RAMAIAH Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Artificial Intelligence(AI)
Course Code: CS53
Credits: 3:0:0:1

UNIT -2

Term: October 2021 – Feb 2022

Faculty:
Dr. Sangeetha V
Assistant Professor
Department of CSE

UNIT -2 Outline

Knowledge-based agents

- Wumpus world
- Logic
- Propositional (Boolean) logic
- Reasoning patterns in Propositional logic
- Equivalence Proportional Model checking
- Agents based on Propositional logic

First Order Logic

- Representation revisited
- Syntax and semantics of first-order logic
- Using first-order logic
- Knowledge engineering in first-order logic

Inference in First Order Logic

- Propositional vs. First-order inference
- Unification and lifting
- Forward chaining
- Backward chaining
- Resolution

Introduction

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

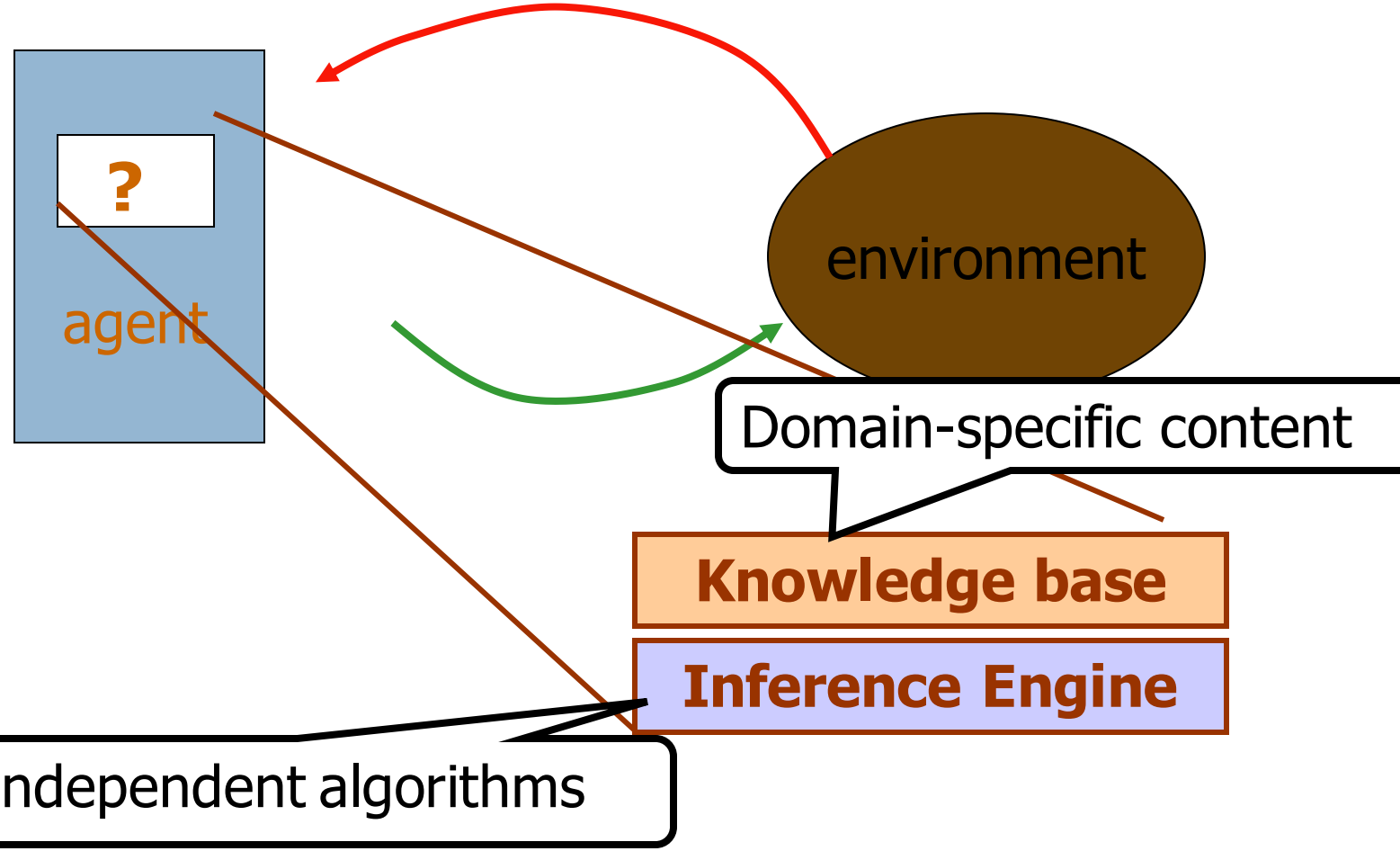
Knowledge - facts, skills, education, experience

Intelligence – ability to use that knowledge

Reasoning – Processing of Knowledge

Knowledge based Agent

- Knowledge-based agent uses some task-specific knowledge to solve a problem efficiently.
- Every Knowledge-based agent has a knowledge base and an inference system



Knowledge based Agent

Central component of Knowledge based Agent is a Knowledge-base(KB)

KB = set of sentences in a formal language

Two generic functions are

1. TELL – add new sentence to KB (TELL IT WHAT IT NEEDS TO KNOW)
2. ASK – Query what is know from KB (ASK WHAT TO DO NEXT)

Tell facts
Ask for inference

For example:

- TELL: Father of John is Bob
- TELL: Jane is John's sister
- TELL: John's Father is the same as John's sister's father
- ASK: Who's Jane's Father?

Knowledge based Agent

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.

Generic KB-Based Agent

Agent maintains a knowledge base KB in 3 steps:

- First it TELLS the KB what it perceives
- It asks the KB what action it should perform
- Process of answering the query requires lots of reasoning

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
```

- MAKE-PERCEPT-SENTENCE generates a sentence, what the agent perceived at the given time.
- MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.
- MAKE-ACTION-SENTENCE generates a sentence which asserts that the chosen action was executed.

Approaches to designing a knowledge-based agent

There are mainly two approaches to build a knowledge-based agent:

Declarative approach: We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with.

Example : Constraints and rules

Procedural approach: We directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

Example: Functions

Wumpus World

Wumpus world – example to illustrate the worth of a knowledge based agent and to represent knowledge

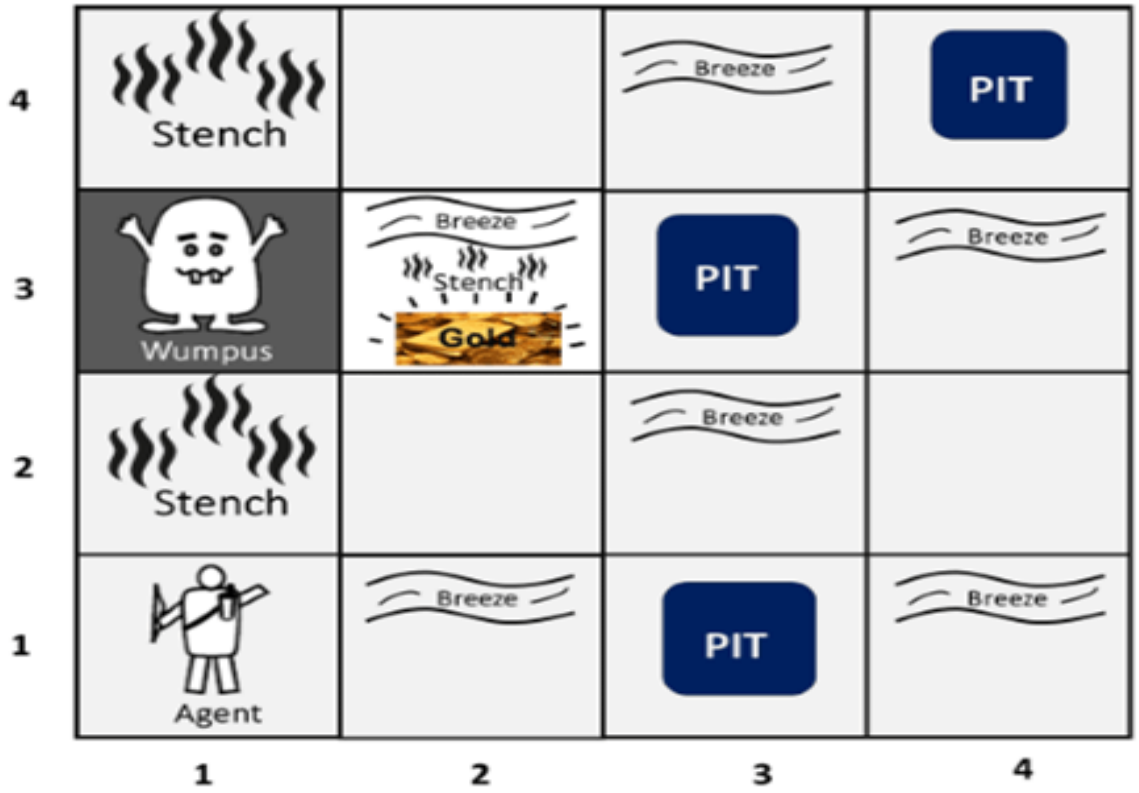
It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973

A Wumpus World

The wumpus world is a cave consisting of rooms connected by passageways.

Goal – Agent should pick the Gold present in a Particular room

Agent has only one arrow



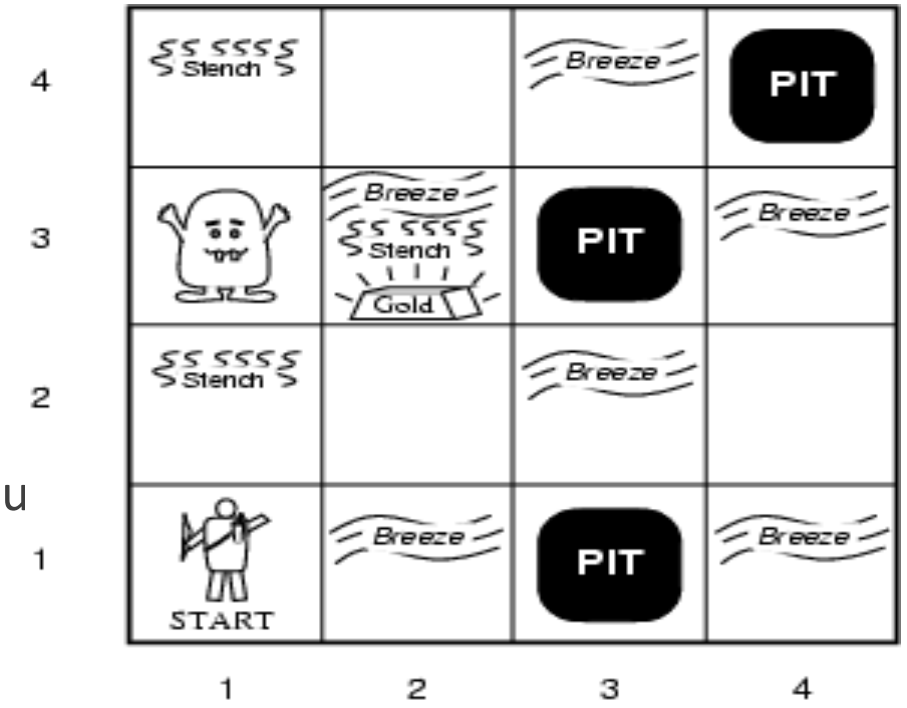
Wumpus World PEAS description

Performance measure
- gold +1000, death -1000
- -1 per step, -10 for using the arrow

Environment: 4 x 4 grid of rooms
- [1,1] start of an Agent

Sensors: Stench, Breeze, Glitter, Bump(Hit wall), Scream (shot Wumpu

Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot



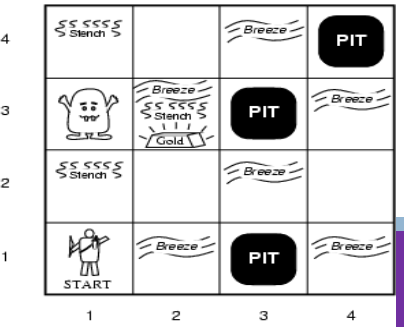
Exploring the Wumpus World

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1	2,1	3,1	4,1

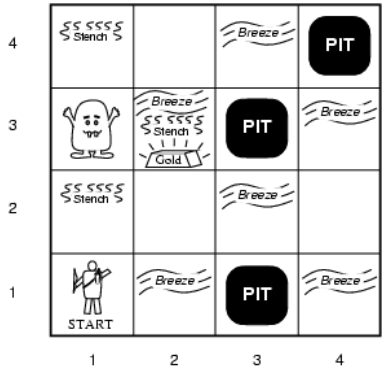
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Percepts given to the agent

1. Stench
2. Breeze
3. Glitter
4. Bumb (ran into a wall)
5. Scream (wumpus has been hit by arrow)



Exploring the Wumpus World



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

Initial situation:

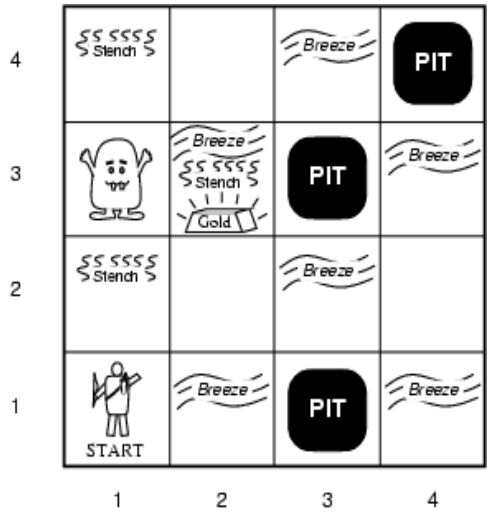
Agent in 1,1 and percept is [None, None, None, None, None]

[Stench,Breeze,Glitter,Bumb,Scream]

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

[None, Breeze, None, None, None]

Exploring the Wumpus World



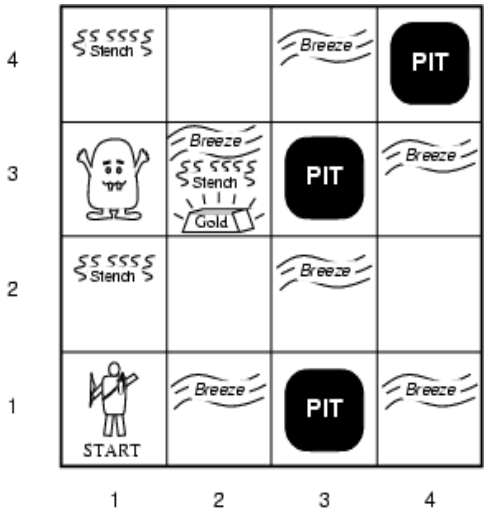
1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

[Stench,None,None,None,None]

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 V OK	2,2 A V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

[None, None, None, None, None]

Exploring the Wumpus World



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

[Stench,Breeze,Glitter,None,None]

Logic

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

- Syntax
- Semantics

E.g., the language of arithmetic

- $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence
- $x+2 \geq y$ is true iff the number $x+2$ is no less than the number y
- $x+2 \geq y$ is true in a world where $x = 7, y = 1$
- $x+2 \geq y$ is false in a world where $x = 0, y = 6$

Syntax

- Syntax is the way sentences are expressed.
- The syntax are the rules that determine whether or not a sentence is well formed.
- Example : name John my is

  my name is John
- 1+=2 3

  1+2=3

Semantics

- Semantics is the meaning behind the sentences.
- In logic, semantics talk about whether a sentence is true or false.
- Example : my name is Paul - False

  my name is John - True

Models

- A model is a version of the environment where variable states take on a specific value, and this helps us evaluate the truthfulness of sentences.
- Example: $x+2 \geq y$ is true in a world where $x = 7, y = 1$
- When a sentence is true under a particular model and we use the word "satisfies"

Models

- Sentence is represented as α
- We use the notation M(α) - set of all models of α

- If a sentence α is true in model m,

  we say that m satisfies α or sometimes m is a model of α.

Entailment

Entailment is a specific kind of relationship between two sentences.

Example

- **Statement A**: "I will turn 28 this year;"
- **Statement B**: "I am currently living."

Entailment is present here because the truth of A requires the truth of B: if I am not currently living, then I cannot age, and therefore I will not turn 28 this year.

Logical Entailment

In mathematical notation,

we write : $\alpha \models \beta$, sentence α entails the sentence β

The formal definition of entailment is this:

$\alpha \models \beta$ if and only if, in every model in which α is true, β is also true.

we can write $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.

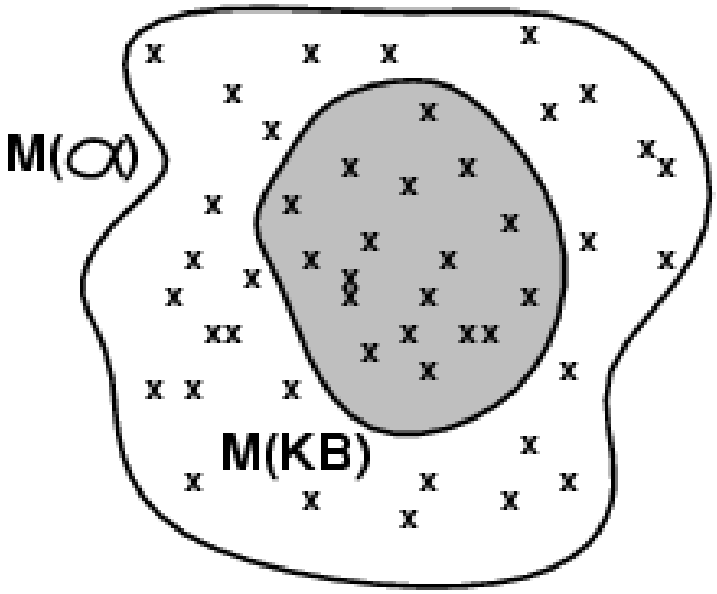
Models and Entailment

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

We say m is a model of a sentence α if α is true in *m*

M(α) is the set of all models of α

Then KB $\models$ α iff M(KB) $\subset$ M(α)



Inference and Entailment

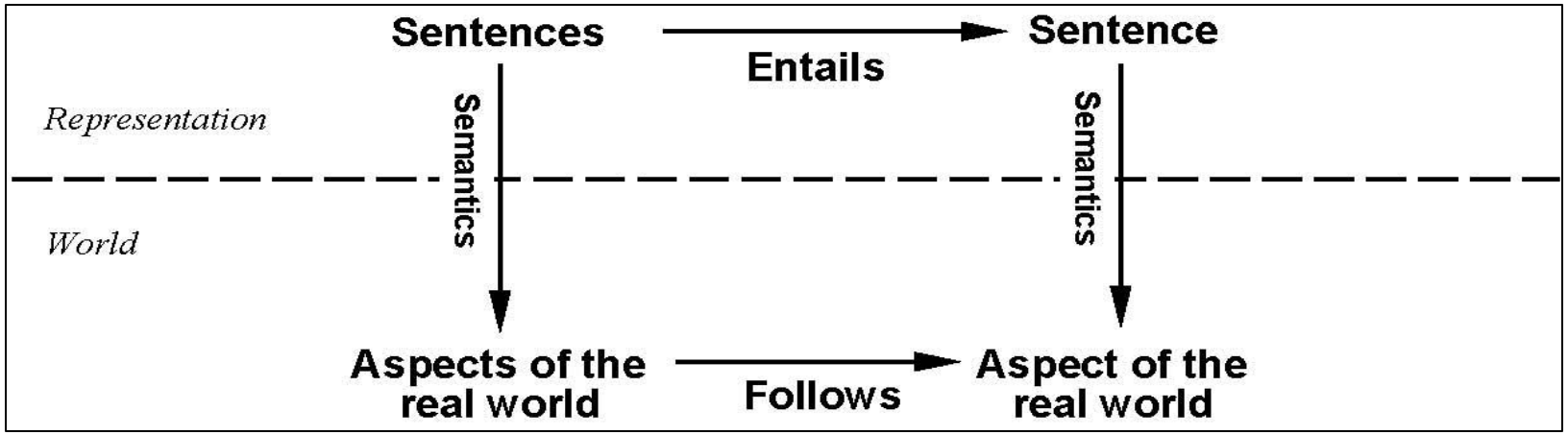
Inference is a procedure that allows new sentences to be derived from a knowledge base.

If an inference algorithm i can derive α from KB, we write $KB \vdash_i \alpha$

Pronounced "α is derived from KB by i" or "i derives α from KB".

Inference and Entailment

- An inference algorithm that derives only entailed sentences is called sound or TRUTH-PRESERVING
- 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.



Propositional logic(Boolean Logic)

- One of the Knowledge representation scheme called propositional logic.
- Syntax and Semantics of Propositional Logic

Propositional logic Syntax

- The Atomic sentences consist of a single proposition symbol.

  Example: P, Q, R, $W_{1,3}$ and North.

- Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.

Type	Connective	Symbol	Notation	Read as
Conjunction	And	∧	$P \wedge Q$	P and Q
Disjunction	Or	∨	$P \vee Q$	P or Q
Conditional	If then	→	$P \rightarrow Q$	P implies Q i.e If P then Q
Bi-conditional	If and only if	↔	$P \leftrightarrow Q$	P double implies Q i.e. P if and only if Q
Negation	Not or No	~ or ¬	$\sim P$	Negation P (or) Not P

Propositional logic Syntax

There are five connectives in common use

Negation: A sentence such as ¬ P is called negation of P.
A literal can be either Positive literal or negative literal.

Conjunction: A sentence which has ∧ connective such as, P ∧ Q is called a conjunction.
Example: Rohan is intelligent and hardworking.
It can be written as,

- P= Rohan is intelligent,
- Q= Rohan is hardworking. → P∧ Q.

Disjunction: A sentence which has ∨ connective, such as P ∨ Q. is called disjunction, where P and Q are the propositions.
Example: "Ritika is a doctor or Engineer",
Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as P ∨ Q.

Propositional logic Syntax

Implication: A sentence such as P → Q, is called an implication.

Implications are also known as if-then rules.

If it is raining, then the street is wet.

Let P= It is raining, and Q= Street is wet, so it is represented as P → Q

Biconditional: A sentence such as P⇔ Q is a Biconditional sentence,

Example If I am breathing, then I am alive

P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Propositional logic Syntax

Figure 7.7 gives a formal grammar of propositional logic with the BNF notation

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

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

Propositional logic Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- All sentences are constructed from atomic sentences and the five connectives

Propositional logic Semantics

- How to compute the truth of atomic sentences

  True is true in every model and False is false in every model

- For complex sentences, we have five rules, which hold for any subsentences P and Q in any model m (here "iff" means "if and only if"):
  - $\neg P$ is true iff P is false in m.
  - $P \wedge Q$ is true iff both P and Q are true in m.
  - $P \vee Q$ is true iff either P or Q is true in m.
  - $P \Rightarrow Q$ is true unless P is true and Q is false in m.
  - $P \Leftrightarrow Q$ is true iff P and Q are both true or both false in m.

Propositional logic Semantics

- The rules can also be expressed with truth tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Propositional logic Semantics

- The rules can also be expressed with truth tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Propositional Theorem Proving

Entailment by theorem proving, so we need additional concepts

- Logical equivalence
- Validity
- Satisfiability

Propositional Theorem Proving

Logical equivalence

- Two sentences α and β are logically equivalent if they are true in the same set of models.

$$\alpha \equiv \beta$$

- For example, $P \wedge Q$ and $Q \wedge P$ are logically equivalent;
- Any two sentences α and β are equivalent only if each of them entails the other: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

Propositional Theorem Proving

Laws of Logic

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

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

Propositional Theorem Proving

Validity

A sentence is valid if it is true in all models.

Example, the sentence P ∨ ¬P is valid.

p	¬p	p ∨ ¬p
T	F	T
F	T	T

Valid sentences are also known as tautologies—they are necessarily true.

α and β, α |= β if and only if the sentence (α ⇒ β) is valid

Propositional Theorem Proving

Satisfiability

A Compound proposition is satisfiable if there is at least one TRUE results in Truth table

Example : P ^ Q

p	q	p^q
T	T	T
T	F	F
F	T	F
F	F	F

The problem of determining the satisfiability of sentences in propositional logic — SAT problem

Propositional Theorem Proving

Inference Rule

AI , we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

Inference : Deriving conclusion from evidences(Premises/Assumptions)

Rules of Inference : Are the templates for constructing valid arguments

Propositional Theorem Proving

Proof is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.

The last sentence is the **theorem** (also called goal or query) that we want to prove.

Example for the "weather problem" given above.

1 Hu	Premise	"It is humid"
2 Hu→Ho	Premise	"If it is humid, it is hot"

Propositional Theorem Proving

Arguments in Propositional Logic

- A argument in propositional logic is a sequence of propositions.
- All the proposition are called premises. The last statement is the conclusion.
- The argument is valid if the premises imply the conclusion.

Propositional Theorem Proving

Types of Inference Rule :

- Modus Ponens
- And-Elimination
- Resolution

Propositional Theorem Proving

Modus Ponens

The Modus Ponens rule states that if α and α → β is true, then we can infer that β will be true.

It can be represented as:

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

Any sentences of the form α ⇒ β and α are given, then the sentence β can be inferred.

Propositional Theorem Proving

And-Elimination

From a conjunction, any of the conjuncts can be inferred

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

Example

"He studies very hard and he is the best boy in the class", P∧Q

Therefore – "He studies very hard" P

Propositional Theorem Proving

All of the logical equivalences in Figure 7.11 can be used as inference rules.

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

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

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	p→q
T	T	T
T	F	F
F	T	T
F	F	T

Determine the validity of the following argument using propositional logic: If there is a cream, then I will drink coffee. If there is a donut , then I will drink coffee. There is no cream and there is a donut. So, I drink coffee

P – If there is a cream

Q- I will drink coffee

R - If there is a donut

Premises $P \rightarrow Q$, $R \rightarrow Q$, $\sim P \wedge R$

Conclusion Q

P	Q	~P	R	P → Q	R → Q	~P ∧ R	(P → Q) ∧ (R → Q) ∧ (~P ∧ R)	(P → Q) ∧ (R → Q) ∧ (~P ∧ R) → Q
T	T	F	T	T	T	F	F	T
T	T	F	F	F	T	F	F	T
T	F	F	T	F	F	F	F	T
T	F	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T	T
F	T	T	F	T	T	F	F	T
F	F	T	T	T	F	T	F	T
F	F	T	F	T	T	F	F	T

This shows that $(P \rightarrow Q) \wedge (\neg P) \nrightarrow \neg Q$ is not a tautology. Hence the conclusion $\neg Q$ is not valid.

in earlier examples.

W.E.11. Determine the validity of the following argument:
If two sides of a triangle are equal, then two opposite angles are equal.
Two sides of a triangle are not equal.
Therefore, the opposite angles are not equal.
Solution
Let us indicate the statements as follows :

P : Two sides of a triangle are equal.
Q : The two opposite angle are equal.

The given argument in the form

$$P \rightarrow Q, \neg P \Rightarrow \neg Q.$$

Let us now construct the truth table for $(P \rightarrow Q) \wedge (\neg P) \rightarrow \neg Q$.

P	Q	$P \rightarrow Q$	$\neg P$	$(P \rightarrow Q) \wedge (\neg P)$	$\neg Q$

W.E.12. Determine the validity of the following argument :
My father praises me only if I can be proud of myself.
Either I do well in sports or I cannot be proud of myself. If I study hard, then I cannot do well in sports. Therefore, if father praises me, then I do not study well.

Solution

Let us indicate the statements as follows :

P : My father praises me.
Q : I can be proud of myself.
R : I do well in sports.
S : I study hard.

The given argument is of the form

$$P \rightarrow Q, R \vee \neg Q, S \rightarrow \neg R \Rightarrow P \rightarrow \neg S.$$

As the desired result is $P \rightarrow \neg S$, assume that P is True.
It is enough to verify the validity of

$$P, P \rightarrow Q, R \vee \neg Q, S \rightarrow \neg R \Rightarrow \neg S.$$

From P and $P \rightarrow Q$, we have Q.
From Q and $R \vee \neg Q$, we have R.
From R and $S \rightarrow \neg R$, we have $\neg S$.
Thus the argument is valid.

Is the following argument valid, or is it a fallacy?
"If you wear an enormous cowboy hat, people will stare. If people stare at you, you get embarrassed. Therefore, if you wear an enormous cowboy hat, you will be embarrassed."

Premises:

$p \to q$: If you wear an enormous cowboy hat, people will stare.

$q \to r$: If people stare at you, you get embarrassed.

Conclusion:

$p \to r$: If you wear an enormous cowboy hat, you will be embarrassed.

Argument:

$$[(p \to q) \wedge (q \to r)] \to (p \to r)$$

p	q	r	$p \to q$	$q \to r$	$p \to r$	$(p \to q) \wedge (q \to r)$	$[(p \to q) \wedge (q \to r)] \to (p \to r)$
T	T	T	T	T	T	T	T
T	F	T	F	T	T	F	T
F	T	T	T	T	T	T	T
F	F	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	F	F	T	F	F	T
F	T	F	T	F	T	F	T
F	F	F	T	T	T	T	T

Is the following argument valid, or is it a fallacy?
"A musician can play the guitar or the piano. She can play the guitar, so she can't play the piano."

Premises:

$p \vee q$: She can play the guitar or she can play the piano.

p: She can play the guitar.

Conclusion:

$\sim q$: She can't play the piano.

Argument:

$$[(p \vee q) \wedge p] \rightarrow \sim q.$$

p	q	$\sim q$	$p \vee q$	$(p \vee q) \wedge p$	$[(p \vee q) \wedge p] \rightarrow \sim q$
T	T	F	T	T	F
T	F	T	T	T	T
F	T	F	T	F	T
F	F	T	F	F	T

Propositional Theorem Proving

Monotonicity

- In monotonicity, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base.

Example:

- “Earth revolves around the Sun”
- It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Propositional Theorem Proving

Monotonicity

- Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base—the conclusion of the rule must follow regardless of what else is in the knowledge base
- Set of entailed sentences can increase as information is added to the knowledge base.
- For any sentences α and β

$$\text{if } KB \models \alpha \text{ then } KB \wedge \beta \models \alpha$$

Propositional Theorem Proving

Literal : Is either a propositional variable P or a negated propositional variable ¬P.

Clause : is
– A literal or
– Disjunction of two or more literals or
– The empty clause
Example: { } , P, P ∨ ¬Q ∨ R

$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots
\end{aligned}
$$

Propositional Theorem Proving

Resolution Method

Resolution method is an inference rule which is used in both propositional logic as well as First order predicate logic

This method is used for proving the satisfiability of a sentence

Propositional Theorem Proving

Resolution Rule

In PL, resolution method is the inference rule takes **two premises in the form of clauses** and gives the clause as a conclusion.

$$\frac{(\alpha \vee \neg\beta) \wedge (\gamma \vee \beta) \text{ premise}}{(\alpha \vee \gamma) \text{ conclusion}}$$

The resolution rule applies only to clauses

Propositional Theorem Proving

Two Normal form

Conjunctive Normal Form (CNF)

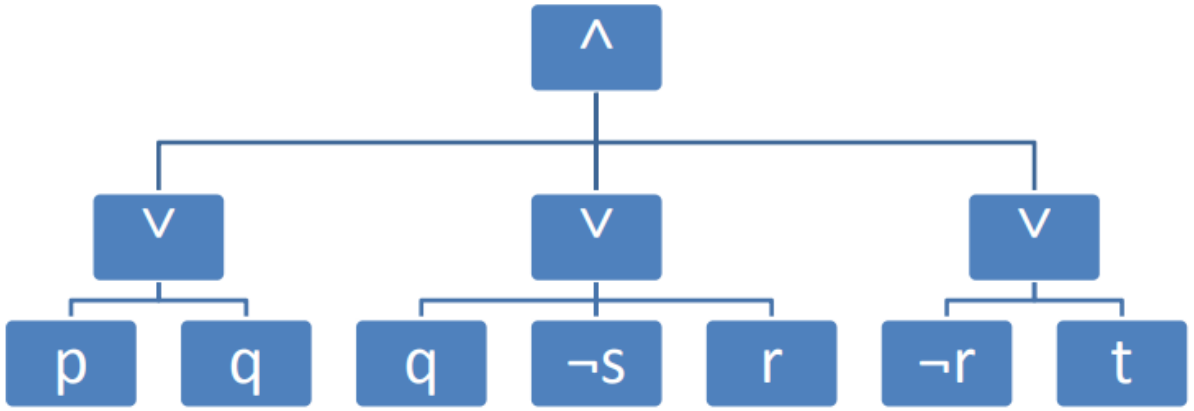
Disjunctive Normal Form (DNF)

Propositional Theorem Proving

Conjunctive Normal Form:

A sentence expressed as a conjunction of clauses is said to be in conjunctive normal form

$$(p \vee q) \wedge (q \vee \neg s \vee r) \wedge (\neg r \vee t)$$



$$\begin{aligned} CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\ Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\ Literal &\rightarrow Symbol \mid \neg Symbol \\ Symbol &\rightarrow P \mid Q \mid R \mid \ldots \end{aligned}$$

Propositional Theorem Proving

Procedure to convert sentence into CNF : Steps

1. Remove Bicondition

   Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

2. Remove Implication

   Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

Propositional Theorem Proving

Procedure to convert sentence into CNF : Steps

3. Move Negation inwards

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

4. Apply Distributive and/or Commutative law

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Propositional Theorem Proving

Example : $A \Leftrightarrow (B \vee C)$

1. Remove $\Leftrightarrow$

   $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

   $[A \Rightarrow (B \vee C)] \wedge [(B \vee C) \Rightarrow A]$

2. Remove Implication

   $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

   $[\neg A \vee (B \vee C)] \wedge [\neg (B \vee C) \vee A]$

Propositional Theorem Proving

[¬ A ∨ (B ∨ C)] ∧ [¬ (B ∨ C) ∨ A]

3. Move Negation inwards

[¬ A ∨ (B ∨ C)] ∧ [(¬ B ∧ ¬ C) ∨ A]

4. Apply Distributive and/or Commutative law

(¬ A ∨ B ∨ C) ∧ (¬ B ∨ A) ∧ (¬ C ∨ A)

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

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

This Sentence is now in CNF, as a conjunction of three clauses.

Propositional Theorem Proving

A resolution algorithm

To show that KB |= α, we show that (KB ∧ ¬α) is unsatisfiable.

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

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

Two important properties for inference

Soundness: If KB |- Q then KB |= Q

- If Q is derived from a set of sentences KB using a given set of rules of inference, then Q is entailed by KB.
- Hence, inference produces only real entailments, or any sentence that follows deductively from the premises is valid.

Completeness: If KB |= Q then KB |- Q

- If Q is entailed by a set of sentences KB, then Q can be derived from KB using the rules of inference.
- Hence, inference produces all entailments, or all valid sentences can be proved from the premises.

Propositional Theorem Proving

Completeness of resolution

Resolution closure RC(S) of a set of clauses S, which is the set of all clauses derivable by repeated application of the resolution rule to clauses in S or their derivatives.

The completeness theorem for resolution in propositional logic is called the ground resolution theorem

Propositional Theorem Proving

Completeness of resolution

Ground resolution theorem : If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Construct a model for S with suitable truth values for P1, . . . , Pk.

The construction procedure is as follows:

For i from 1 to k,

- **If a clause in RC(S) contains the literal ¬Pi and all its other literals are false under** the assignment chosen for P1, . . . , Pi−1, then assign false to Pi.
- **Otherwise, assign true to Pi.**

Propositional Theorem Proving

Horn clause

A clause which is a disjunction of literals of with at most one positive literal.

Example : ¬A V ¬C V D

Propositional Theorem Proving

Types of Horn Clauses :

- **Definite clause / Strict Horn clause**
  It has exactly one positive literal.
  Example: ¬p V ¬q V ... V ¬t V u
- **Unit clause**
  Definite clause with no negative literals.
  Example: u
- **Goal clause**
  Horn clause without a positive literal.
  Example : ¬p V ¬q V ... V ¬t

Propositional Theorem Proving

KB containing only definite clauses are interesting for three reasons:

1. Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
2. Inference with Horn clauses can be done through the forward-chaining and backward chaining algorithms
3. Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base

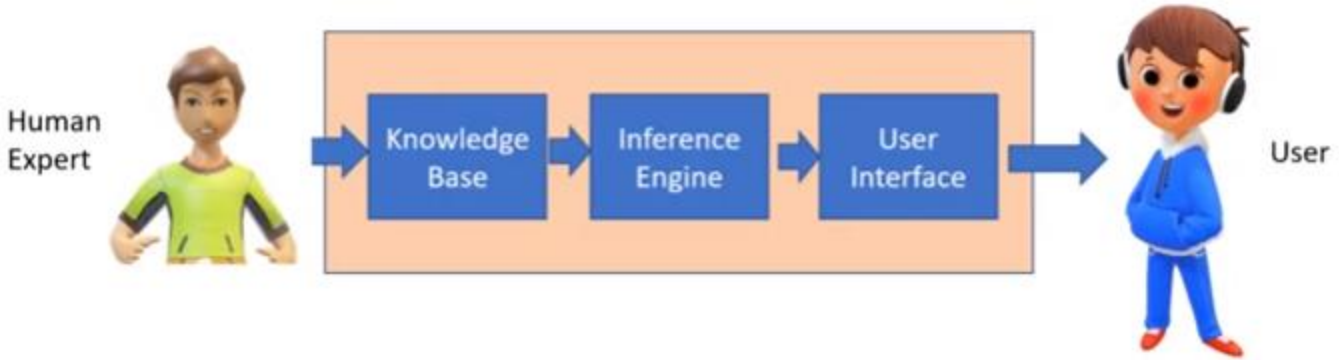
Propositional Theorem Proving

Forward and backward chaining

Inference engine

- It applies logical rules to the knowledge base to infer new information from known facts.
- Inference engine commonly proceeds in two modes:

  1.Forward chaining 2.Backward chaining



Propositional Theorem Proving

Forward chaining

- Forward chaining starts with the available data and uses inference rules to extract more data until a goal is reached.
- **"What will happen next?"**
- It is a bottom up approach.
- Forward chaining is data driven because the reasoning starts from a set of data.

- Example

  It is raining.
  If it is raining, the street is wet.
  The street is wet.

Propositional Theorem Proving

Backward chaining

- Backward chaining starts from goal and proceeds backward to determine the set of rules that match the goal.
- **"Why did this happen?"**
- It is top down approach.
- Backward chaining is goal driven
- Example: diagnose bacterial infections.

Example :

Patient -> Fever (Conclusion)

Temperature/Symptoms(Rules)

viral/bacterial Infection (fact)

- Example:

The street is wet.
If it is raining, the street is wet.
It is raining.

Propositional Theorem Proving

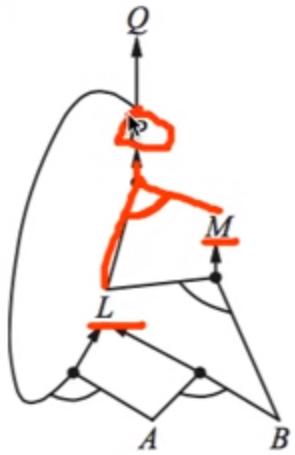
Forward chaining algorithm

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 ← a table, where *count*[c] is the number of symbols in c's premise
  inferred ← a table, where *inferred*[s] is initially *false* for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while *agenda* is not empty **do**
    p ← POP(*agenda*)
    if $p = q$ **then return** *true*
    if *inferred*[p] = *false* **then**
      inferred[p] ← *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*

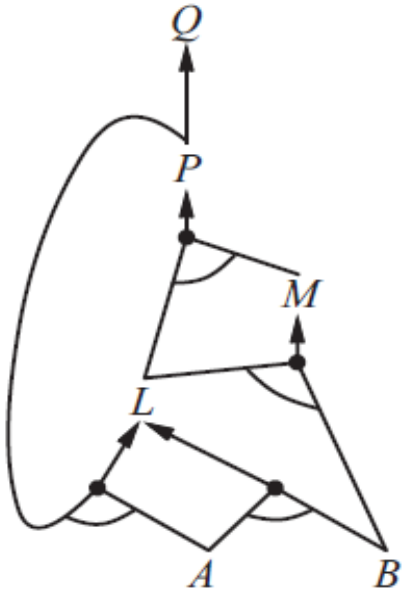
Propositional Theorem Proving

In Forward chaining algorithm knowledge base is drawn as an **AND–OR graph**



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

(a)



(b)

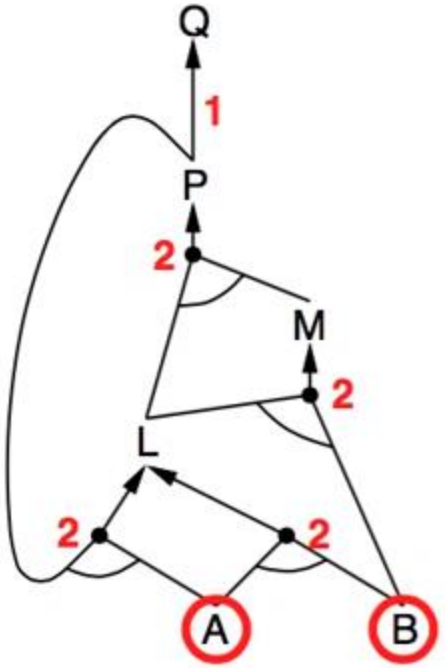


Figure 7.16 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

Outline

First Order logic

- Representation revisited
- Syntax and semantics of first-order logic
- Using first-order logic
- Knowledge engineering in first-order logic

First Order logic

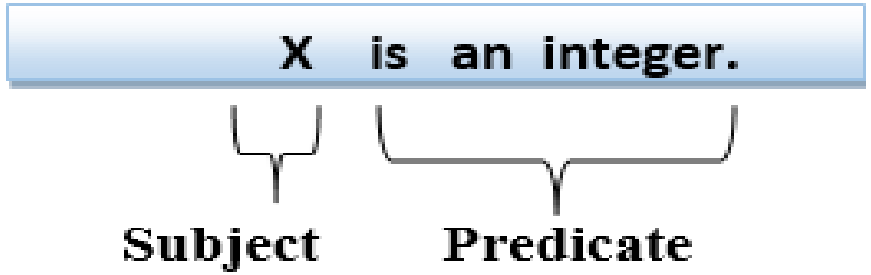
- Propositional logic is too puny a language to represent knowledge of complex environments in a concise way.
- The prepositional logic only deals with the facts, that may be true or false.
- The first order logic assumes that the world contains objects, relations and functions.

First Order logic

First-order logic statements can be divided into two parts:

Subject: Subject is the main part of the statement.

Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement



Integer (x) : x is an integer

Pinky is a cat

cat(x)

First Order logic

- First Order Logic (FOL) is a way of knowledge representation.
- It is an extension of PL
- It is also known as **Predicate Logic**.
- FOL is a collection of objects, their attributes and relations among them to represent knowledge.
- **Objects**: houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- **Relations**: these can be unary relations or properties such as red, round, bogus, prime,multistoried . . ., or more general n-ary relations such as brother of, bigger than, inside,part of, has color, occurred after, owns, comes between, . . .
- **Function**: Father of, best friend, third inning of, end of, ...

First Order logic

Object

1. Richard the Lionheart, King of England from 1189 to 1199;
2. his younger brother, the evil King John, who ruled from 1199 to 1215;
3. the left leg of Richard
4. the left leg of John
5. Crown

A tuple is a collection of objects arranged in a fixed order and is written with angle brackets surrounding the objects.)

<Richard the Lionheart, King John>

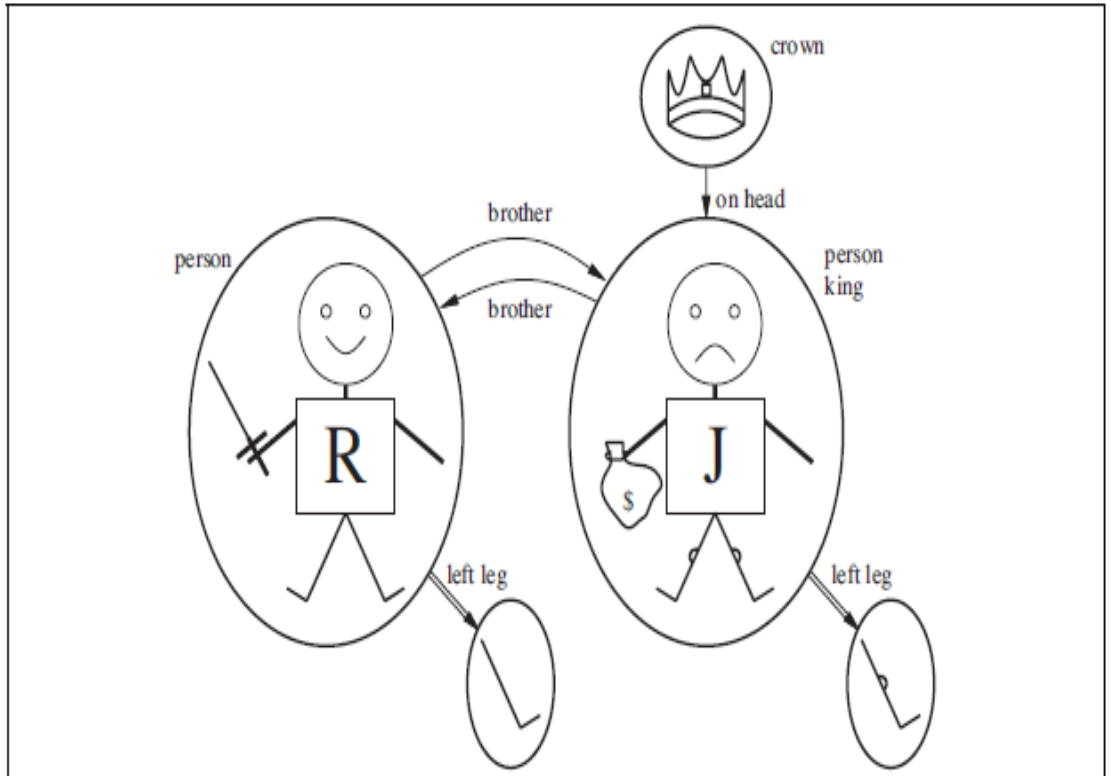


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

First Order logic

- FOL has two main parts
  - Syntax
  - Semantics

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic.

We write statements in short-hand notation in FOL.

First Order logic

Syntax of FOL: Basic elements

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

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

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form

First Order logic

The symbols come in three kinds:

1. Constant symbols which stand for objects
2. Predicate symbols which stand for relations
3. Function symbols which stand for functions

We adopt the convention that these symbols will begin with uppercase letters.

First Order logic

For example,

Constant symbols Richard and John

Predicate symbols Brother , OnHead, Person, King, and Crown

Function symbol LeftLeg.

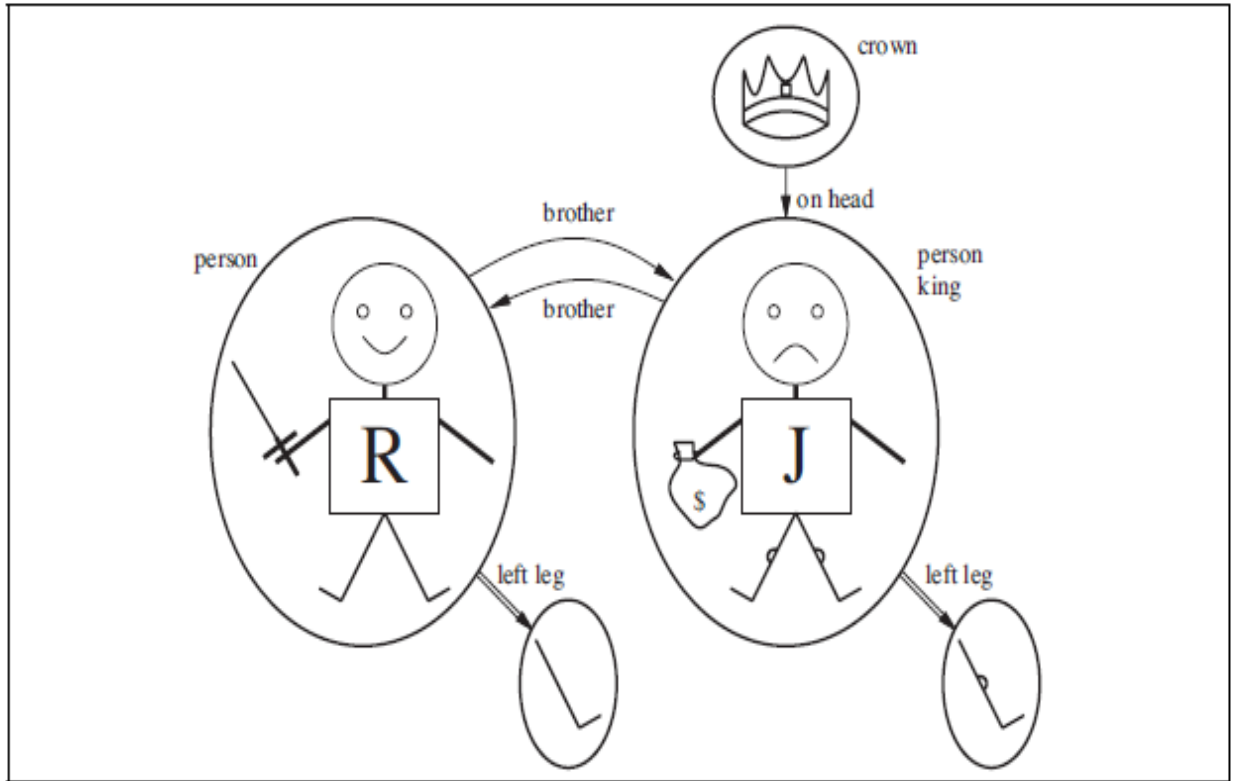


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

First Order logic

Each model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

- Richard refers to Richard the Lionheart and John refers to the evil King John.
- Brother refers to the brotherhood relation
- LeftLeg refers to the “left leg” function

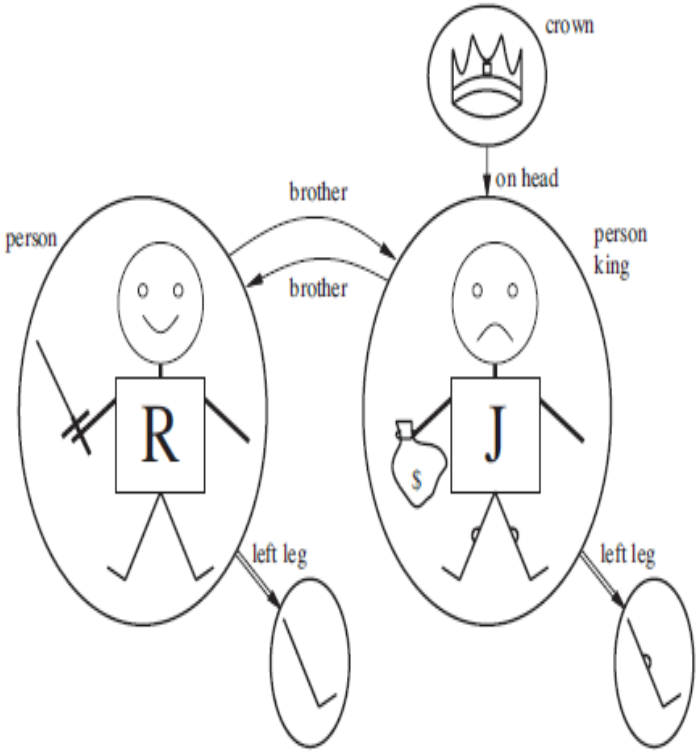


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

First Order logic

Terms

A term is a logical expression that refers to an object.

Constant symbols are therefore terms

A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.

f(t1, . . . , tn), f refers to some function in the model

Example : LeftLeg(John)

First Order logic

Atomic sentences:

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.**

We can represent atomic sentences as **Predicate (term1, term2, ……, term n)**.

Example: Brother (Richard , John)

Complex Sentences:

Complex sentences are made by combining atomic sentences using connectives.

Example : **King(Richard) ∨ King(John)**

First Order logic

Quantifiers in First-order logic

A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.

These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression.

There are two types of quantifier:

- **Universal Quantifier(for all, everyone, everything)**
- **Existential quantifier(for some, at least one).**

First Order logic

Universal Quantifier

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

The main connective for universal quantifier ∀ is implication →.

Example : “All kings are persons” $\forall x \quad King(x) \Rightarrow Person(x)$

For all x, if x is a king, then x is a person.

First Order logic

Universal Quantifier

Example : “All kings are persons” $\forall x \quad King(x) \Rightarrow Person(x)$

For all x, if x is a king, then x is a person.

The symbol x is called a variable.

By convention, variables are lowercase letters.

A variable is a term all by itself, and as such can also serve as the argument of a function

First Order logic

Existential Quantifier

Existential quantifiers express that the statement within its scope is true for at least one instance of something.

- It is denoted by the logical operator ∃, which resembles as inverted E.
- When it is used with a predicate variable then it is called as an existential quantifier.
- The main connective for existential quantifier ∃ is and ∧.
- Example : King John has a crown on his head

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

∃x is pronounced “There exists an x such that . . .” or “For some x . . .”.

First Order logic

Nested quantifiers

- We will often want to express more complex sentences using multiple quantifiers.
- written ∃1 or ∃!, that means "There exists exactly one"

For example, "Brothers are siblings"

$$\forall x \forall y \text{ Brother } (x, y) \Rightarrow \text{Sibling}(x, y)$$

First Order logic

Equality

First-order logic includes one more way to make atomic sentences

Equality symbol signify that two terms refer to the same object.

It can also be used with negation to insist that two terms are not the same object.

For example,

Father (John)=Henry

Using First Order logic

Assertions and queries in first-order logic

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called **assertions.**

For example, we can assert that

John is a king, Richard is a person, and all kings are persons

- TELL(KB, King(John)) .
- TELL(KB, Person(Richard)) .
- TELL(KB, $\forall$ x King(x) $\Rightarrow$ Person(x))

Using First Order logic

Assertions and queries in first-order logic

We can ask questions of the knowledge base using ASK.

Questions asked with ASK are called **queries or goals.**

For example, ASK(KB, King(John))

Example: Write statement in PL

1. Marcus was a man

   man(marcus)

2. X is an integer

   integer(x)

3. Caesor was a rular

   rular(caesor)

Example: Write statement in PL

4.Ravi and Rama are brothers

brothers(ravi,rama)

5. Marcus tried to assassinate Caesar

trytoassassinate(marcus,Caesar)

Example: Write statement in PL

1. All man drink coffee

$\forall$ x : man(x) $\rightarrow$ drink (x, cricket)

2. Every may respects his parent

$\forall$ x : man(x) $\rightarrow$ respects (x, parent)

3. Some boys play cricket

$\exists$ x : boys(x) $\wedge$ play (x, cricket)

Example : Write statement in PL

4.Not all students like both maths and science

$\sim \forall x : students(x) \rightarrow like(x, maths) \wedge like(x, science)$

5.Everyone is loyal to Someone

$\forall x \exists y \; loyal(x, y)$

6. Everyone loves everyone

$\forall x \exists y \; loves(x, y)$

Example : Write statement in PL

- Universal Quantifier

7. “All boys like cricket”

$\forall x : boys(x) \rightarrow like(x, cricket)$

- Existential Quantifier

8. “Some boys like cricket”

$\exists x : boys(x) \wedge like(x, cricket)$

Using First Order logic

In knowledge representation,

A domain is some part of the world about which we wish to express some knowledge.

Domains of

- Family relationships(The kinship domain)
- Wumpus world

Using First Order logic

Kinship domain(family relationship)

It consists of

- Object – People
- Unary Predicates - Male and Female
- Binary Predicates - Parent,Brother,Sister
- Functions – Father, Mother
- Relations – Brotherhood, sisterhood.

Using First Order logic

The kinship domain

- An example KB includes things like:
  - Fact:
    - "Elizabeth is the mother of Charles"
    - "Charles is the father of William"
  - Rules:
    - One's grandmother is the mother of one's parent"
- Object: people
- Unary predicate: Male, Female
- Binary predicate: Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, and Uncle
- Function: Mother, Father

Using First Order logic

Examples

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Rightarrow Sibling(x,y)$

- Male and female are disjoint categories

  $\forall x,\ Male(x) \Leftrightarrow \neg Female(x)$

- Parent and child are inverse relations

  $\forall p,c\ Parent(p,c) \Leftrightarrow Child(c,p)$

Using First Order logic

The Wumpus World

- A percept is a binary predicate:
  - Percept([Stench, Breeze, Glitter, None, None], 5)
- Actions are logical terms:
  - Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb
- Query:
  - ∃a BestAction(a, 5)
  - Answer: {a/Grab}
- Perception: percept implies facts:
  - ∀ t, s, b, m, c Percept([s, b, Glitter, m, c], t) ⇒ Glitter(t)
- Reflex behavior:
  - ∀ t Glitter(t) ⇒ BestAction(Grab, t)

Using First Order logic

The Wumpus World

- The Environment:
  - Objects: squares, pits and the wumpus
  - $\forall x,y,a,b$ *Adjacent*([x,y],[a,b]) $\Leftrightarrow$

    [a,b] $\in$ {[x+1,y], [x-1,y],[x,y+1],[x,y-1]}
  - A unary predicate *Pit*(x)
  - $\forall s,t$ *At*(Agent,s,t) $\wedge$ *Breeze*(t) $\Rightarrow$ *Breezy*(s), the agent infers properties of the square from properties of its current percept
  - Breezy() has no time argument
  - Having discovered which places are breezy or smelly, not breezy or not smelly, the agent can deduce where the pits are and where the wumpus is

Knowledge engineering in first-order logic

The knowledge-engineering Process

- AI goal is to imitate the intelligence of a human being.
- Human uses his/ her intelligence based on his/ her knowledge to make decisions.
- Example: Automate the teaching process for children in the subject of mathematics

- Knowledge engineering is a field of study where we do the engineering of all thought processes for specific domains.

Knowledge engineering in first-order logic

The knowledge-engineering Process

- The general process of constructing knowledge base is called knowledge engineering
- A knowledge engineer investigates a
  - Particular domain
  - learns what concepts are important in that domain
  - creates a formal representation of the objects and relations in the domain

Knowledge engineering in first-order logic

Steps in the knowledge-engineering process

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

Knowledge engineering in first-order logic

Steps in the knowledge-engineering process

1.Identify the task

- Goal formulation step

e.g. In crime mystery solution , the task

- Find the motive
- Find all the suspects
- List all victims
- Find out all the lies the culprit has stated

Knowledge engineering in first-order logic

Steps in the knowledge-engineering process

2.Assemble the relevant knowledge

- Collect common sense knowledge of model
- E.g. in crime mystery story
  - what are typical crime motives,
  - how the crime stories are solved,
  - how the world around the victims and culprits work etc.

Knowledge engineering in first-order logic

Steps in the knowledge-engineering process

3. Decide on a vocabulary of predicates, functions, and constants

E.g.

- person(name, age, sex, occupation)
- had_affair(name, name)
- killed_with(name, object)
- killed(name)
- killer(name)
- motive(vice)
- smeared_in(name, substance)
- owns(name, object)
- operates_identically(object, object)
- owns_probably(name, object)
- suspect(name)

Knowledge engineering in first-order logic

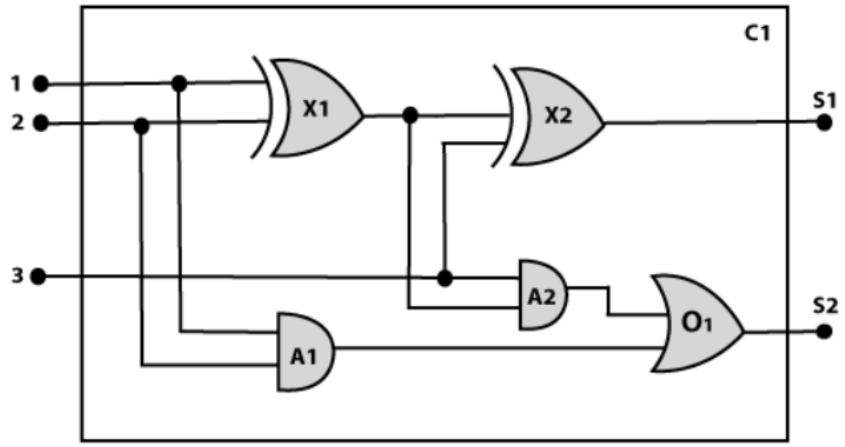
Steps in the knowledge-engineering process

4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

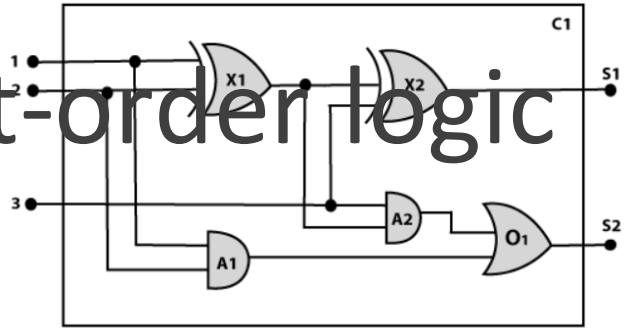
Knowledge engineering in first-order logic

Steps in the knowledge-engineering process

Develop a knowledge base which will allow us to reason about digital circuit (One-bit full adder) which is given below



Knowledge engineering in first-order logic



Steps in the knowledge-engineering process

1. Identify the task

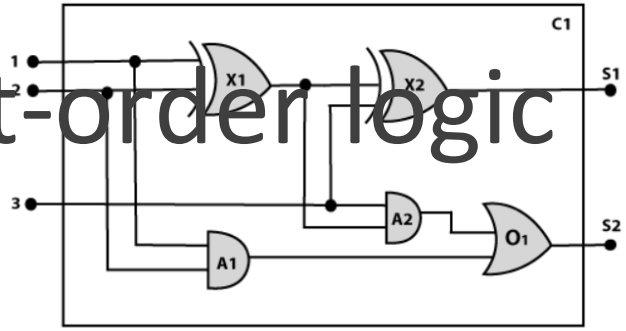
At the first level,we will examine the functionality of the circuit:

Does the circuit add properly?
What will be the output of gate A2, if all the inputs are high?

At the second level, we will examine the circuit structure details such as:

Which gate is connected to the first input terminal?
Does the circuit have feedback loops?

Knowledge engineering in first-order logic



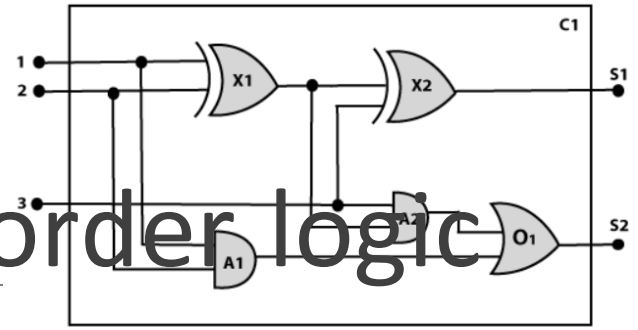
Steps in the knowledge-engineering process
2.Assemble the relevant knowledge

We will assemble the relevant knowledge which is required for digital circuits.

So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.
- In this logic circuit, there are four types of gates used: AND, OR, XOR, and NOT.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

Knowledge engineering in first-order logic



3.Decide on a vocabulary of predicates, functions, and constants

Select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.

Each gate is represented as an object which is named by a constant, **Gate(X1)**.

Circuits will be identified by a predicate: **Circuit (C1)**
Terminal(x)

Objects two signal values 1 and 0, and a function Signal(t) that denotes the signal value for the terminal t.

The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**.

For gate input, we will use the function
In(1, X1) - first input terminal of the gate
Out (1, X1) - output terminal

The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

Knowledge engineering in first-order logic

4.Encode general knowledge about the domain

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

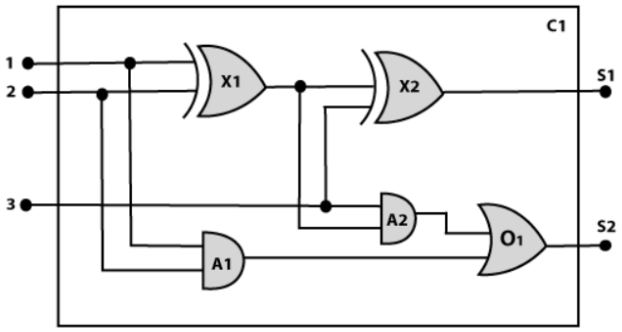
$$\forall\ t1, t2\ \text{Terminal}(t1) \wedge \text{Terminal}(t2) \wedge \text{Connect}(t1, t2) \rightarrow \text{Signal}(t1) = \text{Signal}(2).$$

- Signal at every terminal will have either value 0 or 1, it will be represented as:

$$\forall\ t\ \text{Terminal}(t) \rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0.$$

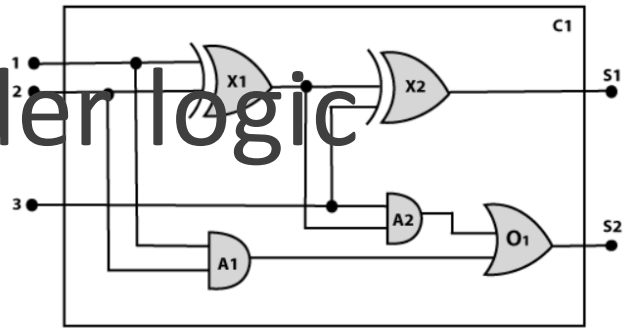
- Connect predicates are commutative:

$$\forall\ t1, t2\ \text{Connect}(t1, t2) \rightarrow \text{Connect}(t2, t1).$$



Knowledge engineering in first-order logic

5.Encode a description of the specific problem insta



This step write simple atomics sentences of instances of concepts

For the given circuit C1, we can encode the problem instance in atomic sentences :

In circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

For XOR gate: Type(x1)= XOR, Type(X2) = XOR

For AND gate: Type(A1) = AND, Type(A2)= AND

For OR gate: Type (O1) = OR.

Knowledge engineering in first-order logic

6.Pose queries to the inference procedure and get answers

Find all the possible set of values of all the terminal for the adder circuit.

The first query will be: What should be the combination of input which would generate the first output of circuit C1 as 0 and a second output to be 1?

$$\exists\, i1, i2, i3\ \text{Signal}(\text{In}(1, C1)) = i1 \wedge \text{Signal}(\text{In}(2, C1)) = i2 \wedge \text{Signal}(\text{In}(3, C1)) = i3$$
$$\wedge\ \text{Signal}(\text{Out}(1, C1)) = 0 \wedge \text{Signal}(\text{Out}(2, C1)) = 1$$

Knowledge engineering in first-order logic

7. Debug the knowledge base

In this step, we will try to debug the issues of knowledge base.

Outline

Inference in First Order logic

- Propositional vs. First-order inference
- Unification
- Forward chaining
- Backward chaining
- Resolution

Propositional vs. First-order inference

Ontology	Facts (P, Q)	Objects, Properties, Relations
Syntax	Atomic sentences Connectives	Variables & quantification Sentences have structure: terms father-of(mother-of(X)))
Semantics	Truth Tables	Interpretations (Much more complicated)
Inference Algorithm	DPLL, GSAT Fast in practice	Unification Forward, Backward chaining Prolog, theorem proving
Complexity	NP-Complete	Semi-decidable

Propositional vs. First-order inference

Substitutions

SUBST(θ,α) denote the result of applying the substitution θ to the sentence α.

Propositional vs. First-order inference

Inference rules for quantifiers

- Universal Generalization
- Universal Instantiation
- Existential Instantiation
- Existential introduction

Propositional vs. First-order inference

Universal Generalization

- It 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).

$$\frac{P(c)}{\forall x\, P(x)}$$

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.

Propositional vs. First-order inference

Universal Instantiation/elimination

It is a valid inference rule, It can be applied multiple times to add new sentences. As per UI, we can infer any sentence obtained by substituting a ground term for the variable.

The UI 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: $\frac{\forall x\, P(x)}{P(c)}$

Propositional vs. First-order inference

Universal Instantiation/elimination

It can be represented as:

$$\frac{\forall x\ P(x)}{P(c)}$$

Example 1:

IF "Every person like ice-cream"=> ∀x P(x)

so we can infer that "John likes ice-cream" => P(c)

Propositional vs. First-order inference

Universal Instantiation/elimination

$$\frac{\forall x\, P(x)}{P(c)}$$

Example 2:

"All kings who are greedy are Evil“

$\forall x$ king(x) $\wedge$ greedy (x) $\rightarrow$ Evil (x)

we can infer any of the following statements using Universal Instantiation:

King(John) $\wedge$ Greedy (John) $\rightarrow$ Evil (John)

King(Richard) $\wedge$ Greedy (Richard) $\rightarrow$ Evil (Richard)

Propositional vs. First-order inference

Existential Instantiation/Elimination

- It is a valid inference rule in first-order logic, It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- 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:

$$\frac{\exists x\, P(x)}{P(c)}$$

Propositional vs. First-order inference

Existential Instantiation/Elimination

$$\frac{\exists x\, P(x)}{P(c)}$$

Example

- From the given sentence: ∃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 base.
- The above used K is a constant symbol, which is called Skolem constant.
- The Existential instantiation is a special case of Skolemization process.

Propositional vs. First-order inference

Existential introduction/generalization

- It 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: $$\frac{P(c)}{\exists x P(x)}$$

Propositional vs. First-order inference

Existential introduction

$$\frac{P(c)}{\exists x P(x)}$$

Example

"Pinky got good marks in English.“ c= pinky
"Therefore, someone got good marks in English."

Substitution

A substitution is a Finite set {t1/v1, · · · ,tn/vn}
where vi is a variable
ti is a term, different from vi ,
and no two elements in the set have the same variable after the / symbol.

$\{f(z)/x, y/z\}$ is a substitution

$\{a/x, g(y)/y, f(g(b))/z\}$ is a substitution

$\{y/x, g(b)/y\}$ is a substitution

$\{a/x, g(y)/x, f(g(b))/z\}$ is not a substitution

$\{g(y)/x, z/f(g(b))\}$ is not a substitution

Unification

- Unification means making expression looks identical.
- Can be done with the process of Substitution.

Example : p(x, F(y)) -------- 1

p(x, F(g(z)) --------- 2

1 and 2 are identical if x is replace with a

y is replaced with g(z)

[a/x , g(z)/y] –substitution set

p(a, F(g(z))

Unification

Example : p(x, F(y)) -------- 1

p(x, F(g(z)) --------- 2

Unification conditions

1. Predicate symbol must be same
2. Number of arguments in both expressions must be identical
3. If two similar variables present in same expression, then unification fails

Unification

Example : Q(a, g(x,a), f(y)) -------- 1

Q(a, g(f(b) ,a),x) --------- 2

- Substitute f(b)/x

Q(a, g(f(b),a), f(y)) Q(a, g(f(b) ,a), f(b))

- Substitute b/y

Q(a, g(f(b),a), f(b)) Q(a, g(f(b) ,a), f(b))

Unification

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound expression
    y, a variable, constant, list, or compound expression
    θ, the substitution built up so far (optional, defaults to empty)

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
  else return failure

---

function UNIFY-VAR(var, x, θ) **returns** a substitution

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

Forward chaining

Start from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.

The process repeats until the query is answered or no new facts are added.

- Forward chaining starts with the available data and uses inference rules to extract more data until a goal is reached.
- **"What will happen next?"**
- It is a bottom up approach.
- Forward chaining is data driven because the reasoning starts from a set of data.

Forward chaining

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: KB, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence
  local variables: *new*, the new sentences inferred on each iteration

  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

Example: Write statement in PL

1. Sky is blue

   sky → blue

2. Knife is a weapon

   knife → weapon

3. John is a king

   king(john)

Example :Forward chaining

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

Prove that **”West is criminal."**

Forward chaining

Convert all the facts into first-order definite clauses, and then use a forward-chaining algorithm to reach the goal.

There are some facts give here:

- It is a crime for an American to sell weapons to an enemy of America
- Country A is the enemy of America
- A colonel of the army sells missiles to A
- Missiles are weapons
- The colonel is an American citizen

Forward chaining

"... it is a crime for an American to sell weapons to hostile nations":

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x) .$$

Forward chaining

Nono has some missiles

There exist something in Universe, that NONO owns, and That something is missile.

"Nono . . . has some missiles." The sentence $\exists x\ Owns(Nono, x) \wedge Missile(x)$ is transformed into two definite clauses by Existential Instantiation, introducing a new constant M_1:

$$Owns(Nono, M_1)$$
$$Missile(M_1)$$

Forward chaining

"All of its missiles were sold to it by Colonel West":

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) .$$

We will also need to know that missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

Forward chaining

and we must know that an enemy of America counts as "hostile":

$$Enemy(x, America) \Rightarrow Hostile(x) .$$

"West, who is American ...":

$$American(West) .$$

"The country Nono, an enemy of America ...":

$$Enemy(Nono, America) .$$

Forward chaining

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x) .$$

$$Owns(Nono, M_1)$$

$$Missile(M_1)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) .$$

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x) .$$

$$American(West) .$$

$$Enemy(Nono, America) .$$

Forward chaining

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x) .$$

$$Owns(Nono, M_1)$$

$$Missile(M_1)$$

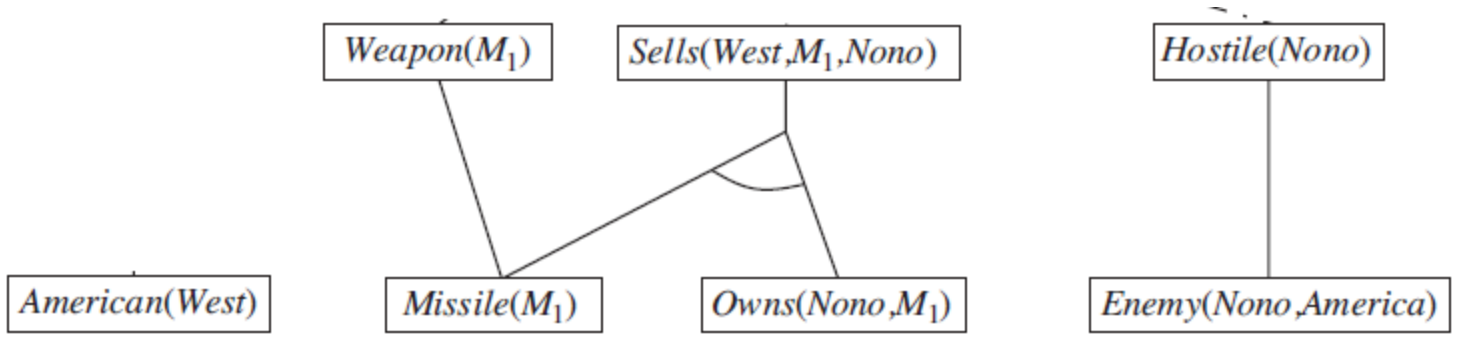
$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) .$$

$$Missile(x) \Rightarrow Weapon(x)$$

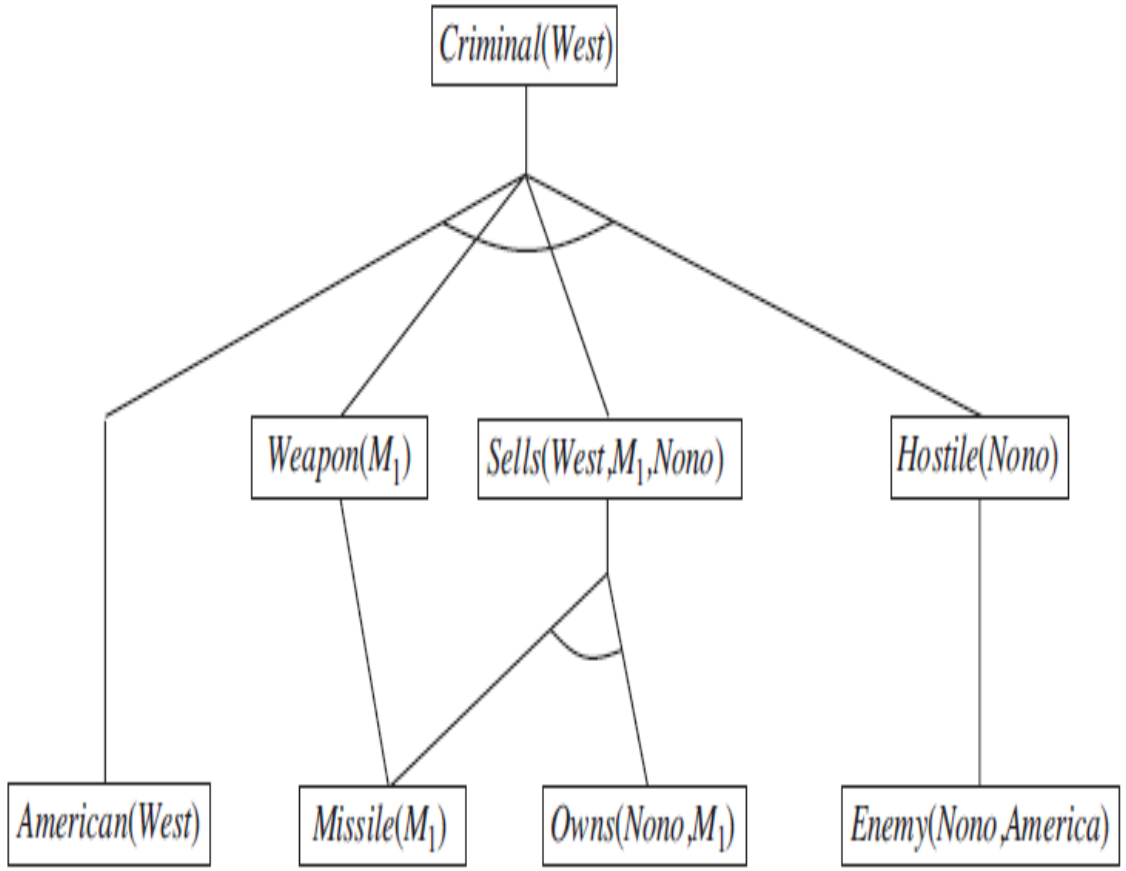
$$Enemy(x, America) \Rightarrow Hostile(x) .$$

$$American(West) .$$

$$Enemy(Nono, America) .$$



Forward chaining



$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x) .$$

$$Owns(Nono, M_1)$$

$$Missile(M_1)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) .$$

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x) .$$

$$American(West) .$$

$$Enemy(Nono, America) .$$

Properties of Forward chaining

Sound and complete for first-order definite clauses

Datalog = first-order definite clauses + no functions (e.g., crime KB)

Forward chaining

Consider the following sentences:

1.John likes all kinds of food

2.Apples are food

3.Chicken is food

4.Anything anyone eats and isn't killed by is food.

5.Bill eats peanut and is still alive

6. sue eats everything bill eats

(i) Translate these sentences into formulas in predicate logic.

(ii) Prove that john likes peanuts using forward chaining

Forward chaining

John likes all kinds of food

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

Apples are food

Apples(x) → food(x)

Chicken is food

Chicken(x) → food(x)

Forward chaining

Anything anyone eats and isn't killed by is food

$\forall x \forall y \text{ eats } (x, y) \wedge \sim \text{killed } (y) \rightarrow \text{ food } (x)$

Bill eats peanut and is still alive

$\text{eats (bill, peanut)} \wedge \sim \text{killed (bill)}$

sue eats everything bill eats

$\forall x \text{ eats (bill, x)} \rightarrow \text{ eat (sue, x)}$

Forward chaining

∀ x : food(x) → likes (John, x)

Apples(x) → food(x) food(Apple)

Chicken(x) → food(x) food(Chicken)

∀ x ∀ y eats (x, y) ∧ ~ killed (y) → food (x)

eats (bill, peanut) ∧ ~ killed (bill)

∀ x eats (bill, x) → eat (sue, x)

eats (bill, peanut)

~ killed (bill)

Forward chaining

∀ x : food(x) → likes (John, x)

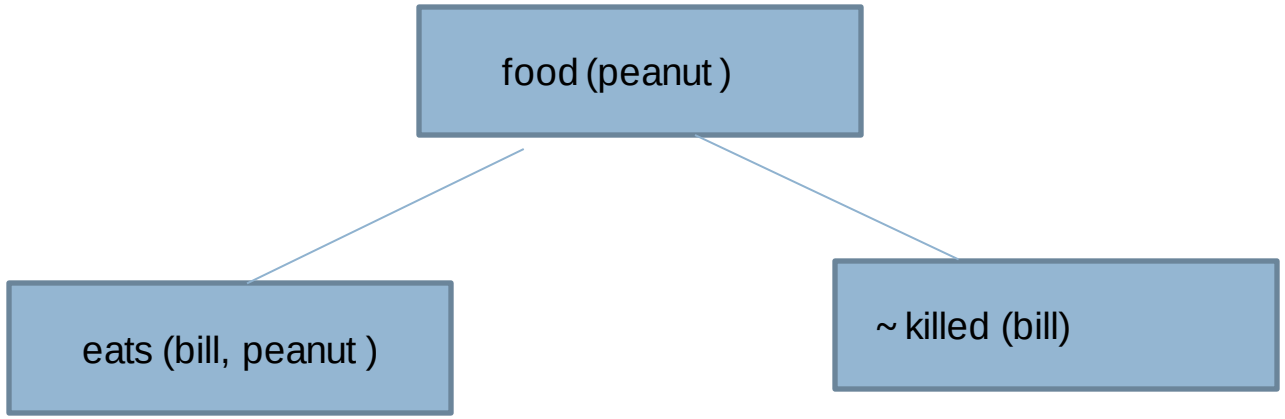
Apples(x) → food(x)

Chicken(x) → food(x)

∀ x ∀ y eats (x, y) ∧ ~ killed (y) → food (x)

eats (bill, peanut) ∧ ~ killed (bill)

∀ x eats (bill, x) → eat (sue, x)



Forward chaining

∀ x : food(x) → likes (John, x)

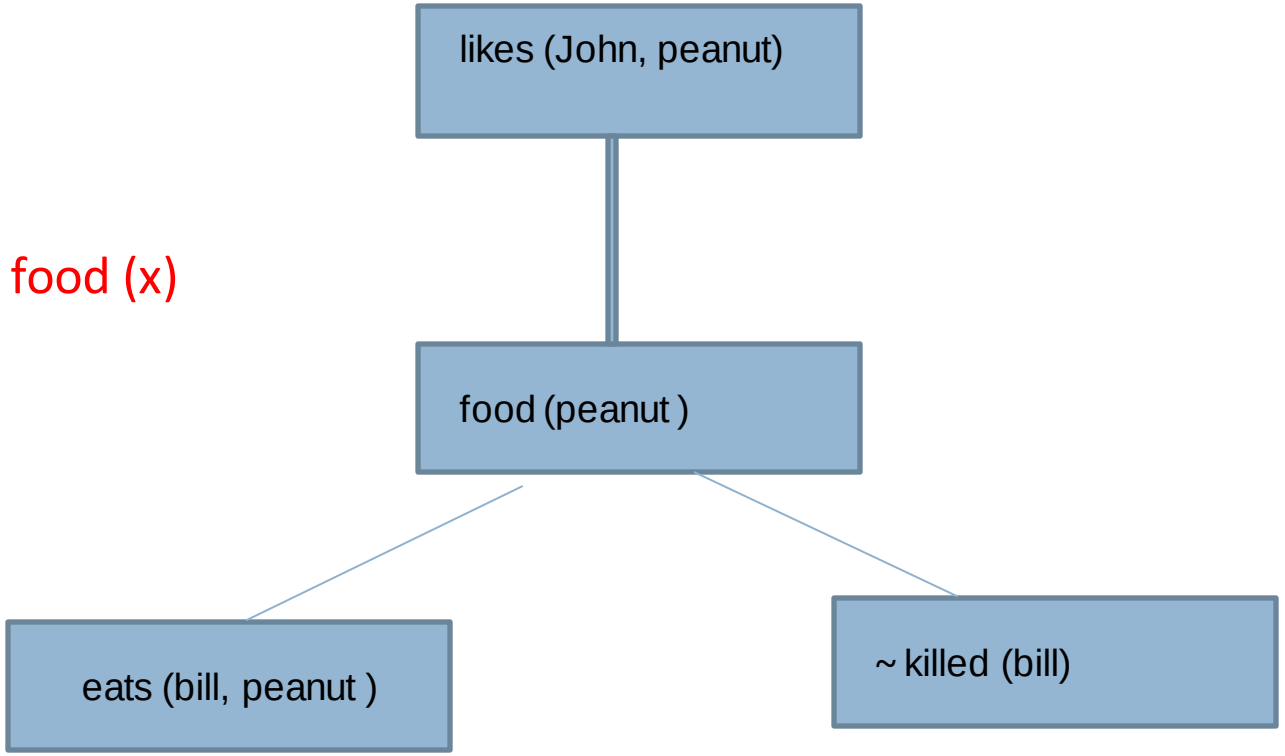
Apples(x) → food(x)

Chicken(x) → food(x)

∀ x ∀ y eats (x, y) ∧ ~ killed (y) → food (x)

eats (bill, peanut) ∧ ~ killed (bill)

∀ x eats (bill, x) → eat (sue, x)



Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: KB, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence
  local variables: *new*, the new sentences inferred on each iteration

  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

Backward-chaining algorithm

function FOL-BC-ASK(KB, $query$) **returns** a generator of substitutions
  return FOL-BC-OR(KB, $query$, $\{\}$)

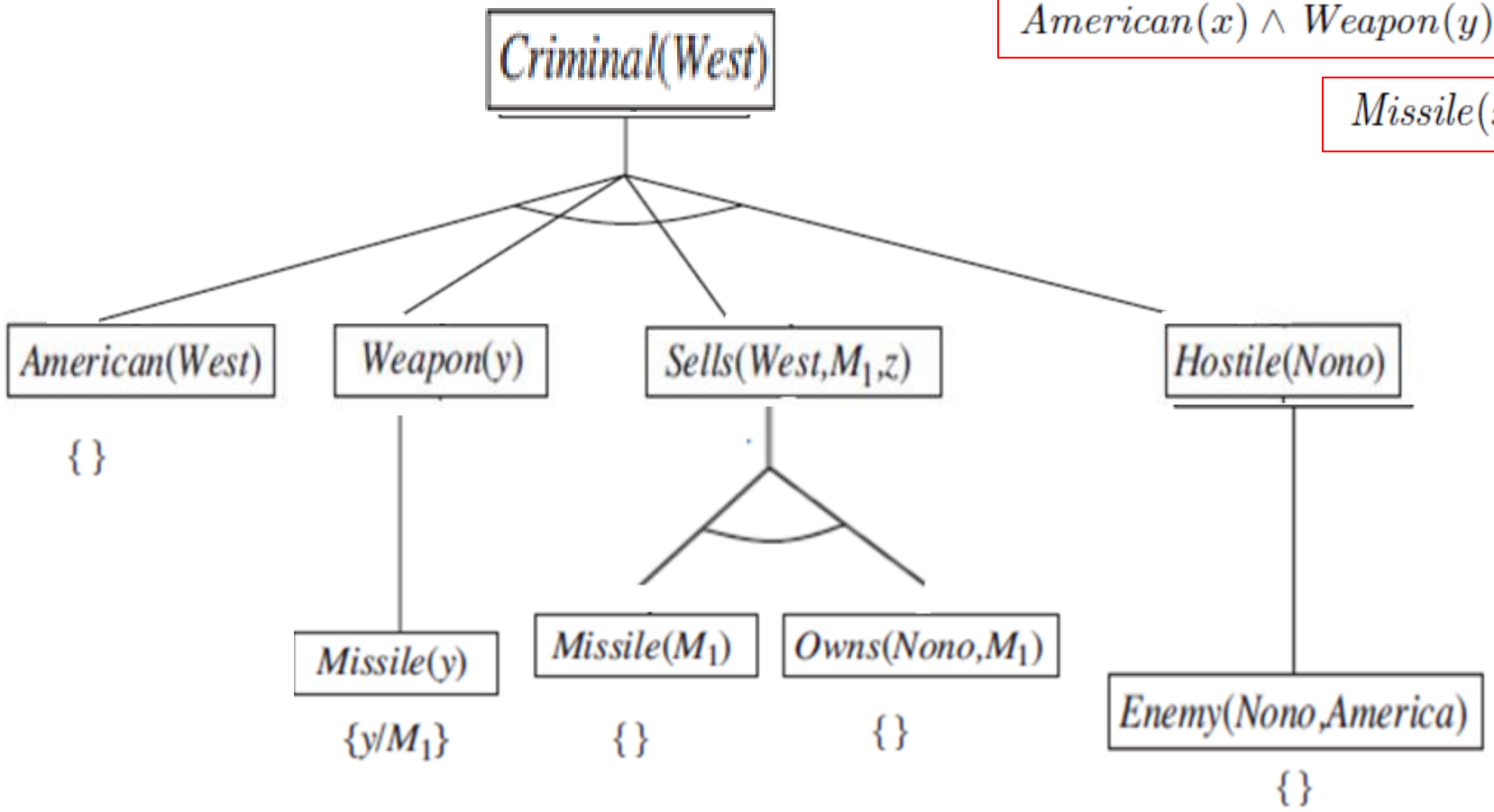
---

generator FOL-BC-OR(KB, $goal$, θ) **yields** a substitution
  for each rule ($lhs \Rightarrow rhs$) in FETCH-RULES-FOR-GOAL(KB, $goal$) **do**
    (lhs, rhs) $\leftarrow$ STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ' **in** FOL-BC-AND(KB, lhs, UNIFY(rhs, $goal$, θ)) **do**
      yield θ'

---

generator FOL-BC-AND(KB, $goals$, θ) **yields** a substitution
  if $\theta = failure$ **then return**
  else if LENGTH($goals$) = 0 **then yield** θ
  else do
    $first$, $rest$ $\leftarrow$ FIRST($goals$), REST($goals$)
    for each θ' **in** FOL-BC-OR(KB, SUBST(θ, $first$), θ) **do**
      for each θ'' **in** FOL-BC-AND(KB, $rest$, θ') **do**
        yield θ''

Backward chaining



$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x) .$$

$$Owns(Nono, M_1)$$

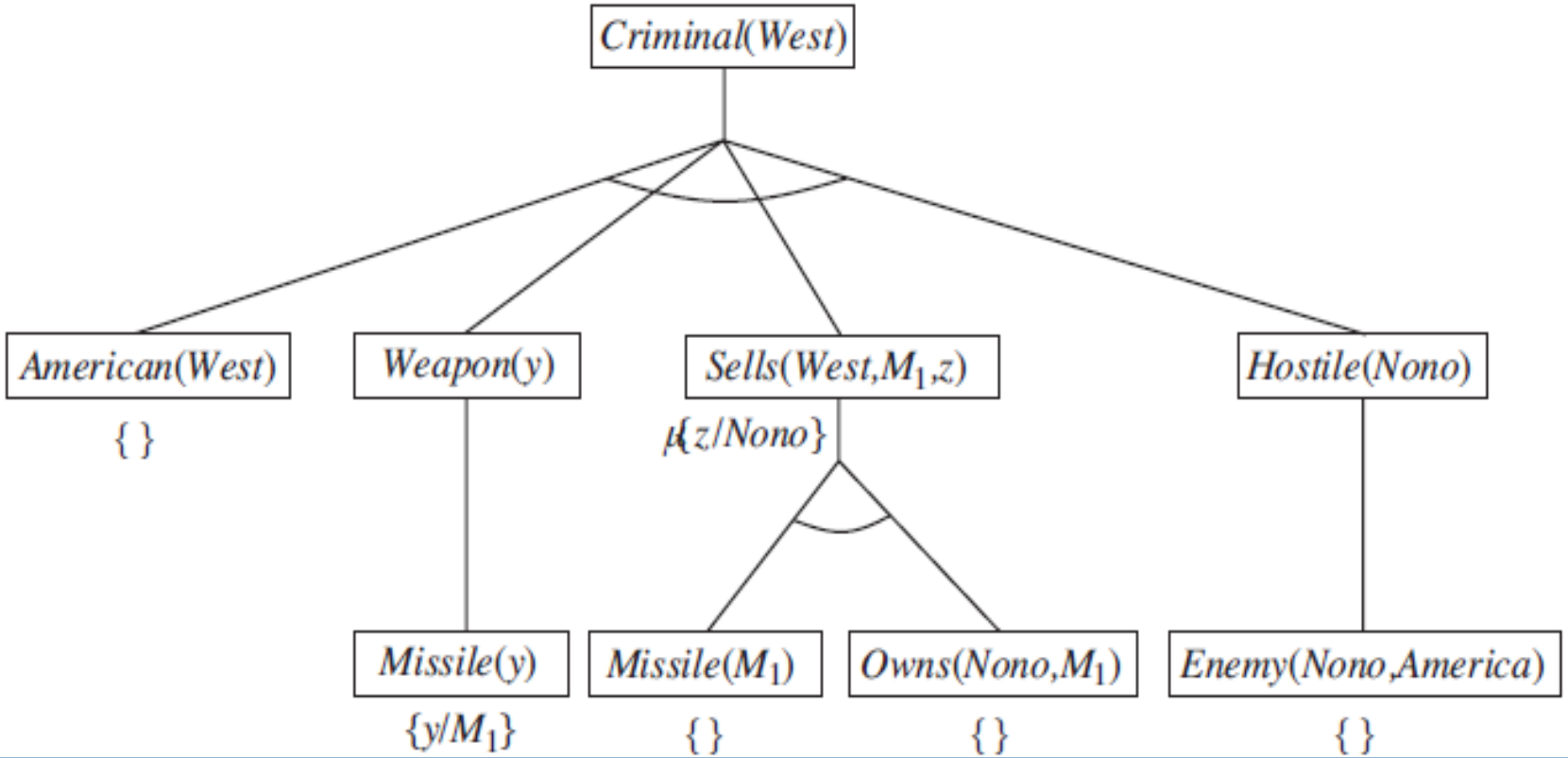
$$Missile(M_1)$$

$$American(West) .$$

$$Enemy(Nono, America)$$

Backward-chaining algorithm

Backward chaining is clearly a depth-first search algorithm.



Backward chaining

Consider the following sentences:

1.John likes all kinds of food

2.Apples are food

3.Chicken is food

4.Anything anyone eats and isn't killed by is food.

5.Bill eats peanut and is still alive

6. sue eats everything bill eats

(i) Translate these sentences into formulas in predicate logic.

(ii) Prove that john likes peanuts using Backward chaining

Resolution

- First-order resolution requires that sentences to be in conjunctive normal form (CNF)
- The procedure for conversion to CNF
- Resolution Refutation

Steps to convert FOL to Clause form

1. Eliminate →
2. Reduce the scope of negation
3. Standardize variables apart
4. Move all quantifiers to left
5. Eleminate existential quantifies
6. Drop all universal uantifiers
7. Convert to connect of disjunct form
8. Make each conjunct a seperate clause
9. Standardize variables apart again

ALGORITHM TO CONVERT TO CLAUSAL FORM

1. Eliminate biconditionals and implications

$(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

$(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

2. Move $\neg$ inwards

$\neg (\neg P) \Rightarrow P$

$\neg (P \vee Q) \Rightarrow \neg P \wedge \neg Q$

$\neg (P \wedge Q) \Rightarrow \neg P \vee \neg Q$

$\neg (\forall x) P \Rightarrow (\exists x) \neg P$

$\neg (\exists x) P \Rightarrow (\forall x) \neg P$

ALGORITHM TO CONVERT TO CLAUSAL FORM

3. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Example:

∀ x : man(x)	∀ x : man(x)
∀ x : Happy(x)	∀ y : Happy(y)
∃ x : boys(x)	∃ z : boys(z)

ALGORITHM TO CONVERT TO CLAUSAL FORM
Skolem constants and functions

4. Skolemization

Eliminate existential quantifier and replace it by skolem constants

Example:

$\exists$ x : Smile(x)

$\exists$ y : Graduating(y)

After Skolemization

Smile(A)

Graduating(B)

ALGORITHM TO CONVERT TO CLAUSAL FORM

5. Drop or Remove universal quantifiers by

(1) moving them all to the left end;

(2) making the scope of each the entire sentence;

(3) dropping the “prefix” part

Example:

$\forall x : Smile(x)$

$\forall y : Graduating(y)$

After Dropping

$Smile(x)$

$Graduating(y)$

ALGORITHM TO CONVERT TO CLAUSAL FORM

6. Apply distribute law / associative laws

$$(P \land Q) \lor R \Rightarrow (P \lor R) \land (Q \lor R)$$
$$(P \lor Q) \lor R \Rightarrow (P \lor Q \lor R)$$

7. Split conjuncts into separate clauses

8. Standardize variables so each clause contains only variable names that do not occur in any other clause

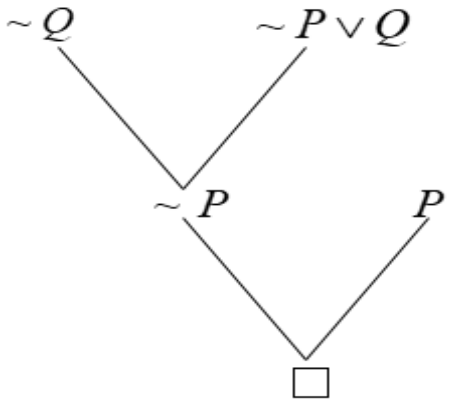
Resolution Refutation

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions.
- It was invented by a Mathematician John Alan Robinson in the year 1965.
- Resolution is used, if there are various statements given, and we need to prove a conclusion of those statements.

Resolution

- Pair of clauses being resolved is called the Resolvents.
- The resulting clause is called the **Resolute**.
- Choosing the correct pair of resolvents is a matter of search.

1. P
2. $\sim P \vee Q$
3. $\sim Q$



Resolution refutation proofs involve the following steps:

1. Put the premises or axioms into *clause form* (13.2.2).
2. Add the negation of what is to be proved, in clause form, to the set of axioms.
3. *Resolve* these clauses together, producing new clauses that logically follow from them (13.2.3).
4. Produce a contradiction by generating the empty clause.
5. The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true (13.2.4).

Resolution refutation proofs steps:

1. Convert facts into First order logic (FOL)
2. Convert FOL into **clause form**
3. Negate the statement to be proved , and add the result to the knowledge base.
4. Draw Resolution graph .
5. If empty clause (NIL) is produced , stop and report that original theorem is true .

Example -1

Problem Statement: **Prove by resolution**

1. If It is sunny and warm day you will enjoy
2. If it is raining you will get wet
3. It is warm day
4. It is raining
5. It is sunny

Goal: You will enjoy

Prove: enjoy

1.Convert to FOL

1. If It is sunny and warm day you will enjoy
2. If it is raining you will get wet
3. It is warm day
4. It is raining
5. It is sunny

Goal: You will enjoy

Prove: enjoy

Sunny ∧ warm → enjoy

raining → wet

warm

raining

Sunny

2.Convert FOL to CNF

Sunny Λ warm → enjoy

raining → wet

warm

raining

Sunny

- **Sunny Λ Warm → enjoy**
  Eliminate implication:
  ¬(Sunny Λ warm) V enjoy
  Moving negation inside
  ¬Sunny V ¬warm V enjoy
- **raining → wet**
  Eliminate implication:
  ¬raining V wet
- **warm**
- **raining**
- **Sunny**

3. Negate the statement to be proved

1. If It is sunny and warm day you will enjoy
2. If it is raining you will get wet
3. It is warm day
4. It is raining
5. It is sunny

Goal: You will enjoy

Prove: enjoy

¬enjoy

Draw resolution graph

1. Convert facts into First order logic (FOL)
2. Convert FOL into CNF (Conjuctive Normal Form)
3. Negate the statement to be proved , and add the result to the knowledge base.
4. Draw Resolution graph .
5. If empty clause (NIL) is produced , stop and report that original theorem is true .

¬Sunny V ¬warm V enjoy

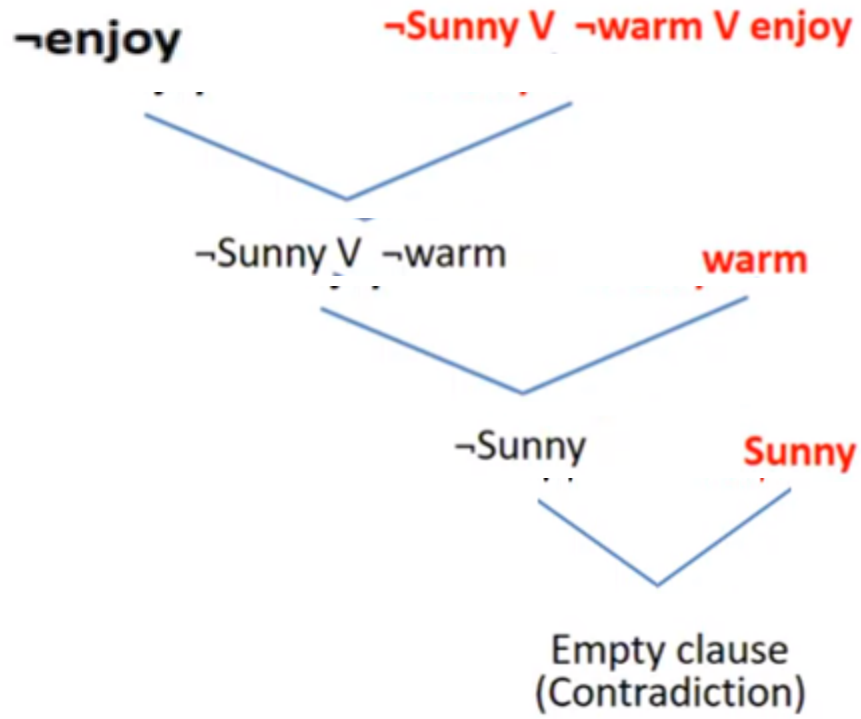
¬raining V wet

warm

raining

Sunny

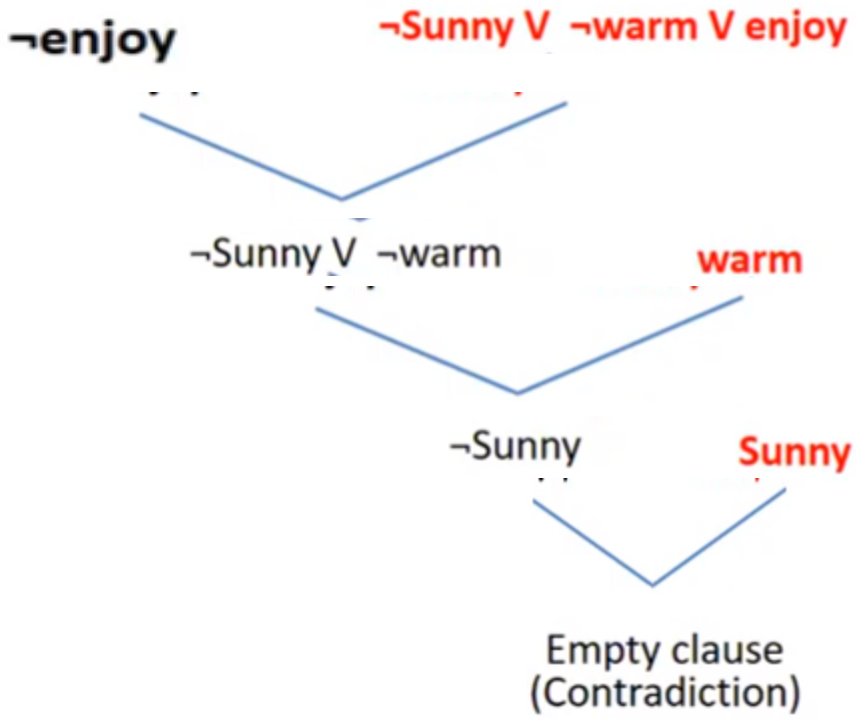
¬enjoy



Draw resolution graph

Hence we see that the negation of the conclusion has been proved as a complete contradiction with the given set of facts.

Hence Proved



Example-2

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

Goal: It will rain.

- Let P : The humidity is high
- Let Q: sky is cloudy

The humidity is high or the sky is cloudy

$P \vee Q$

- Let Q: sky is cloudy
- Let R: it will rain

If the sky is cloudy, then it will rain

$Q \rightarrow R$

- Let P : The humidity is high
- Let S: it is hot

If the humidity is high, then it is hot

$P \rightarrow S$

- Let S: it is hot

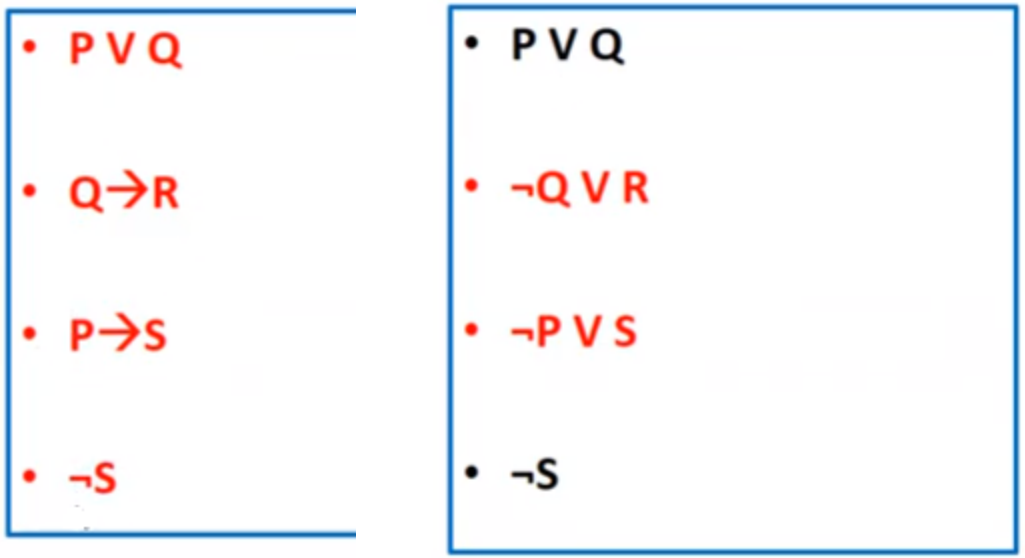
It is not hot

$\neg S$

Example-2

CNF Form

Goal: It will rain. (R)



Example-2

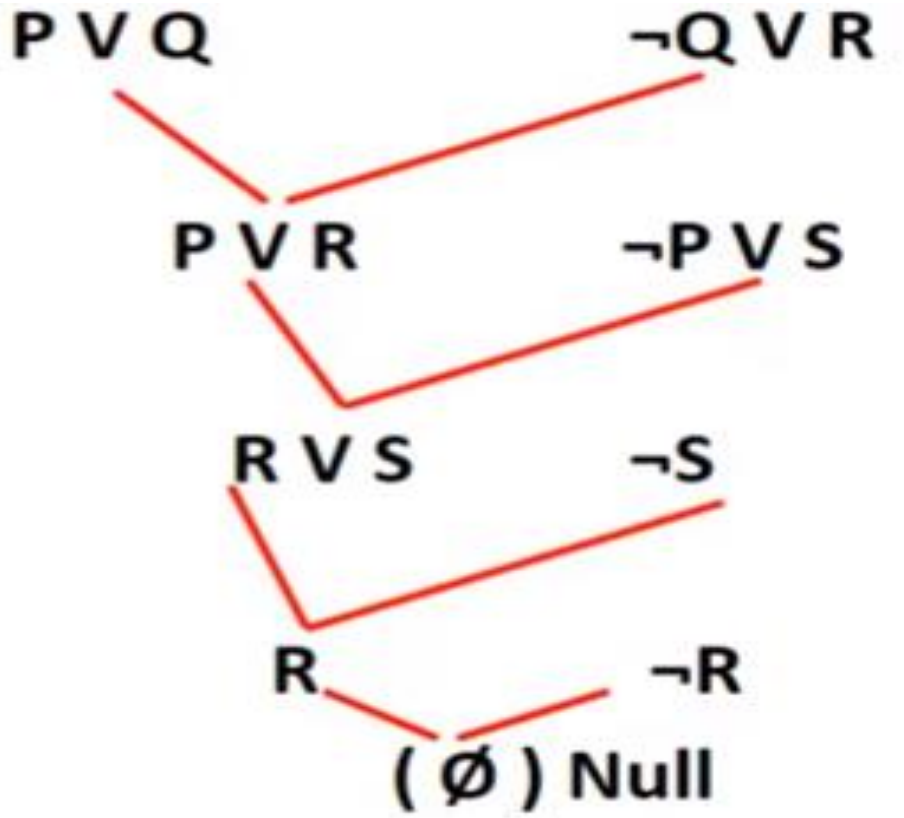
Goal: It will rain. **(R)**

Negation of Goal (¬R): It will not rain.

Example-2

- P V Q
- ¬Q V R
- ¬P V S
- ¬S

- ¬R (Goal)



Hence we see that the negation of the conclusion has been proved as a complete contradiction with the given set of facts.

Hence Proved

Problem using Resolution Refutation

Consider the below argument and using Resolution Refutation Procedure check whether 'John is happy'

"Anyone passing his history exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. John did not study, but John is lucky. Anyone who is lucky wins the lottery."

Problem using Resolution Refutation

Anyone passing his history exams and winning the lottery is happy.

But anyone who studies or is lucky can pass all his exams.

John did not study, but John is lucky.

Anyone who is lucky wins the lottery

Anyone passing his history exams and winning the lottery is happy.

$\forall$ **X (pass (X,history)** $\wedge$ **win (X,lottery)** $\rightarrow$ **happy (X))**

Anyone who studies or is lucky can pass all his exams.

$\forall$ **X** $\forall$ **Y (study (X)** $\vee$ **lucky (X)** $\rightarrow$ **pass (X,Y))**

John did not study but he is lucky.

$\neg$ **study (john)** $\wedge$ **lucky (john)**

Anyone who is lucky wins the lottery.

$\forall$ **X (lucky (X)** $\rightarrow$ **win (X,lottery))**

These four predicate statements are now changed to clause form (Section 12.2.2):

1. $\neg$ **pass (X, history)** $\vee$ $\neg$ **win (X, lottery)** $\vee$ **happy (X)**
2. $\neg$ **study (Y)** $\vee$ **pass (Y, Z)**
3. $\neg$ **lucky (W)** $\vee$ **pass (W, V)**
4. $\neg$ **study (john)**
5. **lucky (john)**
6. $\neg$ **lucky (U)** $\vee$ **win (U, lottery)**

Into these clauses is entered, in clause form, the negation of the conclusion:

7. $\neg$ **happy (john)**

Anyone passing his history exams and winning the lottery is happy.

$\forall$ **X (pass (X,history)** $\wedge$ **win (X,lottery)** $\rightarrow$ **happy (X))**

Anyone who studies or is lucky can pass all his exams.

$\forall$ **X** $\forall$ **Y (study (X)** $\vee$ **lucky (X)** $\rightarrow$ **pass (X,Y))**

John did not study but he is lucky.

$\neg$ **study (john)** $\wedge$ **lucky (john)**

Anyone who is lucky wins the lottery.

$\forall$ **X (lucky (X)** $\rightarrow$ **win (X,lottery))**

1) $\forall x\ (pass(x, history) \wedge win(x, lottery) \rightarrow happy(x))$

$\forall x\ \neg[(pass(x, history) \wedge win(x, lottery)] \vee happy(x)$

$\forall x\ \neg pass(x, history) \vee \neg win(x, lottery) \vee happy(x)$

① $\neg pass(x, history) \vee \neg win(x, lottery) \vee happy(x)$

2) $\forall x \forall y \; (study(x) \vee lucky(x) \rightarrow Pass(x, y))$

Remove implication

$\forall x \forall y \; \neg [Study(x) \vee lucky(x)] \vee Pass(x, y)$

Move Negation inward

$\forall x \forall y \; [\neg study(x) \wedge \neg lucky(x)] \vee Pass(x, x)$

Remove Universal Quantifier

$(\neg study(x) \wedge \neg lucky(x)) \vee Pass(x, y)$

Apply Distributive law

$[\neg Study(x) \vee Pass(x, y)] \wedge [\neg lucky(x) \vee Pass(x, y)]$

Split Conjuct into clauses

② $\neg Study(x) \vee Pass(x, y)$

③ $\neg lucky(x) \vee Pass(x, y)$

② $\neg Study(Y) \vee Pass(Y, Z)$

③ $\neg lucky(W) \vee Pass(W, V)$

3) $\neg$ study(join) $\wedge$ lucky (john)

Split Conjuct into clauses

(4)	$\neg$ study (join)
(5)	lucky (john)

4) $\forall x \, (lucky(x) \rightarrow win(x, lottery))$

$\forall x \, (\neg lucky(x) \vee win(x, lottery))$

⑥ $\neg lucky(U) \vee win(U, lottery)$

These four predicate statements are now changed to clause form (Section 12.2.2):

1. ¬ pass (X, history) ∨ ¬ win (X, lottery) ∨ happy (X)
2. ¬ study (Y) ∨ pass (Y, Z)
3. ¬ lucky (W) ∨ pass (W, V)
4. ¬ study (john)
5. lucky (john)
6. ¬ lucky (U) ∨ win (U, lottery)

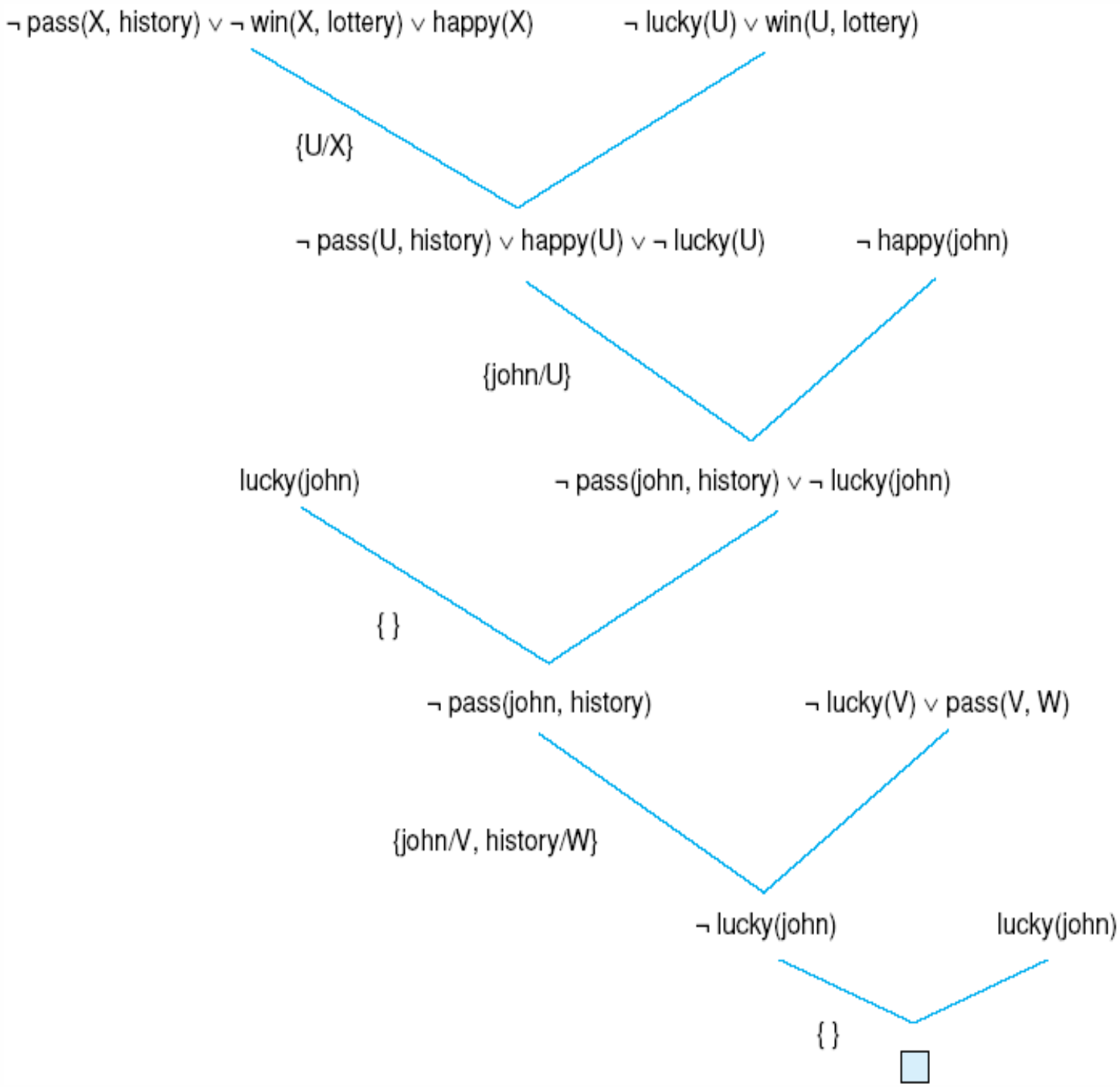
John is Happy

Happy (John)

(7) ¬ happy (John)

RAMAIAH
Institute of Technology

1. ¬ pass (X, history) ∨ ¬ win (X, lottery) ∨ happy (X)
2. ¬ study (Y) ∨ pass (Y, Z)
3. ¬ lucky (W) ∨ pass (W, V)
4. ¬ study (john)
5. lucky (john)
6. ¬ lucky (U) ∨ win (U, lottery)
7. ¬ happy (john)



Problem – Dead Dog Problem

Axioms :

1. All dogs are animals
2. Fido is a dog
3. All animals will die

Theorem to proof : Fido will die

Problem – Dead Dog Problem

Axioms :

1. All dogs are animals : $\forall X [dog(X) \Rightarrow animal(X)]$
2. Fido is a dog : $dog(fido).$
3. All animals will die : $\forall Y [animal(Y) \Rightarrow die(Y)]$

Theorem to proof :

Fido will die : $die(fido).$

Problem – Dead Dog Problem

1. $\forall X\ [dog(X) \Rightarrow animal(X)]$

   $\forall X\ \neg dog(X) \vee animal(X)$

   $\neg dog(X) \vee animal(X)$

2. $dog(fido)$

3. $\forall Y\ [animal(Y) \Rightarrow die(Y)]$

   $\forall Y\ \neg animal(Y) \vee die(Y)$

   $\neg animal(Y) \vee die(Y)$

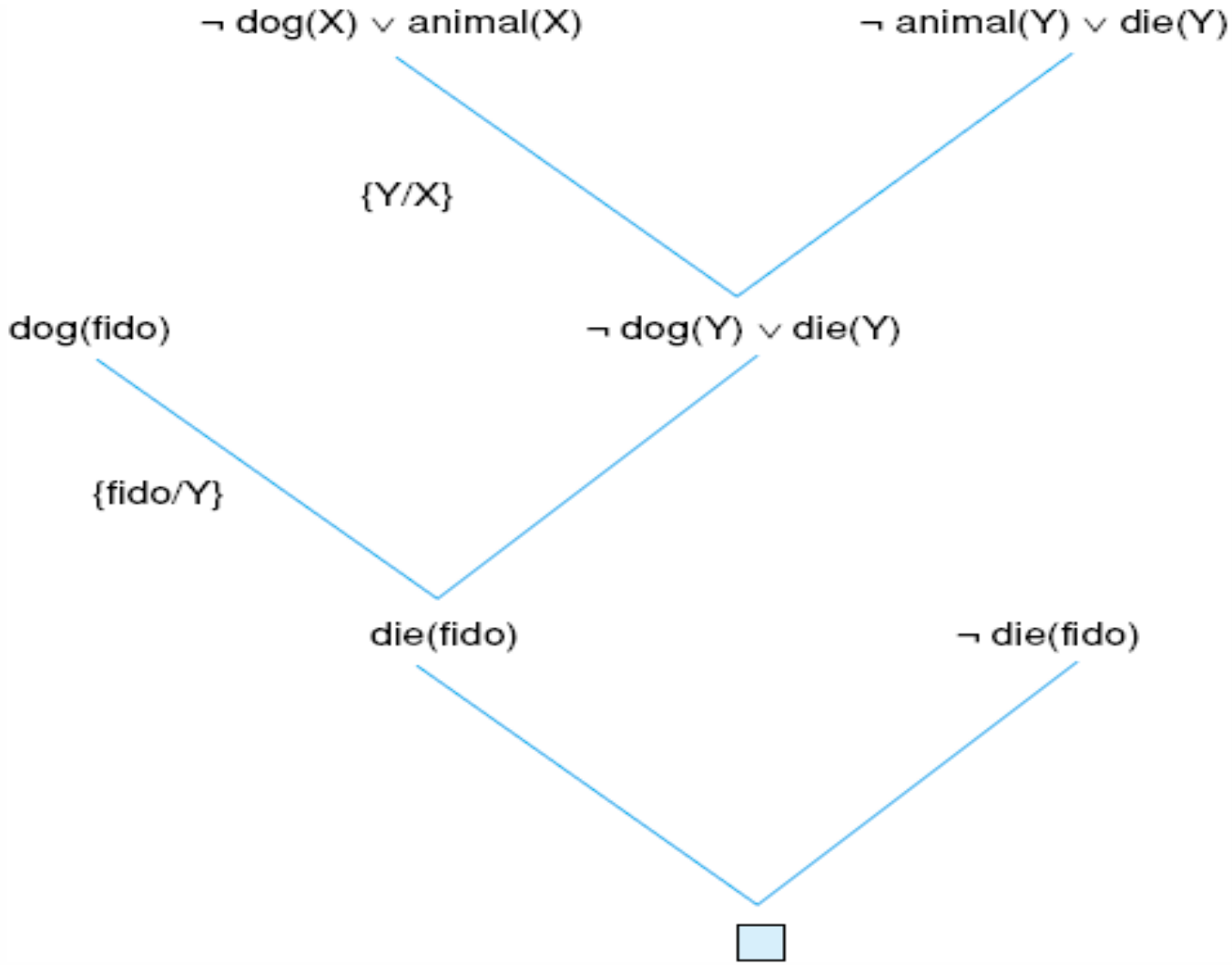
Problem – Dead Dog Problem

die(fido)

¬ die(fido)

Problem – Dead Dog Problem

- ¬ dog(X) ∨ animal(X)
- dog(fido)
- animal(Y) ∨ die(Y)
- ¬ die(fido)



All people who are not poor and are smart are happy. Those people who read are not stupid. John can read and is wealthy. Happy people have exciting lives. Can anyone be found with an exciting life?

We assume $\forall$**X (smart (X)** $\equiv \neg$ **stupid (X)) and** $\forall$**Y (wealthy (Y)** $\equiv \neg$ **poor (Y)), and get**:

$\forall$**X (**$\neg$ **poor (X)** $\wedge$ **smart (X)** $\rightarrow$ **happy (X))**

$\forall$**Y (read (Y)** $\rightarrow$ **smart (Y))**

read (john) $\wedge \neg$ **poor (john)**

$\forall$**Z (happy (Z)** $\rightarrow$ **exciting (Z))**

The negation of the conclusion is:

$\neg \exists$ **W (exciting (W))**

These predicate calculus expressions for the "exciting life" problem are transformed into the following clauses:

poor (X) $\vee \neg$ **smart (X)** $\vee$ **happy (X)**

$\neg$ **read (Y)** $\vee$ **smart (Y)**

read (john)

$\neg$ **poor (john)**

$\neg$ **happy (Z)** $\vee$ **exciting (Z)**

$\neg$ **exciting (W)**

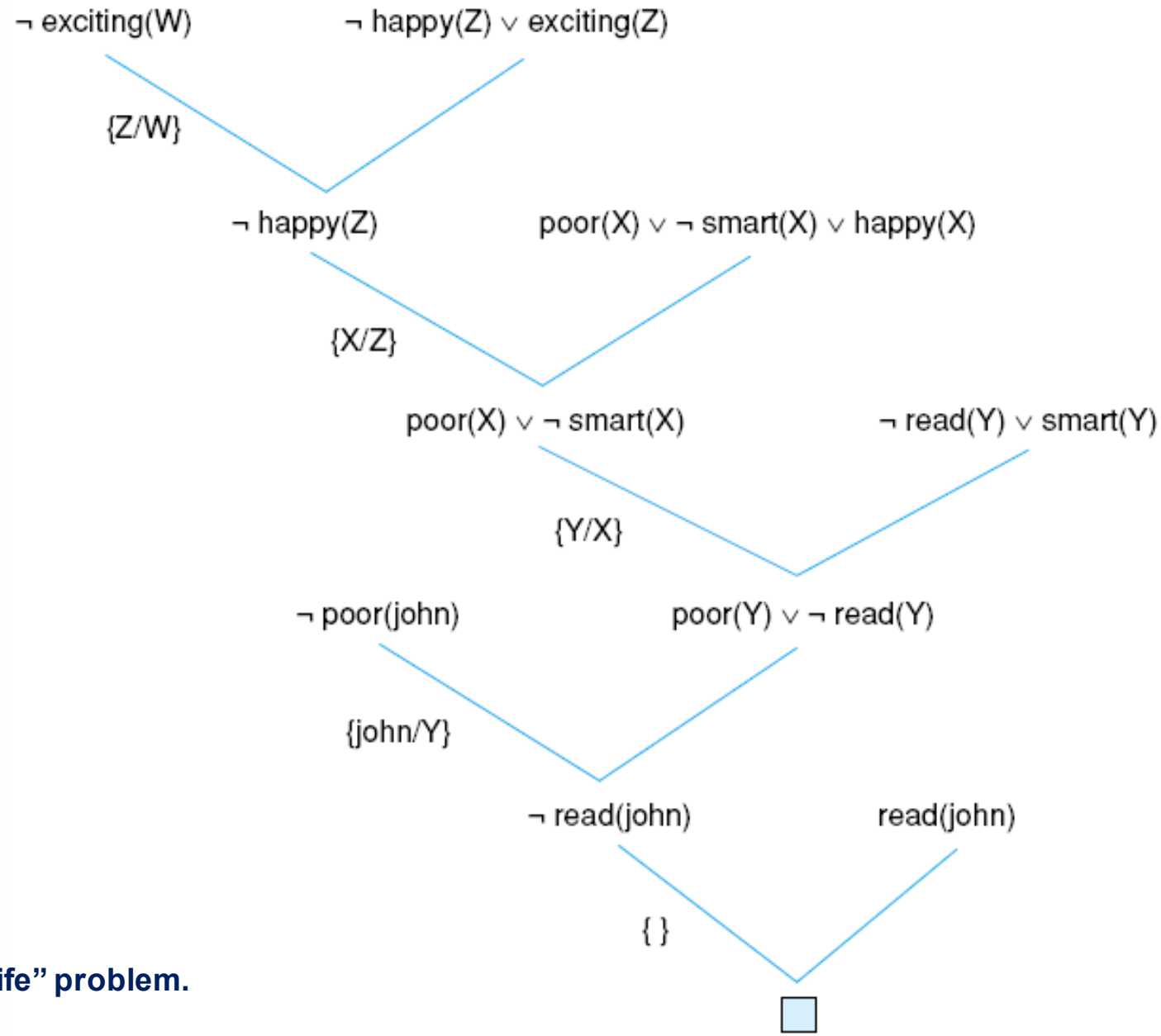
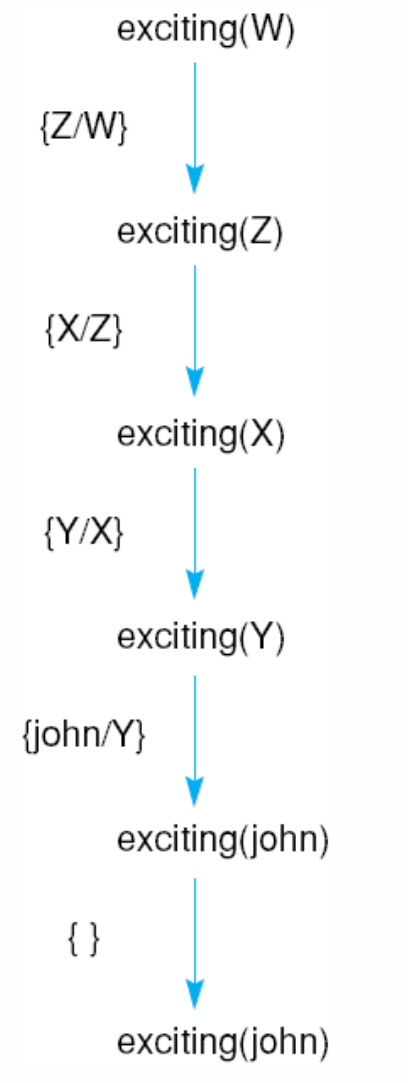
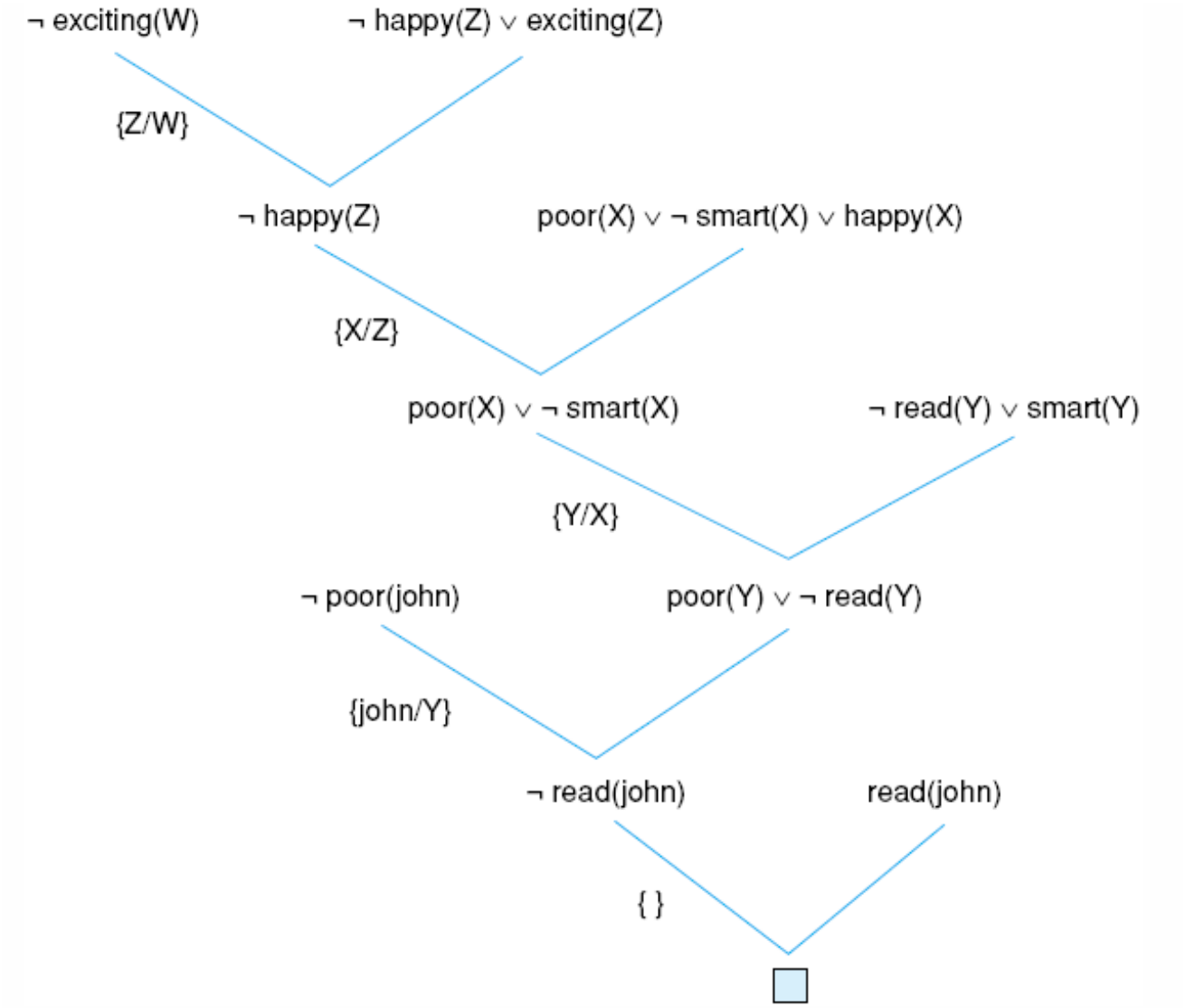


Fig. One refutation for the "exciting life" problem.

Fig Unification substitutions applied to the original query.





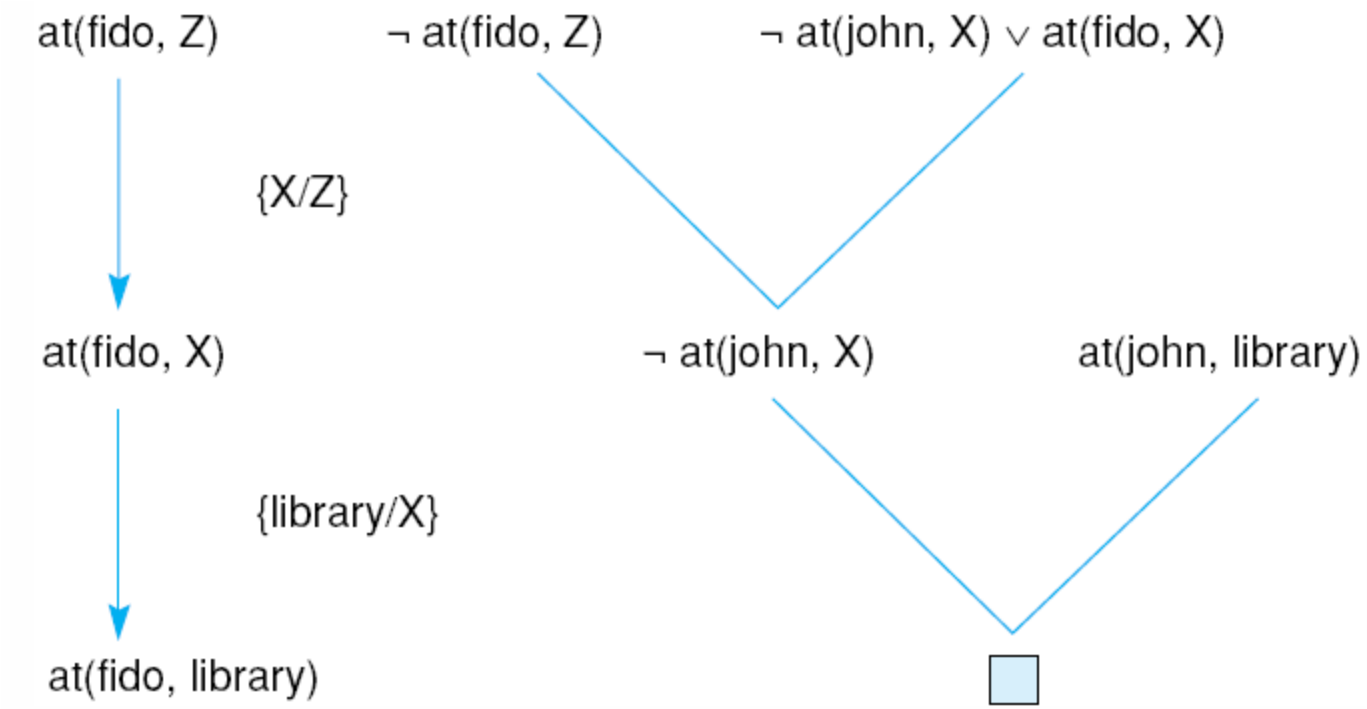
Extract the answer for the question **“Where is Fido?”**, given the below axioms in the KB.

1. Dog Fido goes wherever John goes.

2. John is at library.

Fig. Answer extraction process on the "finding fido" problem.

Dog Fido goes wherever John goes. John is at library. Where is Fido?



" If tom eats whatever Jade eats, and Tom eats banana, then what does Jade eat ?"

1. $\forall X$ eat(tom, X) $\rightarrow$ eat(jade, X)
2. eat (tom, banana)

Conclusion:

$\exists X$ Eat (jade, X) or Eat (jade, X)

Negation of Conclusion

3. $\neg$ Eat (jade, X)

Clausal forms.

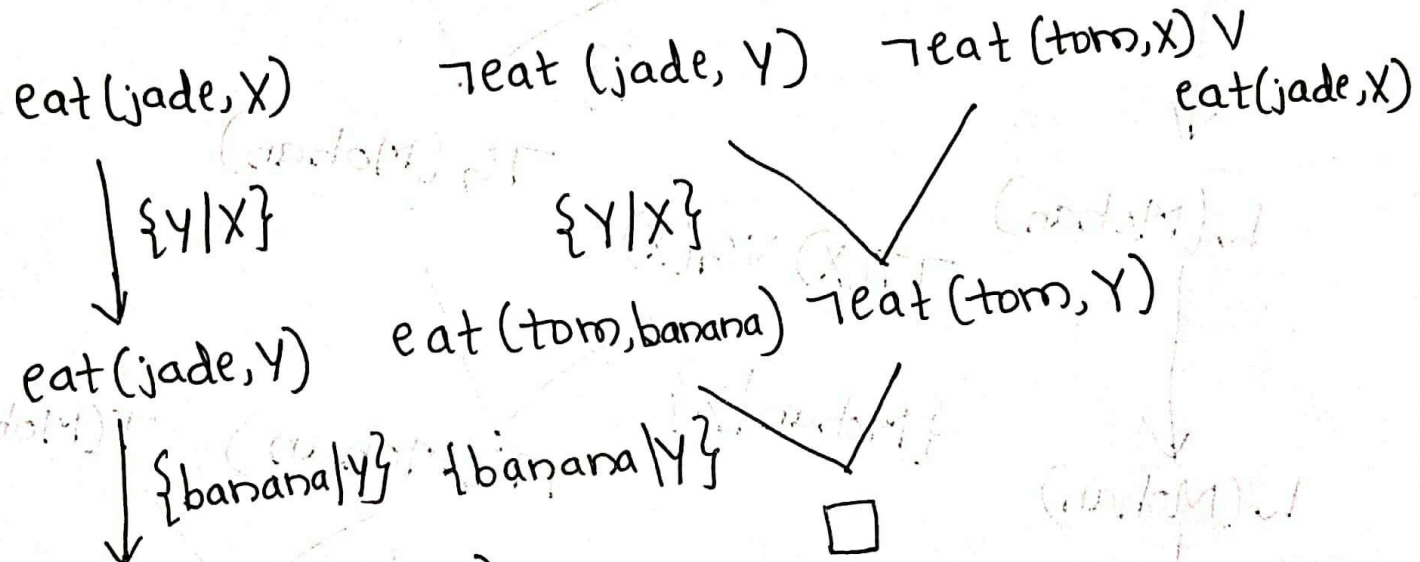
1. $\forall X$ eat (tom, X) $\rightarrow$ eat (jade, X)

$\Rightarrow \forall X \neg$ eat (tom, X) $\vee$ eat (jade, X)

$\Rightarrow \neg$ eat (tom, X) $\vee$ eat (jade, X)

2. eat (tom, banana)

3. $\neg$ eat (jade, Y)



Resolution Problem

All teachers are good. Anyone who is good and intelligent will deliver excellent lecture. Mohan is an intelligent teacher. Show that Mohan will deliver an excellent lecture.

- All teachers are good.
- Anyone who is good and intelligent will deliver excellent lecture.
- Mohan is an intelligent teacher.
- Show that Mohan will deliver an excellent lecture.

Predicates:

T(X): X is a teacher

G(X): X is good

I(X): X is intelligent

L(X): X delivers an excellent lecture

Predicate forms for given statements:

1. $\forall X \; T(X) \rightarrow G(X)$
2. $\forall Y \; G(Y) \wedge I(Y) \rightarrow L(Y)$
3. $I(\text{Mohan}) \wedge T(\text{Mohan})$

Conclusion:

$L(\text{Mohan})$

Negation of Conclusion:

4. $\neg L(\text{Mohan})$

Predicate form to Clausal form:

1. $\neg T(X) \vee G(X)$

2. $\forall Y \; (G(Y) \wedge I(Y)) \rightarrow L(Y)$

$\Rightarrow \forall Y \; \neg(G(Y) \wedge I(Y)) \vee L(Y)$

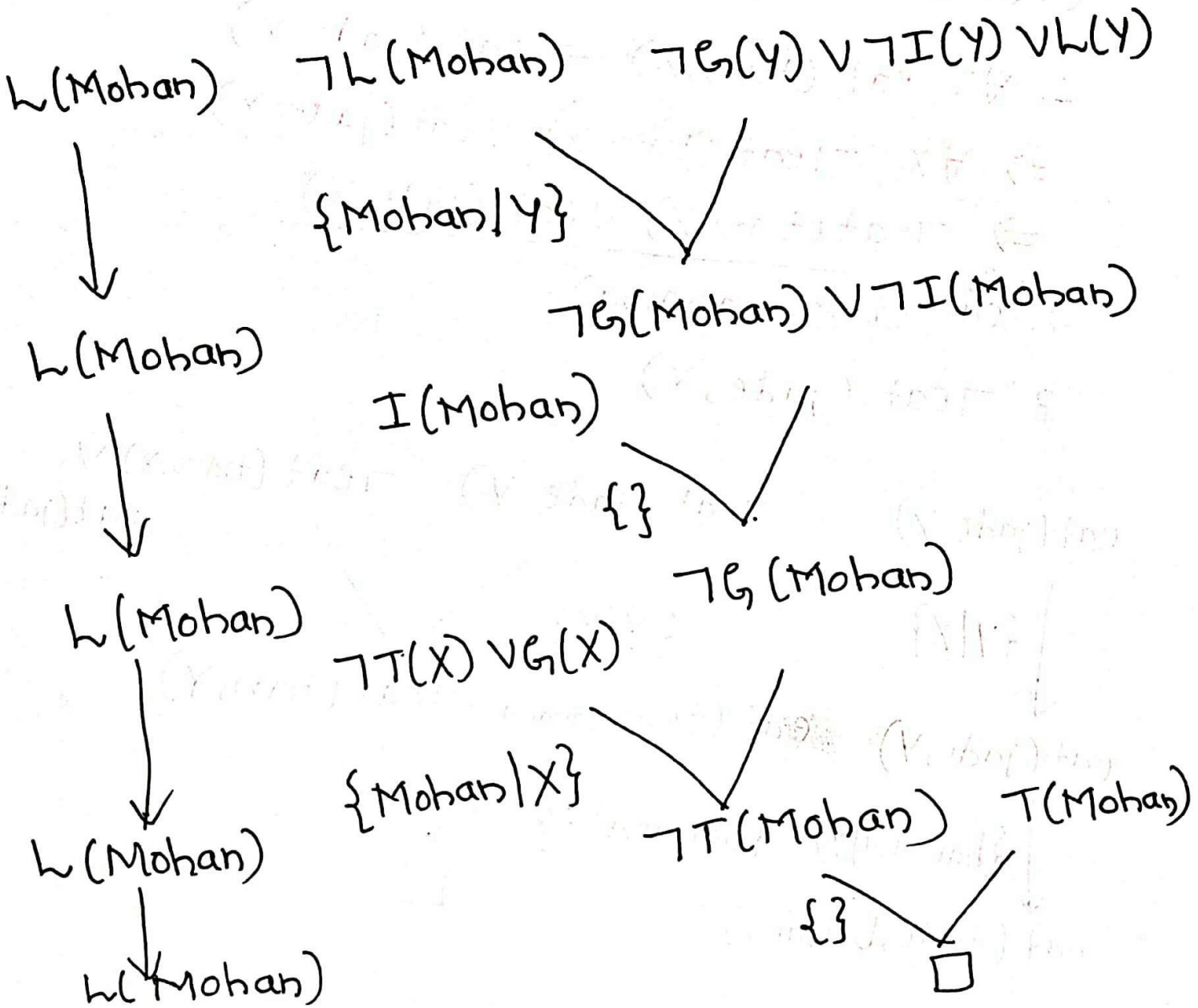
$\Rightarrow \forall Y \; \neg G(Y) \vee \neg I(Y) \vee L(Y)$

$\Rightarrow \underline{\neg G(Y) \vee \neg I(Y) \vee L(Y)}$

3. I (Mohan)

4. T (Mohan)

5. $\neg L$ (Mohan)


L(Mohan)
¬L(Mohan) ¬G(Y) V ¬I(Y) VL(Y)
{Mohan|Y}
¬G(Mohan) V ¬I(Mohan)
L(Mohan)
I(Mohan)
{}
¬G(Mohan)
L(Mohan)
¬T(X) VG(X)
{Mohan|X}
L(Mohan)
¬T(Mohan)
T(Mohan)
{}
L(Mohan)

Thank you