation(p1, Lawyer) $\wedge$ [Customer(p2, p1) $\Rightarrow$ Occupation(p2, Doctor)]

Or

$\exists$ p1 Occupation(p1, Lawyer) $\wedge$ [$\forall$ p2 Customer(p2, p1) $\Rightarrow$ Occupation(p2, Doctor)]

g) Every surgeon has a lawyer (i.e., every surgeon is a customer of a lawyer).

$\forall$ p1 $\exists$ p2 Occupation(p1, Surgeon) $\Rightarrow$ [Customer(p1, p2) $\wedge$ Occupation(p2, Lawyer)]

Or

$\forall$ p1 Occupation(p1, Surgeon) $\Rightarrow$ [$\exists$ p2 Customer(p1, p2) $\wedge$ Occupation(p2, Lawyer)]

8.19 Assuming predicates Parent(p, q) and Female(p) and constants Joan and Kevin, with the obvious meanings, express each of the following sentences in first-order logic. (You may use the abbreviation ∃1 to mean "there exists exactly one.")

a. Joan has a daughter (possibly more than one, and possibly sons as well).

b. Joan has exactly one daughter (but may have sons as well).

c. Joan has exactly one child, a daughter.

d. Joan and Kevin have exactly one child together.

e. Joan has at least one child with Kevin, and no children with anyone else.

a) ∃x: Parent(Joan, x) ^ Female(x)
b) ∃1x: Parent(Joan, x) ^ Female(x)
c) ∃1x: Parent(Joan, x) -> Female(x)
d) ∃1x: Parent(Joan, x) ^ Parent(Kevin, x)
e) ∃1x: Parent(Joan, x) -> Parent(Kevin, x)

Free and Bound Variable

There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x is the bound variables.

Unification

The process of finding a substitution for predicate parameters is called unification. 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. We need to know: that 2 literals can be matched. The substitution is that makes the literals identical. There is a simple algorithm called the unification algorithm that does this.

Let's say there are two different expressions, P(x, y), and P(a, f(z)).

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x,y).....(i)

P(a, f(z))..... (ii)

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 will be: [a/x, f(z)/y].

The substitution variables are called Most General Unifier or MGU.

Conditions for Unification:

Following are some basic 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.

The Unification 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:

   a. Substitute t_i / v_i in the existing substitutions

   b. Add t_i /v_i to the substitution setlist.

   c. 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 Example

- P(x) and P(y): substitution = (x/y) *"substitute x for y"*
- P(x,x) and P(y,z): (z/y)(y/x) *y for x, then z for y*
- P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
- P(f(x)) and P(x): can't do it!
- P(x) ∨ Q(Jane) and P(Bill) ∨ Q(y):

  (Bill/x, Jane/y)

Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, **Unification failed.**

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))

S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}

SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}

SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}

SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**

And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

3. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))

S_0 => {p (X, X), p (Z, f(Z))}

SUBST θ= {X/Z}

S1 => {p (Z, Z), p (Z, f(Z))}

SUBST θ= {f(Z) / Z}, **Unification Failed.**

4. Find the MGU of UNIFY(prime (11), prime(y))

Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}

S_0 => {prime(11) , prime(y)}

SUBST θ= {11/y}

S_1 => {prime(11) , prime(11)} , **Successfully unified.**

Unifier: {11/y}.

5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)

S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}

SUBST θ= {f(b)/x}

S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}

S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, **Successfully Unified.**

Unifier: [a/a, f(b)/x, b/y].

6. UNIFY(knows(Richard, x), knows(Richard, John))

Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)

S_0 => { knows(Richard, x); knows(Richard, John)}

SUBST θ= {John/x}

S_1 => { knows(Richard, John); knows(Richard, John)}, **Successfully Unified.**

Unifier: {John/x}.

Qn. For each pair of atomic sentences, find the most general unifier if it exists:

a) P (A, B, B), P (x, y, z).

b) Q(y, G(A, B)), Q(G(x, x), y).

Ans. P(A,B,B), P(x,y,z)

x/A	P(A,B,B)	P(A,y,z)
y/B	P(A,B,B)	P(A,B,z)
z/B	P(A,B,B)	P(A,B,B)

MGU σ = {x/A, y/B, z/B}

Q(y, G(A, B)), Q(G(x, x), y)

Replacing A or B with x or y won't make any difference here so unification fails

MGU= NULL

(b) For each pair of atomic sentences, give the most general unifier if it exists: **(8)**
Older (Father (y), y), Older (Father (x), John).

x/John

Older(Father(y),y), Older(Father(John), John)

y/John

Older(Father(John), John), Older(Father(John), John)

MGU = {x/John, y/John}

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

Example

[Animal (g(x) V Loves (f(x), x)] **and** **[¬ Loves(a, b) V ¬Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for resolution

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

Example

a. **John likes all kind of food.**
b. **Apple and vegetable are food**
c. **Anything anyone eats and not killed is food.**
d. **Anil eats peanuts and still alive**
e. **Harry eats everything that Anil eats.**
   Prove by resolution that:
f. **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. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } added predicates.
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, 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**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) ∧ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) ∧ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) ∧ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

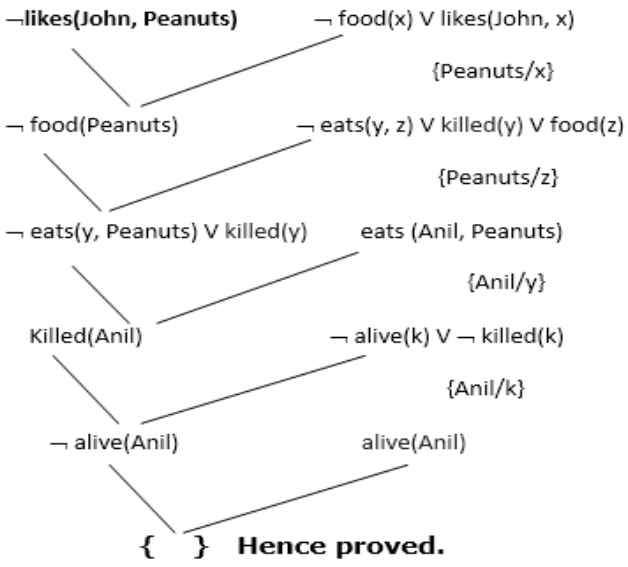
- **Move negation (¬)inwards and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) ∧ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  d. eats (Anil, Peanuts) ∧ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x ¬killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

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

a. ¬ food(x) V likes(John, x)
b. food(Apple)
c. food(vegetables)
d. ¬ eats(y, z) V killed(y) V food(z)
e. eats (Anil, Peanuts)
f. alive(Anil)
g. ¬ eats(Anil, w) V eats(Harry, w)
h. killed(g) V alive(g)
i. ¬ alive(k) V ¬ killed(k)
j. 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)



17 (a) Convert the following sentences into first order logic: **(6)**

Everyone who loves all animals is loved by someone.

Anyone who kills an animal is loved by no one.

Jack loves all animals.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

(b) Give a resolution proof to answer the question "Did Curiosity kill the cat? " **(8)**

Resolution step 1: Conversion of facts into FOL

1. Everyone who loves all animals is loved by someone

$$\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow \exists y \text{ loves}(y,x)$$

2. Anyone who kills an animal is loved by no one

$$\forall x [\exists z \text{ Animal}(z) \wedge \text{kills}(x,z)] \Rightarrow \forall y \neg \text{Loves}(y,x)$$

3. Jack loves all animals

$$\forall x \text{ Animal}(x) \Rightarrow \text{Loves}(\text{jack}, x)$$

4. Either Jack or Curiosity killed the cat, who is named Tuna

$$\text{kills}(\text{jack}, \text{tuna}) \vee \text{kills}(\text{curiosity}, \text{tuna})$$

5. The cat is named Tuna

$$\text{cat}(\text{tuna})$$

6. Cats are animals.

$$\forall x \text{ cat}(x) \Rightarrow \text{Animal}(x)$$

Added predicates (background knowledge)

7. Curiosity kill the cat

$$\text{kills}(\text{curiosity}, \text{tuna})$$

Step 2: Conversion of FOL into CNF

1. $\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow \exists y \text{ Loves}(y,x)$

scoping : y and y are different variables

Eliminate implications :

$\equiv \forall(x) [(\neg \forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x,y)) \vee \exists y \text{ Loves}(y,x)$

$\equiv \forall x (\neg \forall y \neg \text{Animal}(y) \vee \text{Loves}(x,y)) \vee \exists y \text{ Loves}(y,x)$

· Move $\neg$ inward

$\neg \forall x p$ becomes $\exists x \neg p$

$\neg \exists x p$ becomes $\forall x \neg p$

$\forall x [\exists y \neg(\neg \text{Animal}(y) \vee \text{Loves}(x,y)] \vee \exists y \text{ Loves}(y,x)$

$\equiv \forall x [\exists y \text{ Animal}(y) \vee \neg \text{Loves}(x,y)] \vee \exists y \text{ Loves}(y,x)$

Standardize variables:

Rename the second y to z

$\forall x [\exists y \text{ Animal}(y) \wedge \neg \text{loves}(x,y)] \vee \exists z \text{ Loves}(z,x)$

Skolemization: create a unique constant for each existing quantifier variables:

$\forall x [\text{Animal}(f(x)) \wedge \neg \text{loves}(x, f(x))] \vee \text{Loves}(g(x), x)$

Drop universal quantifiers:

$[\text{Animal}(f(x)) \wedge \neg \text{loves}(x, f(x))] \vee \text{Loves}(g(x), x)$

Distribute $\vee$ over $\wedge$:

A_1: $[Animal(f(x)) \vee Loves(g(x), x)] \wedge$

A_2: $[\neg Loves(x, f(x)) \vee Loves(g(x), x)]$

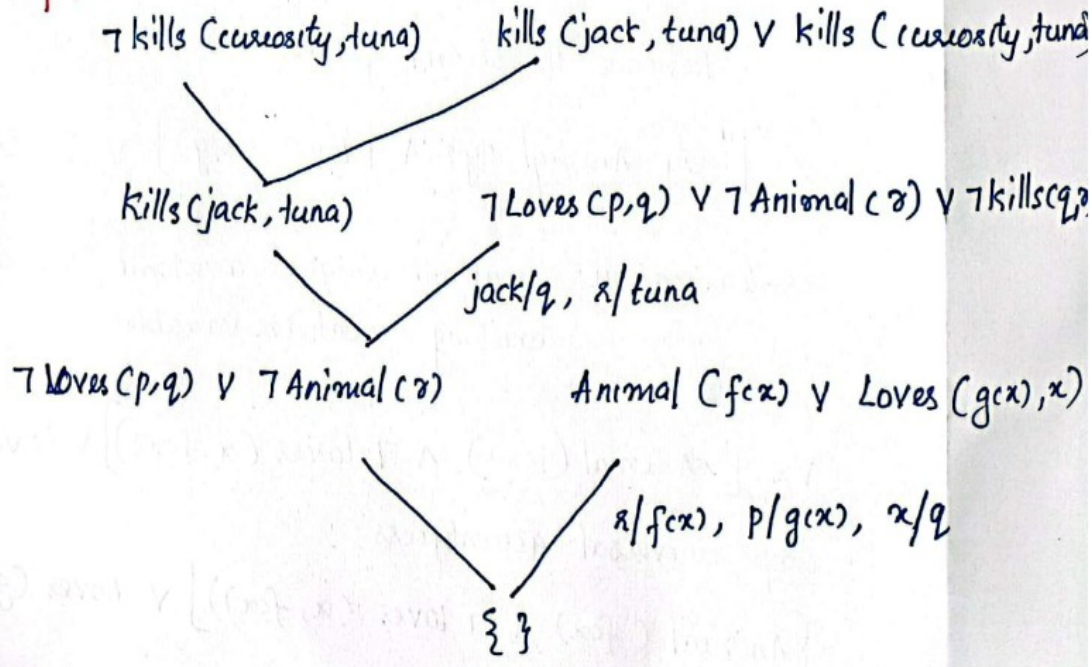
Similarly,

2. $\neg loves(p, q) \vee \neg Animal(r) \vee \neg kills(q, r)$
3. $\neg Animal(s) \vee Loves(jack, s)$
4. $kills(jack, tuna) \vee kills(curiosity, tuna)$
5. $Cat(tuna)$
6. $\neg Cat(t) \vee Animal(t)$

Step 3: Negate the statement to be proved.

$\neg kills(curiosity, tuna)$

Step 4:



Inference engine

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. Inference engine commonly proceeds in two modes, which are:

- Forward chaining
- Backward chaining

Forward Chaining

Forward chaining 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 extract more data 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.

Given a new fact, generate all consequences

Assumes all rules are of the form C1 and C2 and C3 and…. --> Result. Each rule & binding generates a new fact This new fact will "trigger" other rules. Keep going until the desired fact is generated. (Semi-decidable as is FOL in general)

FC: Example Knowledge Base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy America, has some missiles, and all of its missiles were sold to it by Col. West, who is an American.

Prove that Col. West is a criminal.

…it is a crime for an American to sell weapons to hostile nations
$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono…has some missiles
$\exists x\ Owns(Nono, x) \land Missiles(x)$

$Owns(Nono, M_1)$ and $Missle(M_1)$

…all of its missiles were sold to it by Col. West
$\forall x\ Missle(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

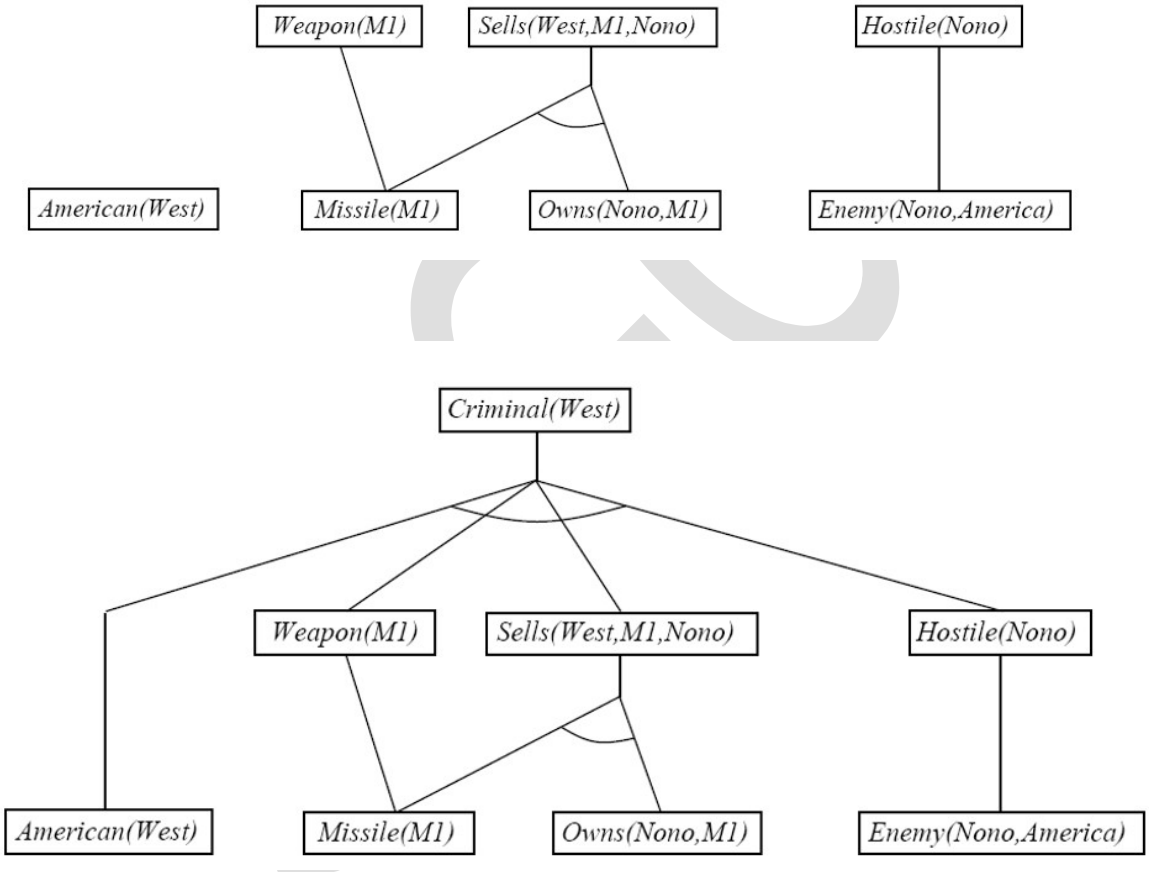
Missiles are weapons
$Missle(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile"
Enemy(x, America) ⇒ Hostile(x)

Col. West who is an American
American(Col. West)

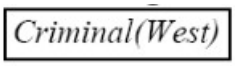
The country Nono, an enemy of America
Enemy(Nono, America)

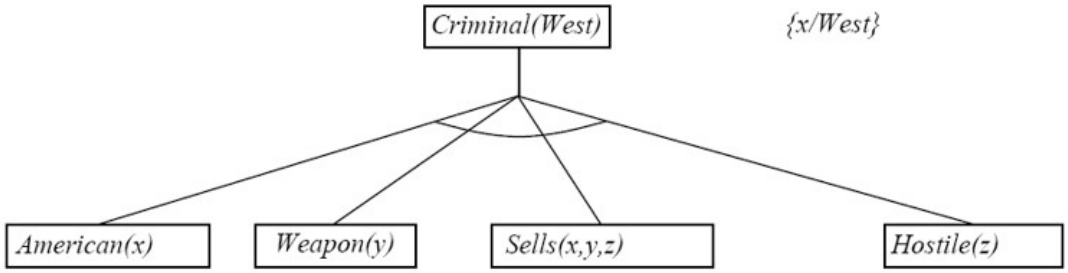
FC: Example Knowledge Base

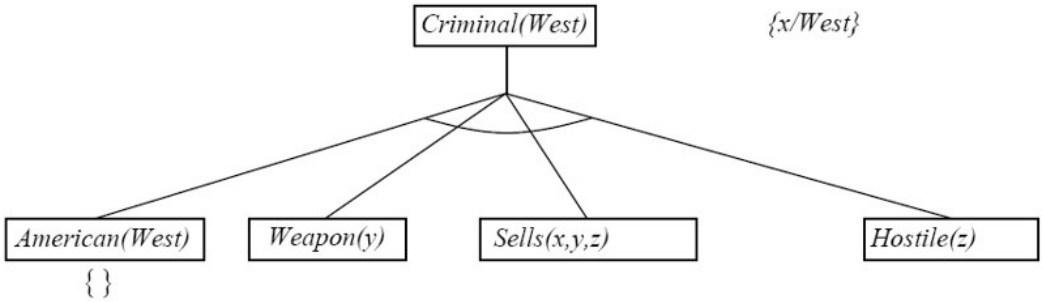


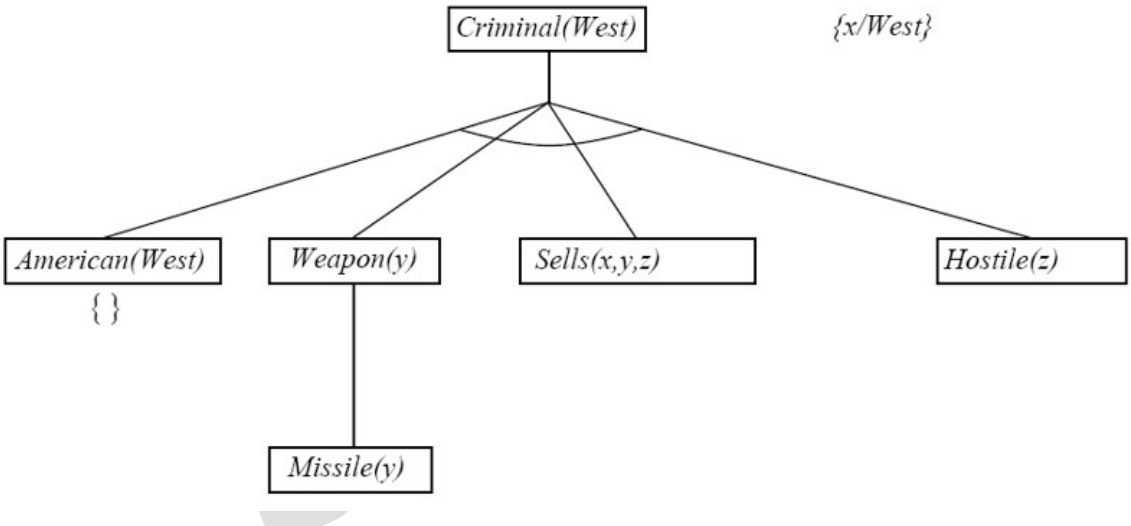
Backward Chaining

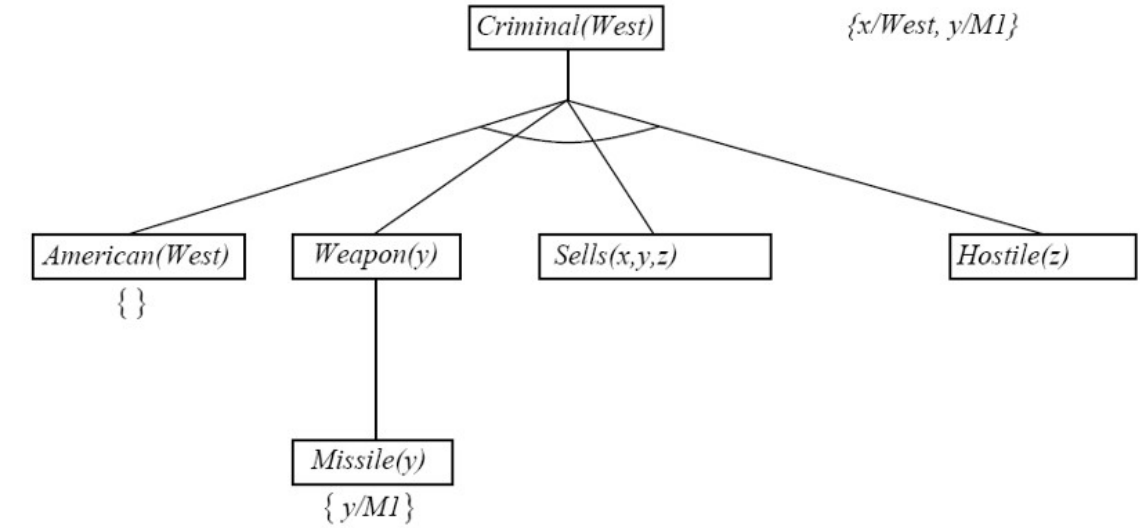
Back Chaining considers the item to be proven a goal. Find a rule whose head is the goal (and bindings). Apply bindings to the body, and prove these (subgoals) in turn. If you prove all the subgoals, increasing the binding set as you go, you will prove the item.

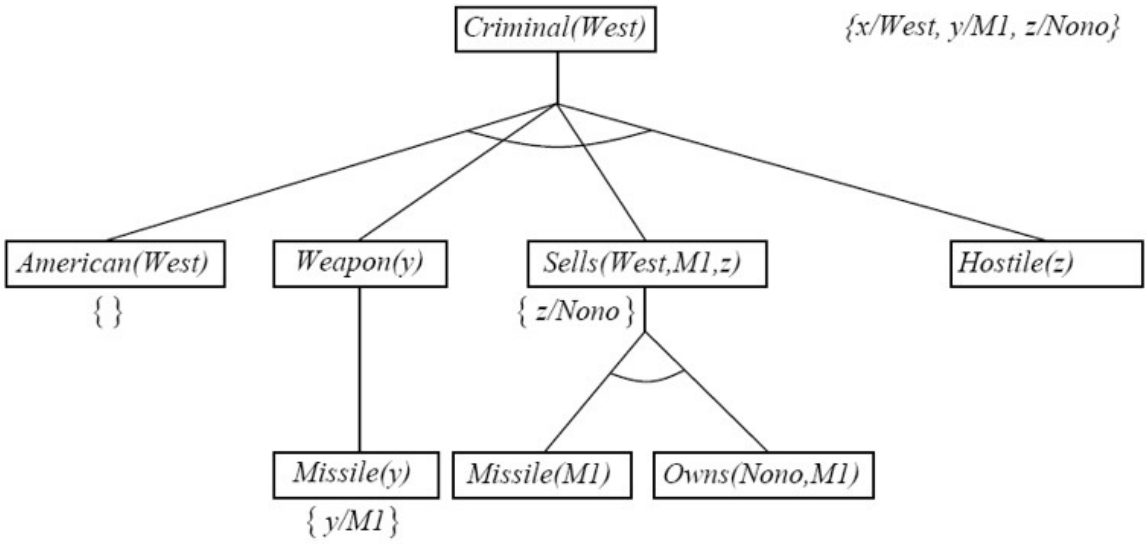


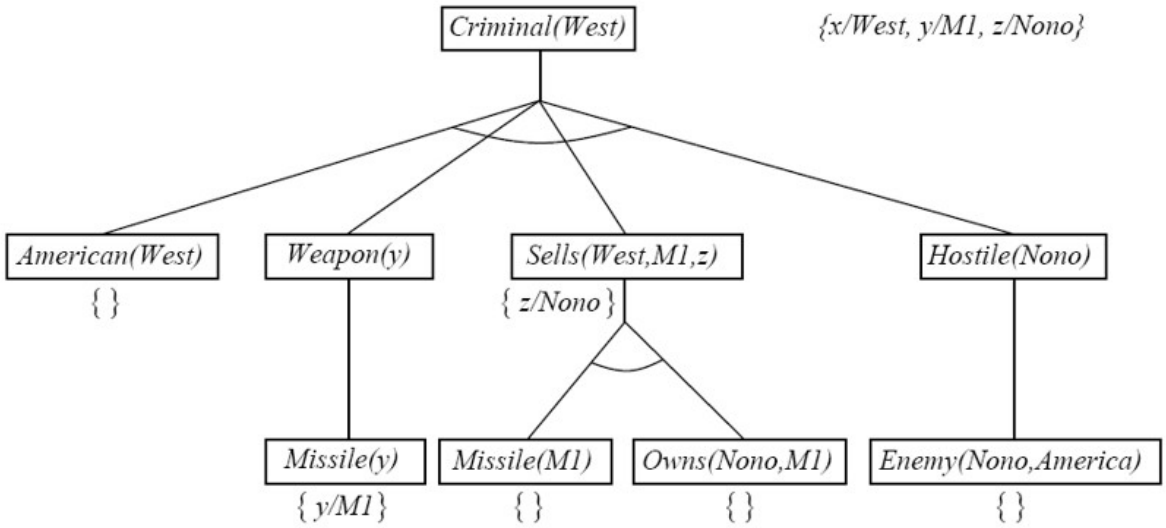

Criminal(West)
{x/West}
American(x)
Weapon(y)
Sells(x,y,z)
Hostile(z)


Criminal(West)
{x/West}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)


Criminal(West)
{x/West}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)
Missile(y)


Criminal(West)
{x/West, y/M1}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)
Missile(y)
{ y/M1 }


Criminal(West)
{x/West, y/M1, z/Nono}
American(West)
{ }
Weapon(y)
Sells(West,M1,z)
{ z/Nono }
Hostile(z)
Missile(y)
{ y/M1 }
Missile(M1)
Owns(Nono,M1)


Criminal(West)
{x/West, y/M1, z/Nono}
American(West)
{ }
Weapon(y)
Sells(West,M1,z)
{ z/Nono }
Hostile(Nono)
Missile(y)
{ y/M1 }
Missile(M1)
{ }
Owns(Nono,M1)
{ }
Enemy(Nono,America)
{ }