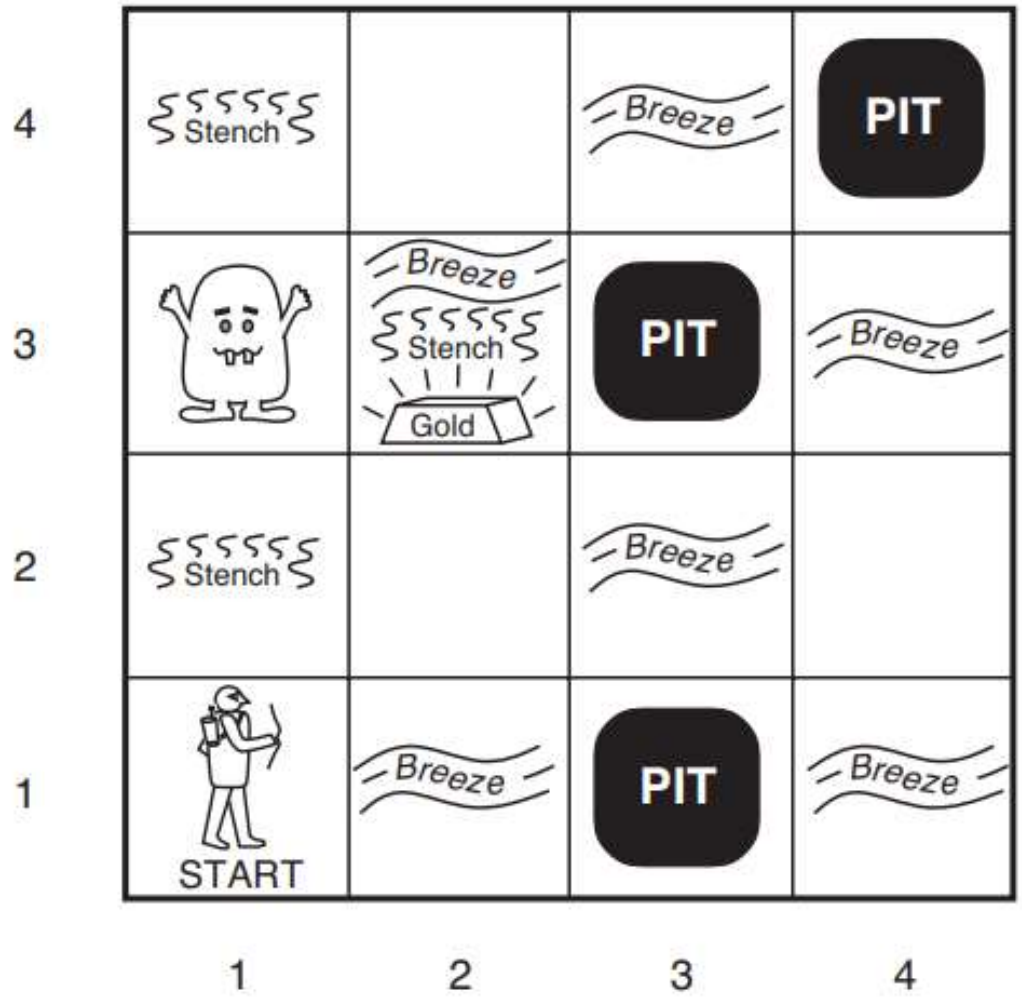
Knowledge Representation

Topics

- *Logical Agents*
  - Knowledge Based Agents
  - Logic
  - Propositional logic
  - First order logic
  - Syntax and Semantics in First order Logic
- *Inference in first order logic*
  - propositional vs. First order inference
  - Unification and Lifting
  - Forward chaining, Backward chaining
  - Resolution

Wumpus W


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

Wumpus World

Performance measure: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: A 4 × 4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.

Wumpus World

Actuators: The agent can move Forward, TurnLeft by 90◦, or TurnRight by 90◦. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and bumps into a wall, then the agent does not move. The action Grab can be used to pick up the gold if it is in the same square as the agent. The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action Climb can be used to climb out of the cave, but only from square [1,1].

Wumpus World

Sensors: The agent has five sensors, each of which gives a single bit of information:

– In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.

– In the squares directly adjacent to a pit, the agent will perceive a Breeze.

– In the square where the gold is, the agent will perceive a Glitter.

– When an agent walks into a wall, it will perceive a Bump.

– When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

Wumpus World

- The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].

Wumpus World

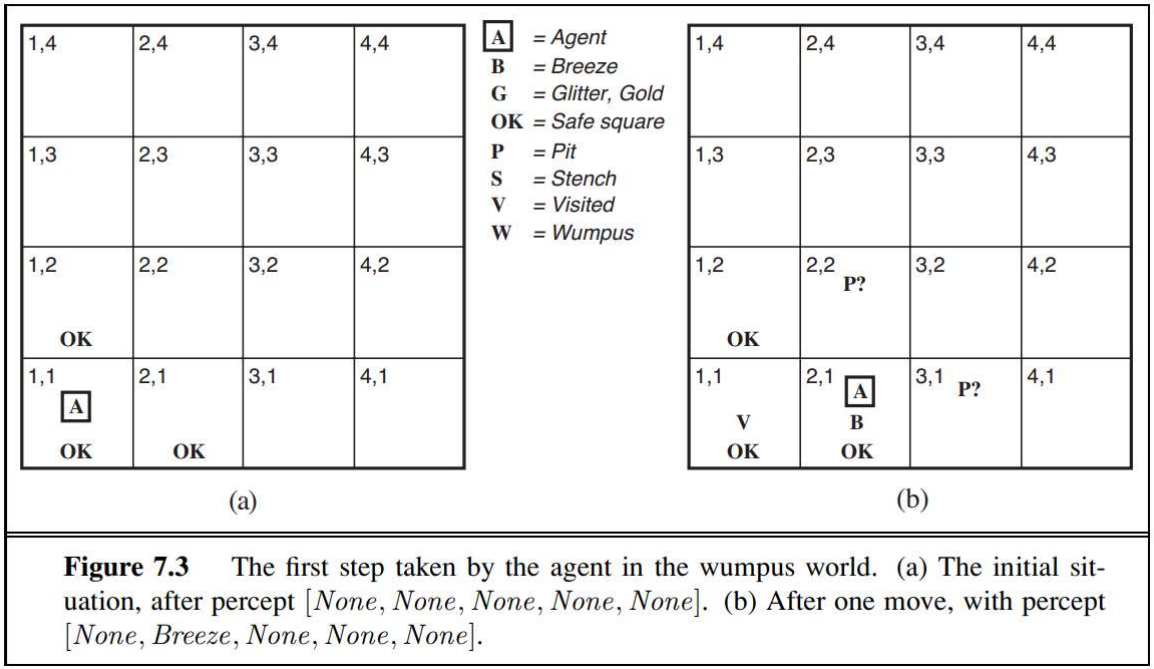


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept $[None, None, None, None, None]$. (b) After one move, with percept $[None, Breeze, None, None, None]$.

Knowledge Based Agents

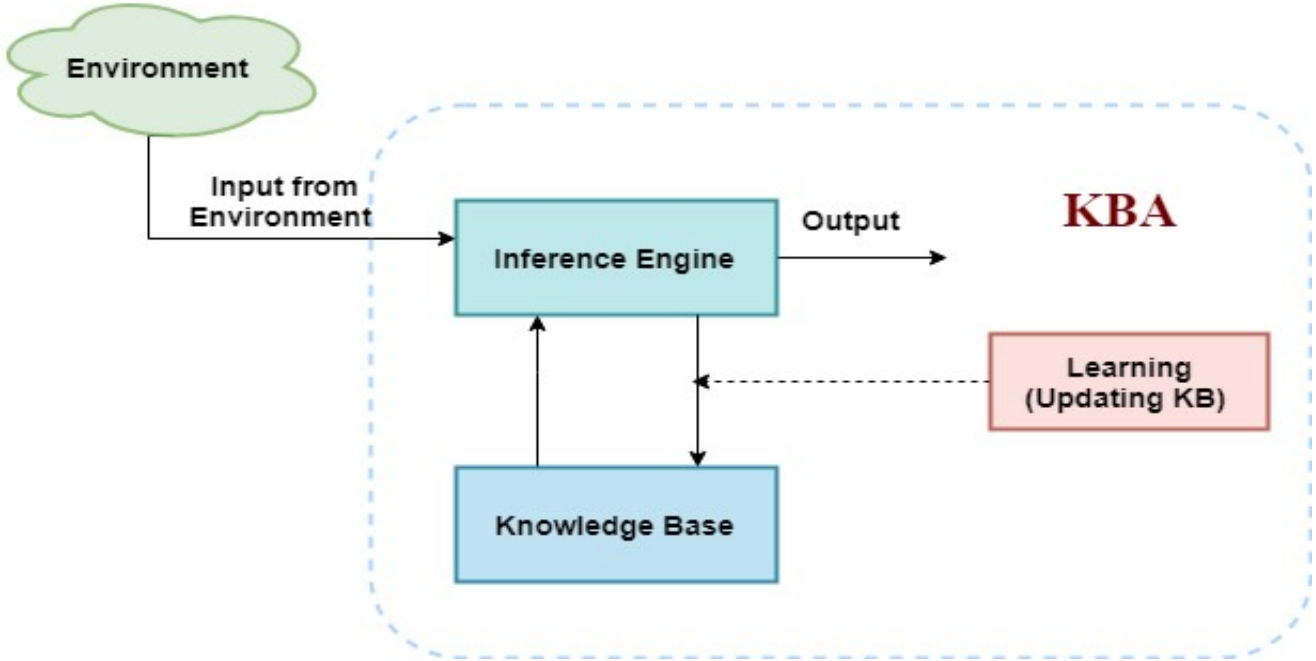
- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.
- Knowledge based agents give the current situation in the form of sentences. They have complete knowledge of current situation of mini-world and its surroundings. These agents manipulate knowledge to infer new things at “Knowledge level”.

Knowledge Based Agents

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts.
- An agent can update the internal representation of the world.
- An agent can deduce the internal representation of the world.
- An agent can deduce appropriate actions.

Knowledge Based Agents Architecture



Types of Agents

- ***Declarative Approach**:* In this beginning from an empty knowledge base, the agent can TELL sentences one after another till the agent has knowledge of how to work with its environment. This is known as the declarative approach. It stores required information in empty knowledge-based system.
- ***Procedural Approach**:* This converts required behaviours directly into program code in empty knowledge-based system. It is a contrast approach when compared to Declarative approach. In this by coding behaviour of system is designed .

Logic

Logic is the basis of all mathematical reasoning, and of all automated reasoning. The rules of logic specify the meaning of mathematical statements. These rules help us understand and reason with statements such as –

$\exists x$ such that $x \neq a^2 + b^2$, where $x, a, b \in Z$

Which in Simple English means “There exists an integer that is not the sum of two squares”. **Importance of Mathematical Logic** The rules of logic give precise meaning to mathematical statements. These rules are used to distinguish between valid and invalid mathematical arguments. Apart from its importance in understanding mathematical reasoning, logic has numerous applications in Computer Science, varying from design of digital circuits, to the construction of computer programs and verification of correctness of programs.

Propositional Logic

A proposition is the basic building block of logic. It is defined as a declarative sentence that is either True or False, but not both. The **Truth Value** of a proposition is True(denoted as T) if it is a true statement, and False(denoted as F) if it is a false statement. For Example,

1. The sun rises in the East and sets in the West.
2. 1 + 1 = 2
3. 'b' is a vowel.

Basic Terminology

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.

Propositional logic

- **Logical constants**: true, false
- **Propositional symbols**: P, Q, S, ... (**atomic sentences**)
- Wrapping **parentheses**: (…)
- Sentences are combined by **connectives**:

∧ ...and	[conjunction]
∨ ...or	[disjunction]
⇒...implies	[implication / conditional]
⇔..is equivalent	if and only if [biconditional]
¬ ...not	[negation]

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

Continued…

```
S := <Sentence> ;
<Sentence> := <AtomicSentence> |
<ComplexSentence> ;
<AtomicSentence> := "TRUE" | "FALSE" |
                    "P" | "Q" | "S" ;
<ComplexSentence> := "(" <Sentence> ")" |
          <Sentence> <Connective> <Sentence> |
          "NOT" <Sentence> ;
<Connective> := "NOT" | "AND" | "OR" | "IMPLIES"
| "EQUIVALENT" ;
```

Examples of Propositional Logic sentences

- P means “It is hot.”
- Q means “It is humid.”
- R means “It is raining.”
- $(P \wedge Q) \rightarrow R$

“If it is hot and humid, then it is raining”

- $Q \rightarrow P$

“If it is humid, then it is hot”

Continued…

- A simple language useful for showing key ideas and definitions
- User defines a set of propositional symbols, like P and Q.
- User defines the **semantics** of each propositional symbol:

P means “It is hot”

Q means “It is humid”

R means “It is raining”

- A sentence (well formed formula) is defined as follows:

A symbol is a sentence

If S is a sentence, then $\neg$S is a sentence

If S is a sentence, then (S) is a sentence

If S and T are sentences, then (S $\vee$ T), (S $\wedge$ T), and (S $\rightarrow$ T), are sentences

Continued…

- A **valid sentence** or **tautology** is a sentence that is True under all interpretations, no matter what the world is actually like or how the semantics are defined. Example: “It’s raining or it’s not raining.”
- An **inconsistent sentence** or **contradiction** is a sentence that is False under all interpretations. The world is never like what it describes, as in “It’s raining and it’s not raining.”
- **P entails Q**, written P |= Q, means that whenever P is True, so is Q. In other words, i.e. in every model in which P is True, Q is also True.

Truth tables

And

p	q	$p \cdot q$
T	T	T
T	F	F
F	T	F
F	F	F

Or

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

If . . . then

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

Not

p	$\sim p$
T	F
F	T

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

- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.

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

Inference

- Inference is the process of deriving new sentences from old

Sound inference derives true conclusions given true premises

Complete inference derives all true conclusions from a set of premises

This section covers **inference rules** that can be applied to derive a **proof**—a chain of conclusions that leads to the desired goal. The best-known rule is called **Modus Ponens** (Latin for *mode that affirms*) and is written

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta} .$$

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

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

The unit resolution rule can be generalized to the full **resolution** rule,

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

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

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

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

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

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

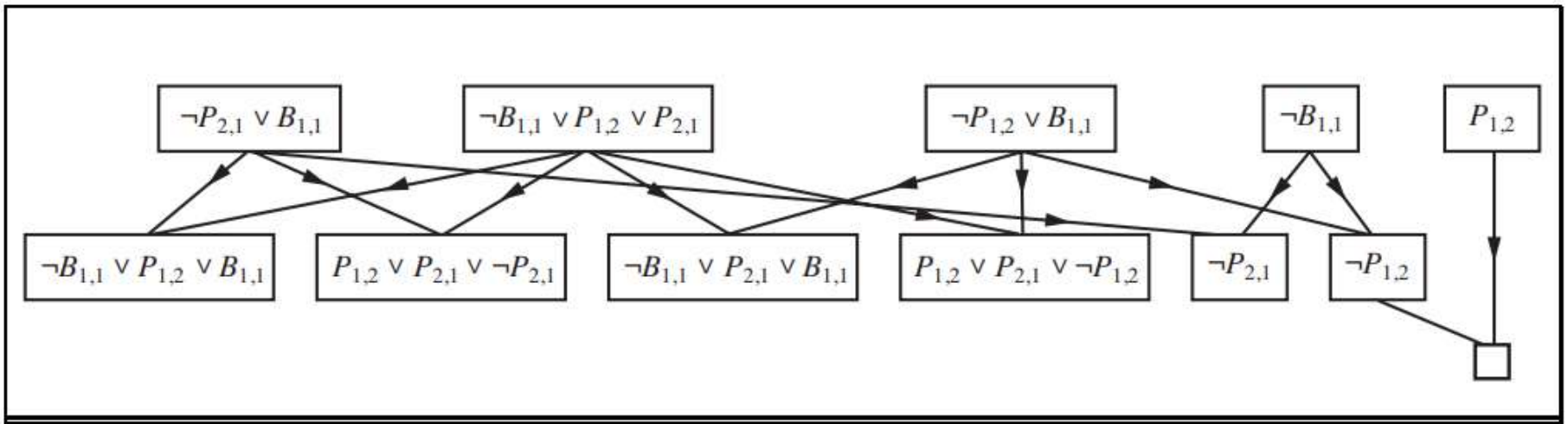


Figure 7.13 Partial application of PL-RESOLUTION to a simple inference in the wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row.

One such restricted form is the **definite clause**, which is a disjunction of literals of which *exactly one is positive*. For example, the clause $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ is a definite clause, whereas $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ is not.

Slightly more general is the **Horn clause**, which is a disjunction of literals of which *at most one is positive*. So all definite clauses are Horn clauses, as are clauses with no positive literals; these are called **goal clauses**. Horn clauses are closed under resolution: if you resolve two Horn clauses, you get back a Horn clause.

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

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is the number of symbols in c's premise

$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols

$agenda \leftarrow$ a queue of symbols, initially symbols known to be true in KB

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

return *false*

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

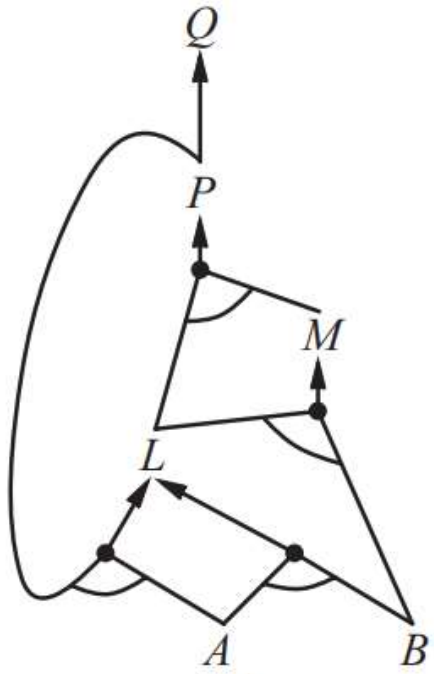
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

(a)



(b)

Figure 7.16 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

Table 4.6 NDS Rules Table

Rule Name	Symbol	Rule	Description
Introducing Λ	$(I{:}\Lambda)$	If $A_1, \ldots, A_n$ then $A_1 \Lambda \ldots \Lambda A_n$	If $A_1, \ldots, A_n$ are true, then their conjunction $A_1 \Lambda \ldots \Lambda A_n$ is also *true*.
Eliminating Λ	$(E{:}\Lambda)$	If $A_1 \Lambda \ldots \Lambda A_n$ then $A_i (1 \le i \le n)$	If $A_1 \Lambda \ldots \Lambda A_n$ is *true*, then any A_i is also *true*.
Introducing V	$(I{:}\mathrm{V})$	If any $A_i (1 \le i \le n)$ then $A_1 \mathrm{V} \ldots \mathrm{V} A_n$	If any $A_i (1 \le i \le n)$ is *true*, then $A_1 \mathrm{V} \ldots \mathrm{V} A_n$ is also *true*.
Eliminating V	$(E{:}\mathrm{V})$	If $A_1 \mathrm{V} \ldots \mathrm{V} A_n, A_1 \to A, \ldots, A_n \to A$ then A	If $A_1 \mathrm{V} \ldots \mathrm{V} A_n, A_1 \to A, A_2 \to A, \ldots,$ and $A_n \to A$ are *true*, then A is *true*.
Introducing $\to$	$(I : \to)$	If from $\alpha_1, \ldots, \alpha_n$ infer β is proved then $\alpha_1 \Lambda \ldots \Lambda \alpha_n \to \beta$ is proved	If given that $\alpha_1, \alpha_2, \ldots,$ and α_n are *true* and from these we deduce β then $\alpha_1 \Lambda \ldots \Lambda \alpha_n \to \beta$ is also *true*.

(Contd.)

Table 4.6 (Contd.)

Rule Name	Symbol	Rule	Description
Eliminating $\rightarrow$	$(E: \rightarrow)$	If $A_1 \rightarrow A, A_1$, then A	If $A_1 \rightarrow A$ and A_1 are *true* then A is also *true*. This is called *Modus Ponen* rule.
Introducing $\leftrightarrow$	$(I: \leftrightarrow)$	If $A_1 \rightarrow A_2$, $A_2 \rightarrow A_1$ then $A_1 \leftrightarrow A_2$	If $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_1$ are *true* then $A_1 \leftrightarrow A_2$ is also *true*.
Elimination $\leftrightarrow$	$(E: \leftrightarrow)$	If $A_1 \leftrightarrow A_2$ then $A_1 \rightarrow A_2, A_2 \rightarrow A_1$	If $A_1 \leftrightarrow A_2$ is *true* then $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_1$ are *true*
Introducing ~	$(I: \sim)$	*If from A infer* $A_1 \wedge \sim A_1$ is proved then $\sim A$ is proved	If from A (which is *true*), a contradiction is proved then truth of $\sim A$ is also proved
Eliminating ~	$(E: \sim)$	*If from* $\sim A$ *infer* $A_1 \wedge \sim A_1$ is proved then A is proved	If from $\sim A$, a contradiction is proved then truth of A is also proved

... NDS written as from α ... α *infer* β leads to the ...

axiomatic system, there are three axioms, which are always true (or
modus ponen (MP). Here, α, β, and γ are well-formed formulae of
three axioms and the rule are stated as follows:

Axiom 1 $\alpha \rightarrow (\beta \rightarrow \alpha)$

Axiom 2 $[\alpha \rightarrow (\beta \rightarrow \gamma)] \rightarrow [(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)]$

Axiom 3 $(\sim\alpha \rightarrow \sim\beta) \rightarrow (\beta \rightarrow \alpha)$

Modus Ponen Rule *Hypotheses:* $\alpha \rightarrow \beta$ and α; *Consequent:* β

Table 4.8 Proof of the Theorem *from* $(A \rightarrow B)$, $(B \rightarrow C)$ *infer* $(A \rightarrow C)$

Description	Formula	Comments	
Theorem	*from* $A \rightarrow B, B \rightarrow C$ *infer* $A \rightarrow C$	To be proved	
Hypothesis 1	$A \rightarrow B$	1	
Hypothesis 2	$B \rightarrow C$	2	
Sub-theorem	*from A infer C*	3	
Hypothesis	A		3.1
E: $\rightarrow$ (1, 3.1)	B		3.2
E: $\rightarrow$ (2, 3.2)	C		3.3
I: $\rightarrow$ (3)	$A \rightarrow C$	Proved	

4.6.1 Semantic Tableau Rules

The semantic tableau rules are given in Table 4.11 where α and β are two formulae.

Table 4.11 Semantic Tableau Rules for α and β

Rule No.	Tableau tree	Explanation
Rule 1	$\alpha \wedge \beta$ *is true if both* α *and* β *are true* $\alpha \wedge \beta$ | α | β	A tableau for a formula $(\alpha \wedge \beta)$ is constructed by adding both α and β to the same path (branch)
Rule 2	$\sim(\alpha \wedge \beta)$ *is true if either* $\sim\alpha$ *or* $\sim\beta$ *is true* $\sim(\alpha \wedge \beta)$ $\sim\alpha$ $\quad$ $\sim\beta$	A tableau for a formula $\sim(\alpha \wedge \beta)$ is constructed by adding two new paths: one containing $\sim\alpha$ and the other containing $\sim\beta$
Rule 3	$\alpha \vee \beta$ *is true if either* α *or* β *is true* $\alpha \vee \beta$ α $\quad$ β	A tableau for a formula $(\alpha \vee \beta)$ is constructed by adding two new paths: one containing α and the other containing β
Rule 4	$\sim(\alpha \vee \beta)$ *is true if both* $\sim\alpha$ *and* $\sim\beta$ *are true* $\sim(\alpha \vee \beta)$ | $\sim\alpha$ | $\sim\beta$	A tableau for a formula $\sim(\alpha \vee \beta)$ is constructed by adding both $\sim\alpha$ and $\sim\beta$ to the same path
Rule 5	$\sim(\sim\alpha)$ *is true then* α *is true* $\sim(\sim\alpha)$ | α	A tableau for $\sim(\sim\alpha)$ is constructed by adding α on the same path
Rule 6	$\alpha \rightarrow \beta$ *is true then* $\sim\alpha \vee \beta$ *is true* $\alpha \rightarrow \beta$ $\sim\alpha$ $\quad$ β	A tableau for a formula $\alpha \rightarrow \beta$ is constructed by adding two new paths: one containing $\sim\alpha$ and the other containing β
Rule 7	$\sim(\alpha \rightarrow \beta)$ *true then* $\alpha \wedge \sim\beta$ *is true* $\sim(\alpha \rightarrow \beta)$ | α | $\sim\beta$	A tableau for a formula $\sim(\alpha \rightarrow \beta)$ is constructed by adding both α and $\sim\beta$ to the same path

(Cont

Example 4.9 Show that $\alpha : (A \wedge B) \wedge (B \rightarrow \sim A)$ is unsatisfiable using the tableau method.

Solution It can be proven that α: $(A \wedge B) \wedge (B \rightarrow \sim A)$ is unsatisfiable as shown in Table 4.13.

Table 4.13 Tableau Method for Example 4.9

Description	Formula	Line number
Tableau root	$(A \wedge B) \wedge (B \rightarrow \sim A)$	1
Rule 1 (1)	$A \wedge B$	2
	$B \rightarrow \sim A$	3
Rule 1 (2)	A	4
	B	5
Rule 6 (3)	$\sim B$ | $\times \{B, \sim B\}$; $\sim A$ | $\times \{A, \sim A\}$	

Example 4.5 Establish that $A \rightarrow C$ is a deductive consequence of $\{A \rightarrow B, B \rightarrow C\}$, i.e., $\{A \rightarrow B, B \rightarrow C\} \vdash (A \rightarrow C)$.

Solution We can prove the theorem as shown below in Table 4.9.

Table 4.9 Proof of the theorem $\{A \rightarrow B, B \rightarrow C\} \vdash (A \rightarrow C)$

Description	Formula	Comments
Theorem	$\{A \rightarrow B, B \rightarrow C\} \vdash (A \rightarrow C)$	*Prove*
Hypothesis 1	$A \rightarrow B$	1
Hypothesis 2	$B \rightarrow C$	2
Instance of Axiom 1	$(B \rightarrow C) \rightarrow [A \rightarrow (B \rightarrow C)]$	3
MP (2, 3)	$[A \rightarrow (B \rightarrow C)]$	4
Instance of Axiom 2	$[A \rightarrow (B \rightarrow C)] \rightarrow [(A \rightarrow B) \rightarrow (A \rightarrow C)]$	5
MP (4, 5)	$(A \rightarrow B) \rightarrow (A \rightarrow C)$	6
MP (1, 6)	$(A \rightarrow C)$	*Proved*

Hence, we can conclude that $A \rightarrow C$ is a deductive consequence of $\{A \rightarrow B, B \rightarrow C\}$.

Example 4.15 Find resolvent of the clauses in the set $\{A \vee B, \sim A \vee D, C \vee \sim B\}$.

Solution The method of resolution is shown in Fig. 4.1.

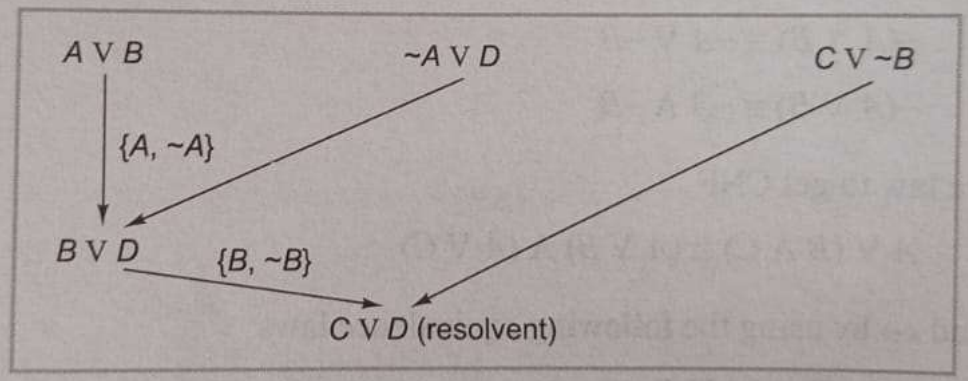


Figure 4.1 Resolution of clauses of Example 4.15

We can clearly see that $C \vee D$ is a resolvent of the set $\{A \vee B, \sim A \vee D, C \vee \sim B\}$.

Now that we are familiar with the resolution process, we can state a few results.

Table 4.11 (Contd.)

Rule No.	Tableau tree	Explanation
Rule 8	$\alpha \leftrightarrow \beta$ *is true then* $(\alpha \wedge \beta) \vee (\sim \alpha \wedge \sim \beta)$ *is true* $\alpha \leftrightarrow \beta$ $\alpha \wedge \beta$ $\quad$ $\sim\alpha \wedge \sim\beta$	A tableau for a formula $\alpha \leftrightarrow \beta$ is constructed by adding two new paths: one containing $\alpha \wedge \beta$ and other $\sim\alpha \wedge \sim\beta$ which are further expanded
Rule 9	$\sim(\alpha \leftrightarrow \beta)$ is *true* then $(\alpha \wedge \sim\beta) \vee (\sim\alpha \wedge \beta)$ *is true* $\sim(\alpha \leftrightarrow \beta)$ $\alpha \wedge \sim\beta$ $\quad$ $\sim\alpha \wedge \beta$	A tableau for a formula $\sim(\alpha \leftrightarrow \beta)$ is constructed by adding two new paths: one containing $\alpha \wedge \sim\beta$ and the other $\sim\alpha \wedge \beta$ which are further expanded

Let us consider an example to illustrate the method of constructing semantic tableau for a for-

rule number applied on line number. The second column contains the derivation of semantic tableau and last column contains the line number.

Example 4.7 Construct a semantic tableau for a formula $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$.

Solution The construction of the semantic tableau for the given formula $(A \wedge \sim B) \wedge (\sim B \rightarrow C)$ is shown in Table 4.12.

Table 4.12 Semantic Tableau for Example 4.7

Description	Formula	Line number
Tableau root	$(A \wedge \sim B) \wedge (\sim B \rightarrow C)$	1
Rule 1 (1)	$A \wedge \sim B$	2
	$\sim B \rightarrow C$	3
Rule 1 (2)	A	4
	$\sim B$	5
Rule 6 (3)	$\sim(\sim B)$ $\quad$ C	6
Rule 3 (6)	B $\quad$ $\surd$(open)	
	$\times$ (closed) $\{B, \sim B\}$	

4.7.2 Conversion of a Formula to its CNF

Any formula in propositional logic can be easily transformed into its equivalent CNF representation by using the equivalence laws described below. The following steps are taken to transform a formula to its equivalent CNF.

- Eliminate double negation signs by using

$$\sim(\sim A) \equiv A$$

- Use De Morgan's Laws to push ~ (negation) immediately before the atomic formula

$$\sim(A \wedge B) \equiv \sim A \vee \sim B$$

$$\sim(A \vee B) \equiv \sim A \wedge \sim B$$

- Use distributive law to get CNF

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

- Eliminate $\rightarrow$ and $\leftrightarrow$ by using the following equivalence laws:

$$A \rightarrow B \equiv \sim A \vee B$$

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$

Advantages Vs Limitations

- Advantages
  - Simple KR language sufficient for some problems
  - Lays the foundation for higher logics (e.g., FOL)
  - Reasoning is decidable, though NP complete, and efficient techniques exist for many problems
- Limitations
  - Not expressive enough for most problems
  - Even when it is, it can very “un-concise”

PL is a weak KR language

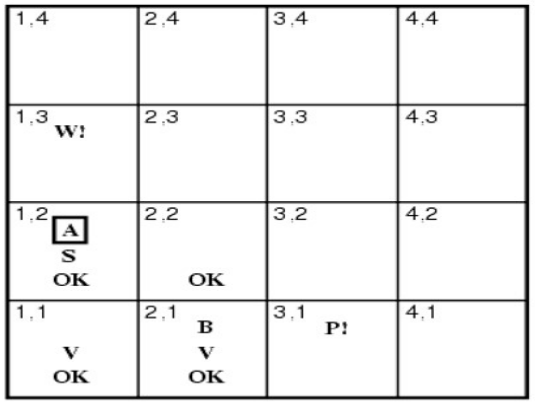
- Hard to identify “individuals” (e.g., Mary, 3)
- Can’t directly talk about properties of individuals or relations between individuals (e.g., “Bill is tall”)
- Generalizations, patterns, regularities can’t easily be represented (e.g., “all triangles have 3 sides”)
- First-Order Logic (FOL) is expressive enough to represent this kind of information using relations, variables and quantifiers, e.g.,

  Every elephant is gray: $\forall$ x (elephant(x) $\rightarrow$ gray(x))

  There is a white alligator: $\exists$ x (alligator(X) ^ white(X))

Hunt the Wumpus domain

- Some atomic propositions:
  - S12 = There is a stench in cell (1,2)
  - B34 = There is a breeze in cell (3,4)
  - W22 = Wumpus is in cell (2,2)
  - V11 = We've visited cell (1,1)
  - OK11 = Cell (1,1) is safe.
  - …



A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

- Some rules:
  - *(R1)* $\neg S11 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W21$
  - *(R2)* $\neg S21 \rightarrow \neg W11 \wedge \neg W21 \wedge \neg W22 \wedge \neg W31$
  - *(R3)* $\neg S12 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W22 \wedge \neg W13$
  - *(R4)* $S12 \rightarrow W13 \vee W12 \vee W22 \vee W11$
  - …
- The lack of variables requires us to give similar rules for each cell!

Proving Wumpus in W13

Apply MP with $\neg$S11 and R1:

$\neg$ W11 $\wedge$ $\neg$ W12 $\wedge$ $\neg$ W21

Apply And-Elimination to this, yielding 3 sentences:

$\neg$ W11, $\neg$ W12, $\neg$ W21

Apply MP to ~S21 and R2, then apply And-elimination:

$\neg$ W22, $\neg$ W21, $\neg$ W31

Apply MP to S12 and R4 to obtain:

W13 $\vee$ W12 $\vee$ W22 $\vee$ W11

Apply Unit resolution on (W13 $\vee$ W12 $\vee$ W22 $\vee$ W11) and $\neg$W11:

W13 $\vee$ W12 $\vee$ W22

Apply Unit Resolution with (W13 $\vee$ W12 $\vee$ W22) and $\neg$W22:

W13 $\vee$ W12

Apply UR with (W13 $\vee$ W12) and $\neg$W12:

W13

First-order 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, companies, cars ...
Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
Properties: blue, oval, even, large, ...
Functions: father-of, best-friend, second-half, one-more-than ...

Elements of FOL

- **Variable symbols**

  E.g., x, y, foo

- **Connectives**

  Same as in PL: not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), if and only if (biconditional $\leftrightarrow$)

- **Quantifiers**

  Universal $\forall$**x** or **(Ax)**

  Existential $\exists$**x** or **(Ex)**

Quantifiers

◉**Universal quantification**

(∀x)P(x) means that P holds for **all** values of x in the domain associated with that variable. Universal quantifiers are often used with "implies" to form "rules":

E.g., (∀x) dolphin(x) → mammal(x)

◉**Existential quantification**

(∃ x)P(x) means that P holds for **some** value of x in the domain associated with that variable. Existential quantifiers are usually used with "and" to specify a list of properties about an individual

E.g., (∃ x) mammal(x) ∧ lays-eggs(x)

Permits one to make a statement about some object without naming it

Quantifier Scope

- Switching the order of universal quantifiers *does not* change the meaning:
  - $(\forall x)(\forall y)P(x,y) \leftrightarrow (\forall y)(\forall x)\ P(x,y)$
- Similarly, you can switch the order of existential quantifiers:
  - $(\exists x)(\exists y)P(x,y) \leftrightarrow (\exists y)(\exists x)\ P(x,y)$
- Switching the order of universals and existentials *does* change meaning:
  - Everyone likes someone: $(\forall x)(\exists y)$ likes(x,y)
  - Someone is liked by everyone: $(\exists y)(\forall x)$ likes(x,y)

Translating English to FOL

Every gardener likes the sun.

$\forall x$ gardener(x) $\rightarrow$ likes(x,Sun)

You can fool some of the people all of the time.

$\exists x \; \forall t$ person(x) $\wedge$time(t) $\rightarrow$ can-fool(x,t)

You can fool all of the people some of the time.

$\forall x \; \exists t$ (person(x) $\rightarrow$ time(t) $\wedge$can-fool(x,t))

$\forall x$ (person(x) $\rightarrow \exists t$ (time(t) $\wedge$can-fool(x,t)))

Equivalent

All purple mushrooms are poisonous.

$\forall x$ (mushroom(x) $\wedge$ purple(x)) $\rightarrow$ poisonous(x)

No purple mushroom is poisonous.

$\neg \exists x$ purple(x) $\wedge$ mushroom(x) $\wedge$ poisonous(x)

$\forall x$ (mushroom(x) $\wedge$ purple(x)) $\rightarrow \neg$poisonous(x)

Equivalent

There are exactly two purple mushrooms.

$\exists x \; \exists y$ mushroom(x) $\wedge$ purple(x) $\wedge$ mushroom(y) $\wedge$ purple(y) ^ $\neg$(x=y) $\wedge \forall z$ (mushroom(z) $\wedge$ purple(z)) $\rightarrow$ ((x=z) $\vee$ (y=z))

Clinton is not tall.

$\neg$tall(Clinton)

X is above Y iff X is on directly on top of Y or there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.

$\forall x \; \forall y$ above(x,y) $\leftrightarrow$ (on(x,y) $\vee \; \exists z$ (on(x,z) $\wedge$ above(z,y)))

Propositional Logic Vs FOL

- Propositional Logic converts a complete sentence into a symbol and makes it logical whereas in First-Order Logic relation of a particular sentence will be made that involves relations, constants, functions, and constants.
- The limitation of PL is that it does not represent any individual entities whereas FOL can easily represent the individual establishment that means if you are writing a single sentence then it can be easily represented in FOL.
- PL does not signify or express the generalization, specialization or pattern for example ‘QUANTIFIERS’ cannot be used in PL but in FOL users can easily use quantifiers as it does express the generalization, specialization, and pattern.

Propositional Logic	Predicate Logic
Propositional logic is the logic that deals with a collection of declarative statements which have a truth value, true or false.	Predicate logic is an expression consisting of variables with a specified domain. It consists of objects, relations and functions between the objects.
It is the basic and most widely used logic. Also known as Boolean logic.	It is an extension of propositional logic covering predicates and quantification.
A proposition has a specific truth value, either true or false.	A predicate's truth value depends on the variables' value.
Scope analysis is not done in propositional logic.	Predicate logic helps analyze the scope of the subject over the predicate. There are three quantifiers : Universal Quantifier (∀) depicts for all, Existential Quantifier (∃) depicting there exists some and Uniqueness Quantifier (∃!) depicting exactly one.
Propositions are combined with Logical Operators or Logical Connectives like Negation(¬), Disjunction(∨), Conjunction(∧), Exclusive OR(⊕), Implication(⇒), Bi-Conditional or Double Implication(⇔).	Predicate Logic adds by introducing quantifiers to the existing proposition.
It is a more generalized representation.	It is a more specialized representation.
It cannot deal with sets of entities.	It can deal with set of entities with the help of quantifiers.

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences.

Substitution:
Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality: First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.
Example: Brother (a) = b.

The equality symbol can also be used with negation to represent that two terms are not the same objects.
Example: ¬(a=b) which is equivalent to a ≠ b.

Continued…

Universal Generalization:

•Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).

•It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.

•**Example:** Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

Universal Instantiation:

- UI is a valid inference rule. It can be applied multiple times to add new sentences.

we can infer any sentence obtained by substituting a ground term for the variable. The rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.

•It can be represented as $\frac{\forall x\, P(x)}{P(c)}$

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that "John likes ice-cream" => P(c)

Continued…

Existential Instantiation: Existential instantiation is a valid inference rule in first-order logic.
•It can be applied only once to replace the existential sentence.
•The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
•It can be represented as $\frac{\exists x\, P(x)}{P(c)}$

∃x Crown(x) ∧ OnHead(x, John),
So we can infer: **Crown(K) ∧ OnHead(K, John),** as long as K does not appear in the knowledge

Existential introduction
•An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
•This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
•It can be represented as $\frac{P(c)}{\exists x P(x)}$
•Example: Let's say that, "Priyanka got good marks in English." "Therefore, someone got good marks in English."

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.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\mathbf{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.

Continued…

The **UNIFY** algorithm takes two sentences and returns a unifier for them if one exists:

Substitution means replacing one variable with another term. It takes two literals as input and make them identical using substitution. It returns fail if the expressions do not match with each other.

UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)

Example: 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 represented as **a/x and f(z)/y.**

With both the substitution the first expression will be identical to the second expression and the substitution set will be **[a/x, f(z)/y]**

Example

- UNIFY$(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Mary)$	$\{x/Mary, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, Mary)$	$fail$

- Standardizing apart eliminates overlap of variables, e.g., $Knows(z_{17}, Mary)$

$Knows(John, x)$	$Knows(z_{17}, Mary)$	$\{z_{17}/John, x/Mary\}$

Find the unification 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}.

Resolution

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**

- **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**

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

- Fact
  - *All hounds howl at night.*
  - *John likes all kind of food.*
  - *John likes peanuts.*

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

Step-1: Conversion of facts into first-order logic

- $\forall x\ (HOUND(x) \rightarrow HOWL(x))$
- $\forall x\ \neg\ food(x) \rightarrow likes(John, x)$
- $likes(John, Peanuts)$

a. $\forall x$: food(x) → likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. $\forall x\ \forall y$: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

e. $\forall x$: eats(Anil, x) → eats(Harry, x)

f. $\forall x$: ¬ killed(x) → alive(x) } **added predicates.**

g. $\forall x$: alive(x) →¬ killed(x) }

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

Eliminate all implication (→) and rewrite:

- ∀x ¬ HOUND(x) ∨ HOWL(x)
- ∀x ¬ food(x) V likes(John, x)
- likes(John, Peanuts)

Move negation (¬)inwards and rewrite

- ∀x ¬ HOUND(x) ∨ HOWL(x)
- ∀x ¬ food(x) V likes(John, x)
- likes(John, Peanuts)

Eliminate all implication (→) and rewrite:

1. ∀x ¬ food(x) V likes(John, x)
2. food(Apple) Λ food(vegetables)
3. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
6. ∀x¬ [¬ killed(x)] V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(John, Peanuts).

Move negation (¬)inwards and rewrite

1. ∀x ¬ food(x) V likes(John, x)
2. food(Apple) Λ food(vegetables)
3. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
6. ∀x ¬killed(x)] V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(John, Peanuts).

Step-2 Continued…

Rename variables or standardize variables

- *¬ HOUND(x) ∨ HOWL(x)*
- *∀x ¬ food(x) V likes(John, x)*
- *likes(John, Peanuts)*

Eliminate existential instantiation quantifier by elimination (Skolemization)

- *¬ HOUND(x) ∨ HOWL(x)*
- *∀x ¬ food(x) V likes(John, x)*
- *likes(John, Peanuts)*

Step-2 Continued…

Drop Universal quantifiers

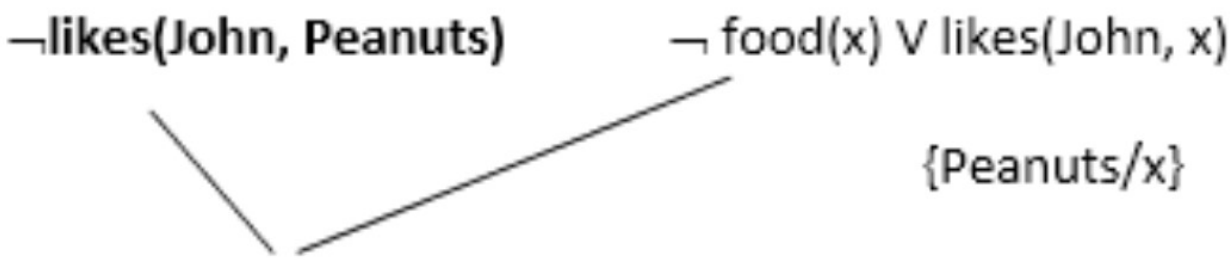
- ¬ *HOUND(x) ∨ HOWL(x)*
- ¬ *food(x) V likes(John, x)*
- *likes(John, Peanuts)*

Step-3: Negate the statement which needs to prove

In this statement, we will apply negation to the conclusion statements

- ¬ *HOUND(x) v HOWL(x)*
- ¬ *food(x) V likes(John, x)*
- ¬ *likes(John, Peanuts)*

Step-4: Draw Resolution graph



•First step: **¬likes(John, Peanuts)** , and **likes(John, x)** get resolved(canceled) by substitution of {Peanuts/x}, and we are left with **¬ food(Peanuts)**

Horn Clause and Definite clause

- **Definite clause:** A clause which is a disjunction of literals with **exactly one positive literal** is known as a 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 the definite clauses are horn clauses.
- **Example: (¬ p V ¬ q V k)**. It has only one positive literal k.

It is equivalent to $p \wedge q \rightarrow k$.

Forward Chaining and backward chaining

The inference engine is an artificial intelligence component that applies logical principles to the knowledge base to infer new information from known facts. The expert system included the first inference engine. Inference engines often operate in two modes:

- **Forward chaining**
- **Backward chaining**

Forward chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.

- It is a bottom-up approach for drawing the inferences.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state. And, is also called as data-driven

Forward Chaining 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."

Robert is criminal.

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)

Continued…

- 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)

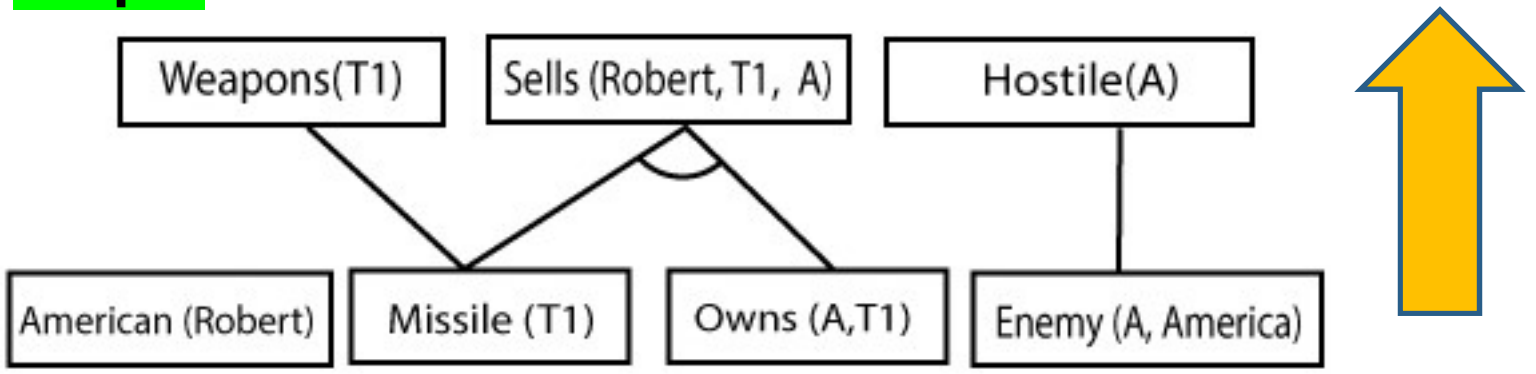
- Robert is American

Criminal(Robert). .....(8)

Forward Chaining Proof

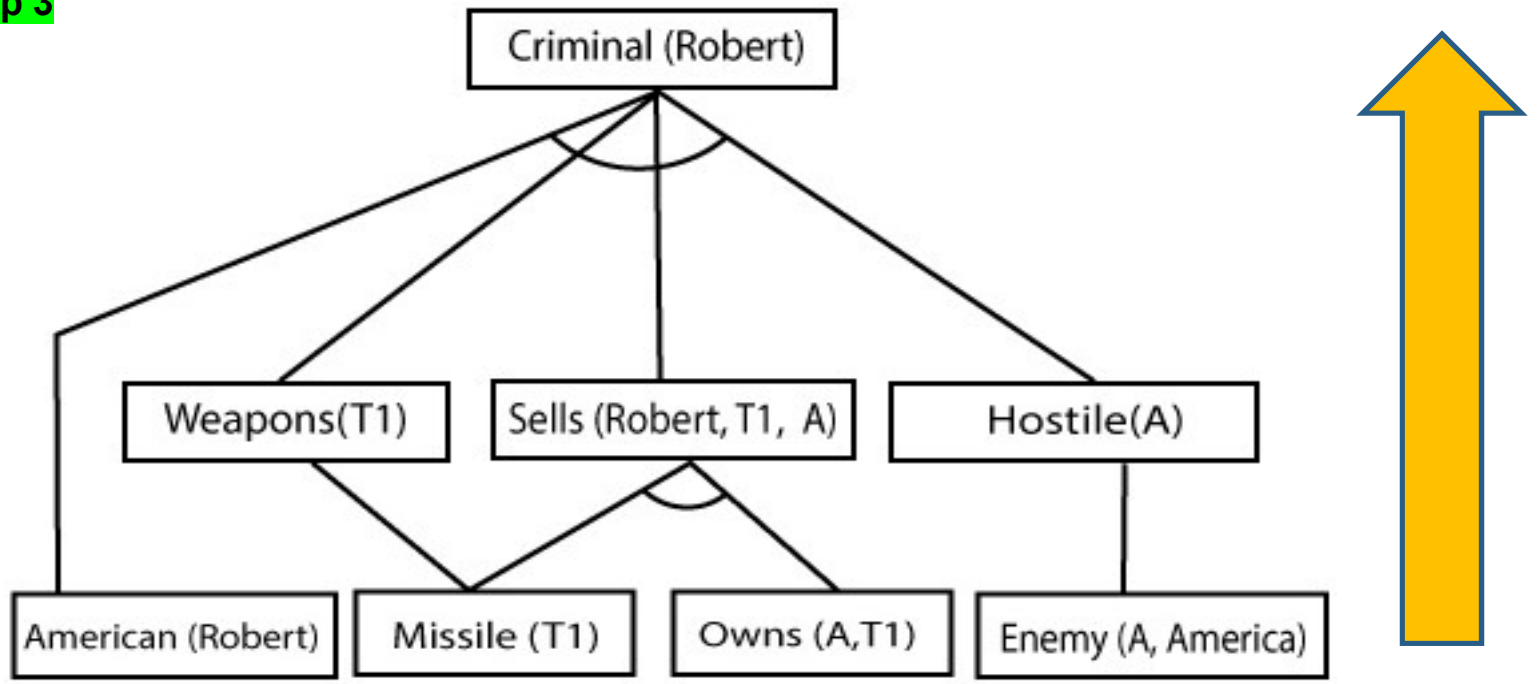
Step 1

Step 2



Continued…

Step 3



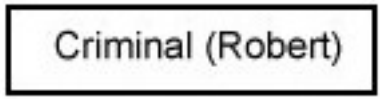
Backward Chaining

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.

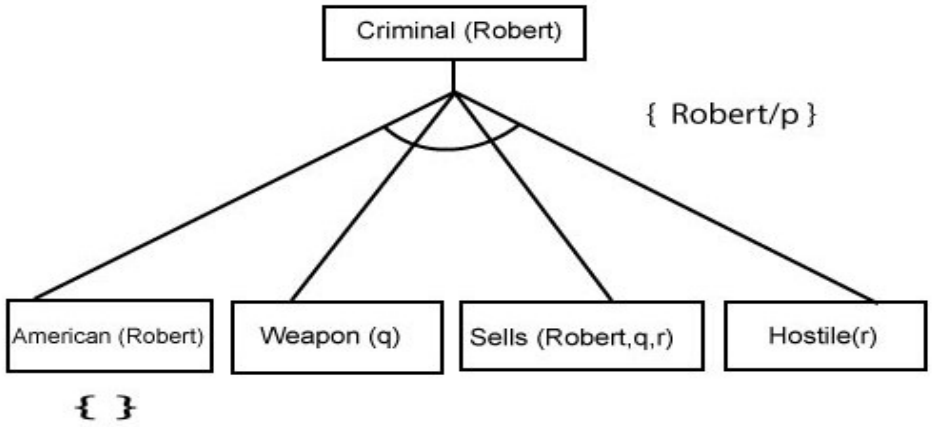
- It is known as a top-down approach.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

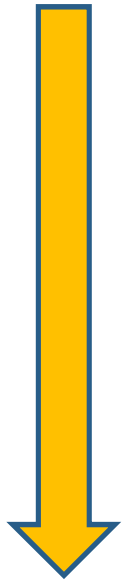
Backward Chaining

Step-1:



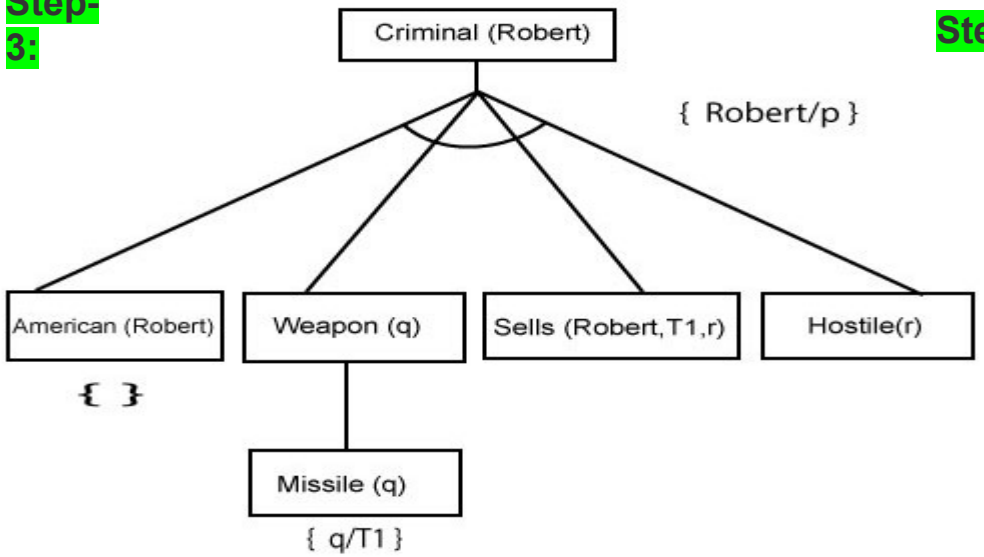
Step-2:



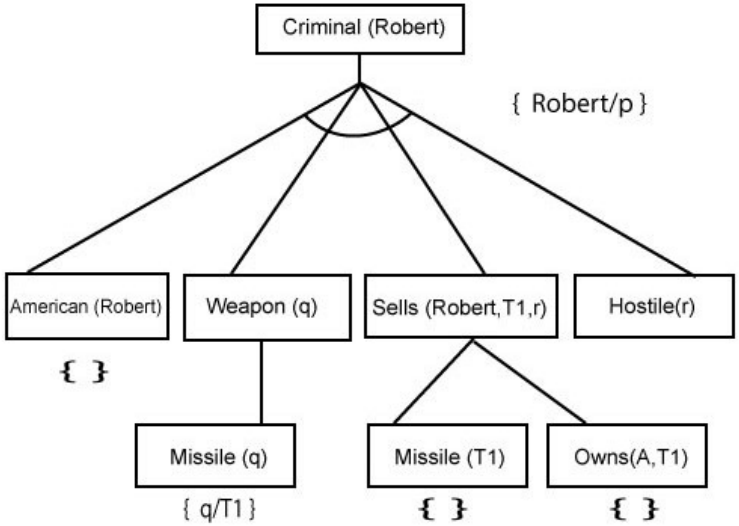


Continued…

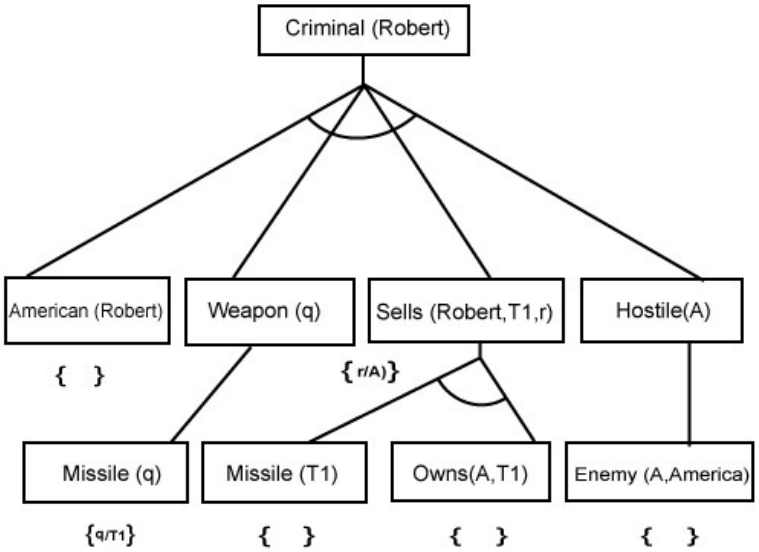
Step-3:



Step-4:



Continued…


Criminal (Robert)
American (Robert)
Weapon (q)
Sells (Robert,T1,r)
Hostile(A)
{ }
{r/A}
Missile (q)
Missile (T1)
Owns(A,T1)
Enemy (A,America)
{q/T1}
{ }
{ }
{ }