Knowledge Representation

What is knowledge representation?

- Humans are best at **understanding, reasoning, and interpreting knowledge.**
- Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world.
- But how machines do all these things comes under knowledge representation and reasoning.
- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents **thinking and how thinking contributes to intelligent behavior of agents.**

Knowledge representation

- It is responsible for **representing information about the real world** so that a computer can understand and can utilize this knowledge
- It is also a way which describes **how we can represent knowledge** in artificial intelligence.
- Knowledge representation **is not just storing data into some database**,but it also enables an intelligent **machine to learn** from that knowledge and experiences so that it can behave intelligently like a human.

Types of Knowledge

1. Declarative Knowledge:

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

2. Procedural Knowledge

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

Types of Knowledge

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.
- Some examples of meta knowledge include planning, learning.

4. Heuristic knowledge:

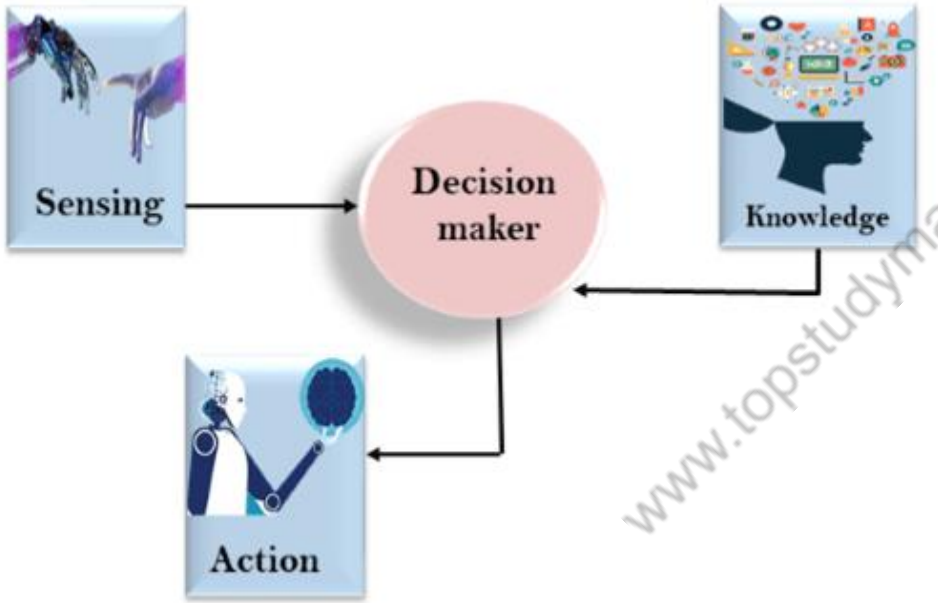
- Heuristic knowledge is representing knowledge of some experts in a field or subject.
- Heuristic knowledge is rules of thumb based on **previous experiences, awareness of approaches.**

Types of Knowledge

5. Structural knowledge:

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

Relation between knowledge and intelligence:



Knowledge plays an important role in demonstrating intelligent behavior in AI agents.

An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

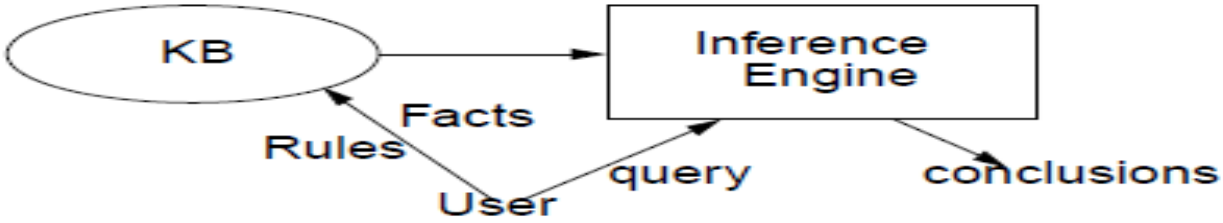
Logical Agents

- Knowledge-based agents – agents that have an explicit representation of knowledge that can be reasoned with.

- These agents can manipulate this knowledge to infer new things at the “knowledge level”

Knowledge Based Agents

- Both TELL and ASK operations may involve **inference.**
- **Inference for** deriving new sentences from old.
- Job of inference: Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told (or TELLed) to the knowledge base previously.
- Inference process should not make things up as it goes along.



Knowledge Based Agents

- Like all agents, it takes a percept as input and returns an action.
- The agent maintains a knowledge base, KB, which may initially contain some **background knowledge**
- Each time the agent program is called, it does three things.

  1. First, it TELLs the knowledge base **what it perceives / input.**

  2. Second, it ASKs the knowledge base **what action it should perform.**

  In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.

  3. Third, the agent program TELLs the knowledge base **which action was chosen,** and the agent executes the action.

Generic Knowledge-Based Agent

function KB-AGENT(percept) **returns** an action

persistent: KB, a knowledge base

t , a counter, initially 0, indicating time (t is an time counter with initial value 0)

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

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

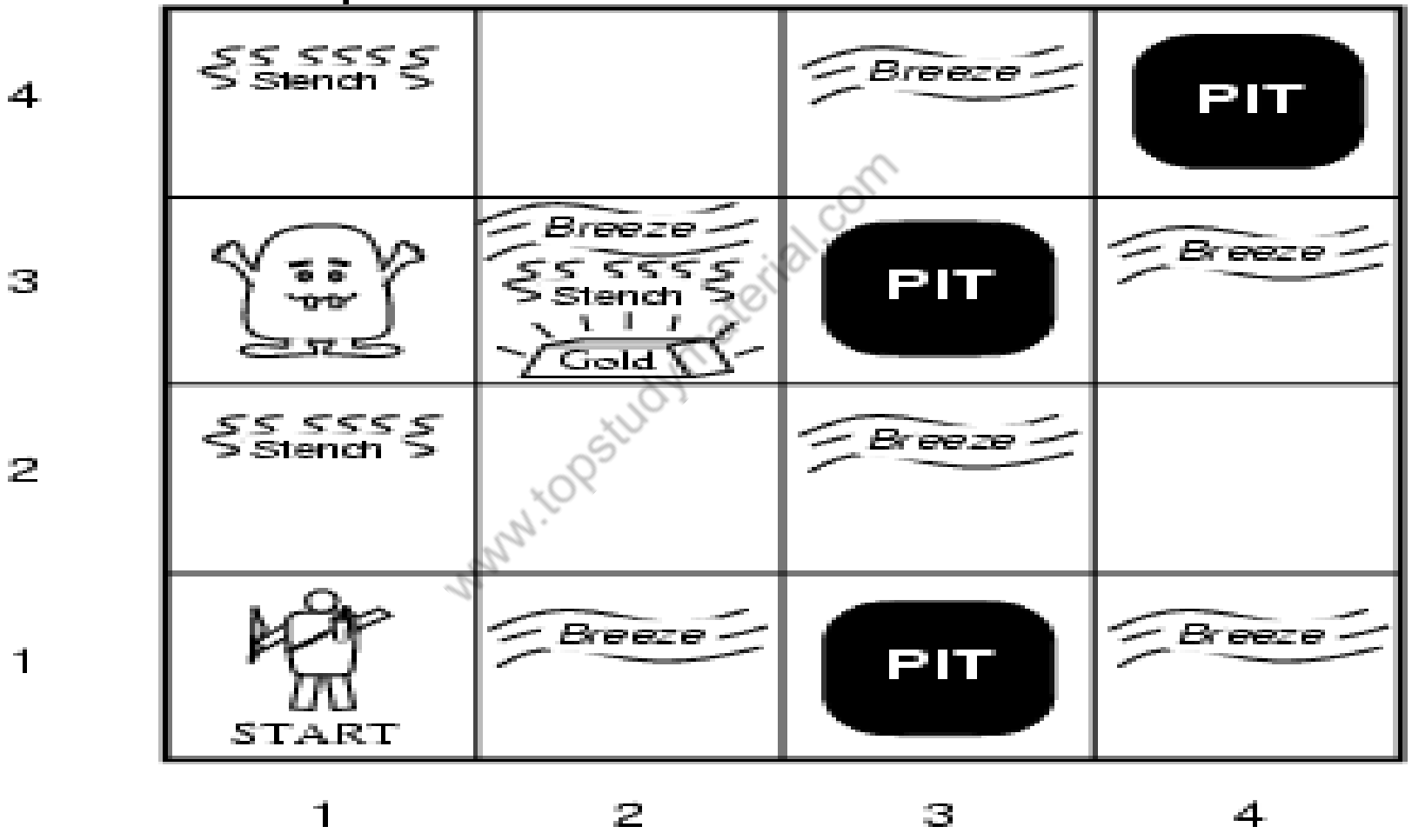
Generic Knowledge-Based Agent

- The details of the representation language are hidden inside three functions.
- MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept (Information) at the given time.
- MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
- MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.
- The details of the inference mechanisms are hidden inside TELL and ASK.

A simple knowledge-based agent

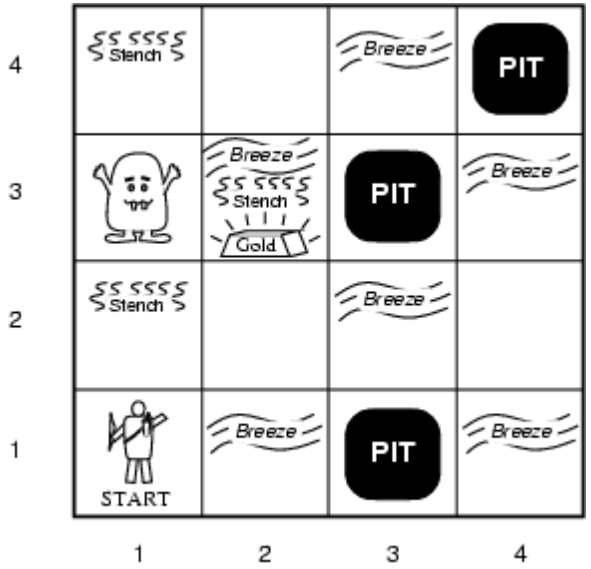
- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

A Wumpus World



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



- **Sensors:** Stench, Breeze, Glitter, Bump, Scream (shot Wumpus)
- **Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot

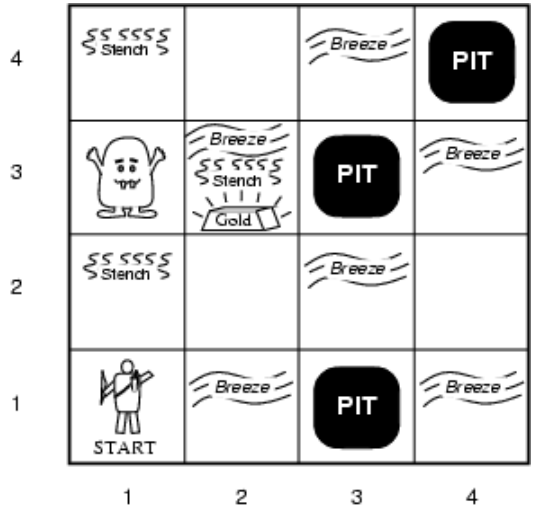
Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic No – sequential at the level of actions
- Static Yes – Wumpus and Pits do not move
- Discrete Yes
- Single-agent? Multi – (wumpus, eventually other agents)

Wumpus World

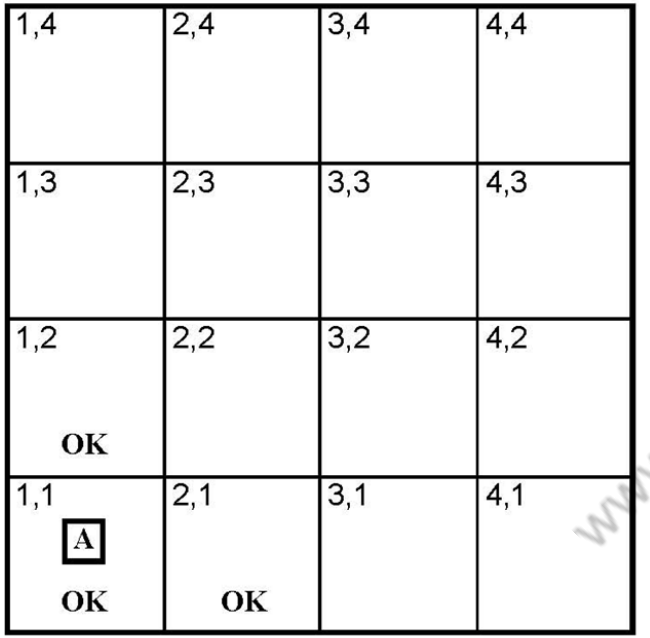
- Percepts given to the agent

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



- Principle Difficulty: agent is initially ignorant of the configuration of the environment – going to have to reason to figure out where the gold is without getting killed!

Exploring the Wumpus World



(a)

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

Initial situation:

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

From this the agent can infer the neighboring squares are safe (otherwise there would be a breeze or a stench)

(b)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

Percept: [None, None, None, None, None]

Deduce: Agent alive, so (1,1) OK
No breeze, so (1,2) and (2,1) OK

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

Percept: [None, None, None, None, None]

Deduce: Agent alive, so (1,1) OK
No breeze, so (1,2) and (2,1) OK

Action: Move East (turnright, goforward)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2 P?	3,2	4,2
1,1 OK	2,1 OK	3,1 P?	4,1

Percept: [None, Breeze, None, None, None]

Deduce: Pit in (2,2) or (3,1)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2 P?	3,2	4,2
1,1 OK	2,1 OK 	3,1 P?	4,1

Percept: [None, Breeze, None, None, None]

Deduce: Pit in (2,2) or (3,1)

Action: Back to (1,1) then to (1,2)
(turnleft, turnleft, goforward,
turnright, goforward)

Sample Run

1,4	2,4	3,4	4,4
1,3 W	2,3	2,4	2,5
1,2 (A) OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, None, None, None, None]

Deduce: Wumpus in (1,3)
No pit in (2,2), pit in (3,1)

Sample Run

1,4	2,4	3,4	4,4
1,3 W	2,3	3,3	4,3
1,2 (A) OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, None, None, None, None]

Deduce: Wumpus in (1,3)
No pit in (2,2), pit in (3,1)

Action: Move to (2,2) (turnright, goforward)
Ignore percept for now
Move to (2,3) (turnleft, goforward)

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 (A) OK G	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None,

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (2,2) (turnright, goforw
Ignore percept for now
Move to (2,3) (turnleft, goforwa

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 (A) OK G	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None, None]

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (1,1) through OK locations

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 (A) OK G	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None, None]

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (1,1) through OK locations

Now we look at

- How to represent facts / beliefs
  - ✓ "There is a pit in (2,2) or (3,1)"
- How to make inferences
  - ✓ "No breeze in (1,2), so pit in (3,1)"

Logic + Reasoning + Inference

- The knowledge bases consist of sentences.
- These sentences are expressed according to the **syntax** of the representation language
- Example, “x + y = 4” is a well-formed sentence, whereas “x4y+ =” is not.
- A logic must also define the **semantics** or meaning of sentences.
- The semantics defines the **truth** of each sentence with respect to each **possible world**.
- For example, the semantics for arithmetic specifies that the sentence “x + y =4” is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.
- In standard logics, every sentence must be either true or false only.

Logic + Reasoning + Inference

- Sentence: Individual piece of knowledge
  - English sentence forms one piece of knowledge in English language
  - Statement in C++ forms one piece of knowledge in C++ programming language

- Syntax: Form used to represent sentences
  - Syntax of C indicates legal combinations of symbols
  - a = 2 + 3; is legal
  - a = + 2 3 is not legal
  - Syntax alone does not indicate meaning

- Semantics: Mapping from sentences to facts in the world
  - They define the truth of a sentence in a "possible world"
  - Add the values of 2 and 3, store them in the memory location indicated by variable a
- In the language of arithmetic:

  x + 2 >= y is a sentence

  x2 + y > is not a sentence

  x + 2 >= y is true in all worlds where the number x + 2 is no less than the number y

  x + 2 >= y is true in a world where

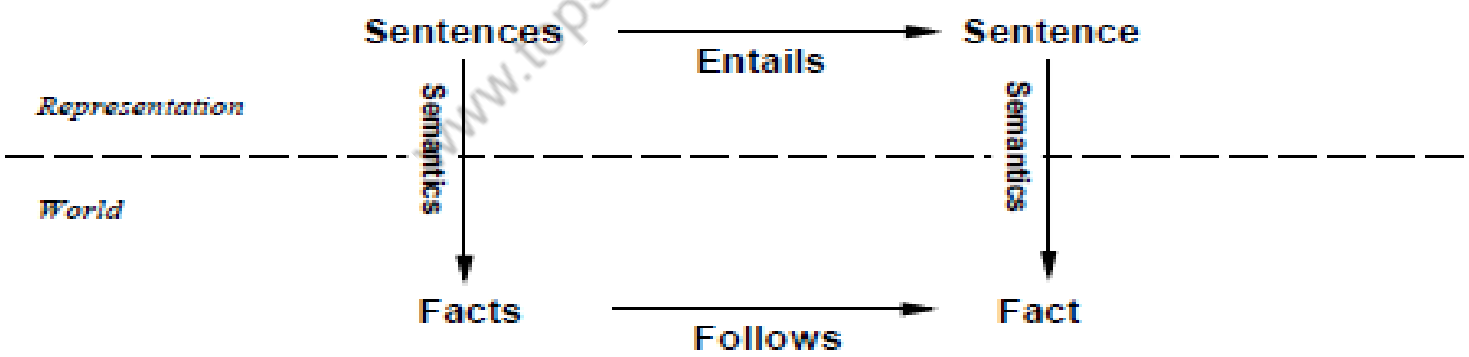
  x = 7, y = 1

  x + 2 >= y is false in a world where

  x = 0, y = 6

Knowledge Representation Languages and Inference

- A KR language is specified by
  - **Syntax**: The atomic symbols used in the language and how they can be composed to formal legal sentences.
  - **Semantics**: What fact about the world is represented by a sentence in the language, which determines whether it is true or false.



Knowledge Representation Languages and Inference

- **Logical inference** (**deduction**) derives new sentences in the language from existing ones.

Socrates is a man.

All men are mortal.

Socrates is mortal.

Proper inference should only derive sound conclusions. (ones that are true assuming the premises are true)

Propositional Logic Syntax

- Logical constants: True, False
- Propositional symbols: P, Q, etc. representing specific facts about the world.
- Constants and symbols are **atomic**, other sentences are **complex**.
- If S is a sentence, then (S) is a sentence
- If S and R are sentences then so are:

  S ∧ R: conjunction, S and R are conjuncts

  S ∨ R: disjunction, S and R are disjuncts

  S ⇒ R: implication, S is a premise or antecedent,

  R is the conclusion or consequent, also known as a rule or if-then statement
- S ⇔ R: equivalence (biconditional implication)
- ¬S: negation
- A **literal** is an atomic sentence or its negation (P, ¬S)
- Precedence of operators: ¬, ∧, ∨, ⇒, ⇔

- True and False indicate truth and falsity in the world.
- A proposition denotes whatever fixed
- e

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

derived from the semantics of their parts according to the following **truth table**.

Validity and Inference

- An **interpretation** is an assignment of True or False to each atomic proposition.
- A sentence that is true under any interpretation is **valid** (also called a **tautology** or **analytic sentence).**

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

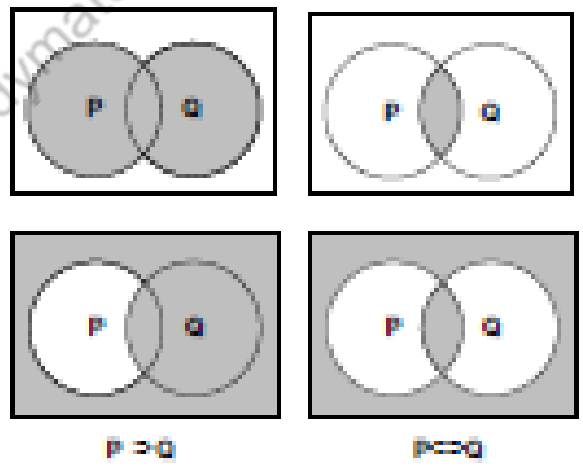
xploring

Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ⊨ β and β ⊨ α

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

- **Entailment: t**he relation between a sentence and another sentence that follows from it a[illegible] leads to a simple.
- Any interpretatio[illegible]tence is true is calle[illegible]f the sentence. (Venn Diagram)



Models and Entailment

- A sentence A **entails** a sentence B, (A |= B) if every model of A is also a model of B. In this case, if A is true then B must be true.
- Correct logical inference is characterized by entailment, we want to be able to infer whether a statement S follows from a knowledge base:

KB |= S

or

(KB -> S) is valid

Validity and Inference

- Inference can be performed by validity checking.
- If one has a set of sentences: $\{S_1,... S_n\}$ defining one's background knowledge, and one want to know whether a conclusion C logically follows, construct the sentence:

$S_1 \wedge S_2 \wedge ... \wedge S_n \Rightarrow C$

and check whether it is valid.

How many rows do we need to check?

Satisfiability and Complexity of Inference

- A sentence is **satisfiable** if it is true under some interpretation.
- Means it has a model, otherwise the sentence is **unsatisfiable**.
- A sentence is **valid** if and only if its negation is unsatisfiable.
- Algorithms for either validity or satisfiability checking are useful for logical inference.
- If there are *n* propositional symbols in a sentence, then simple validity checking must enumerate 2^n rows

Satisfiability and Complexity of Inference

- However, propositional satisfiability is the first problem to be proven NP-complete, and therefore there is assumed to be no polynomial-time algorithm.
- Therefore, sound and complete logical inference in propositional logic is intractable in general.
- But many problems can be solved very quickly.

Validity and satisfiability

- A sentence is valid if it is true in all models,
  e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the Deduction Theorem:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is satisfiable if it is true in some model
  e.g., $A \vee B$, C
- A sentence is unsatisfiable if it is true in no models
  e.g., $A \wedge \neg A$

Rules of Inference

- As an alternative to checking all rows of a truth table, one can use **rules of inference** to draw conclusions.
- A sequence of inference rule applications that leads to a desired conclusion is called a **logical proof**.
- Entailment: A |- B denotes that B can be derived by some inference procedure from the set of sentences A.
- Inference rules can be verified by the truth-table method and then used to construct sound proofs.
- Finding a proof is simply a search problem with the inference rules as operators and the conclusion as the goal.
- Logical inference can be more efficient than truth table construction.

Sample Rules of Inference

Entailments Examples:

- Modus Ponens: $\{\alpha \Rightarrow \beta, \alpha\} \vdash \beta$

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

- And Elimination: $\{\alpha \wedge \beta\} \vdash \alpha$; $\{\alpha \wedge \beta\} \vdash \beta$
- And Introduction: $\{\alpha, B\} \vdash \alpha \wedge \beta$
- Or introduction: $\{\alpha\} \vdash \alpha \vee \beta$
- Double negation Elimination: $\{\neg \neg\alpha\} \vdash \alpha$
- Implication Elimination: $\{\alpha \Rightarrow \beta\} \vdash \neg\alpha \vee \beta$
- Unit resolution: $\{\alpha \vee \beta, \neg\beta\} \vdash \alpha$

Sample Proof

- If John is not married, he is a bachelor. (¬P ⇒ Q)
- John is not a bachelor. (¬Q)
- Therefore, he is married. (P)

¬P ⇒ Q

¬¬P ∨ Q ; Implication élimination ({α ⇒ β} |- ¬α ∨ β)

P ∨ Q, ¬Q ; Double negation elimination ({¬ ¬α} |- α)

P : Unit resolution({α ∨ β, ¬β} |- α)

Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ⊨ β and β ⊨ α

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

Rules of Inference

- As an alternative to checking all rows of a truth table, one can use **rules of inference** to draw conclusions.
- A sequence of inference rule applications that leads to a desired conclusion is called a **logical proof**.
- Entailment: A |- B denotes that B can be derived by some inference procedure from the set of sentences A.
- Inference rules can be verified by the truth-table method and then used to construct sound proofs.
- Finding a proof is simply a search problem with the inference rules as operators and the conclusion as the goal.
- Logical inference can be more efficient than truth table construction.

Sample Rules of Inference

Entailments Examples:

- Modus Ponens: $\{\alpha \Rightarrow \beta, \alpha\} \mid\!- \beta$

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

- And Elimination: $\{\alpha \wedge \beta\} \mid\!- \alpha$; $\{\alpha \wedge \beta\} \mid\!- \beta$
- And Introduction: $\{\alpha, B\} \mid\!- \alpha \wedge \beta$
- Or introduction: $\{\alpha\} \mid\!- \alpha \vee \beta$
- Double negation Elimination: $\{\neg \neg\alpha\} \mid\!- \alpha$
- Implication Elimination: $\{\alpha \Rightarrow \beta\} \mid\!- \neg\alpha \vee \beta$
- Unit resolution: $\{\alpha \vee \beta, \neg\beta\} \mid\!- \alpha$
- Resolution: $\{\alpha \vee \beta, \neg\beta \vee \gamma\} \mid\!- \alpha \vee \gamma$. : Important

Sample Proof

- If John is not married, he is a bachelor. (¬P ⇒ Q)
- John is not a bachelor. (¬Q)
- Therefore, he is married. (P)

¬P ⇒ Q

¬¬P ∨ Q ; Implication élimination ({α ⇒ β} |- ¬α ∨ β)

P ∨ Q, ¬Q ; Double negation elimination ({¬ ¬α} |- α)

P : Unit resolution({α ∨ β, ¬β} |- α)

Proof methods

- Proof methods divide into (roughly) two kinds:
  - Application of inference rules
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule applications
      Can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form

Proof methods

- Proof methods divide into (roughly) two kinds:
  - Model checking
    - truth table enumeration (always exponential in *n*)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL)
    - heuristic search in model space (sound but incomplete)
      e.g., min-conflicts-like hill-climbing algorithms

Application of inference rules

Conversion to CNF

1. Eliminate ⇔, replacing α ⇔ β with (α ⇒ β)∧(β ⇒ α).

2. Eliminate ⇒, replacing α ⇒ β with ¬α∨ β.

3. Move ¬ inwards using de Morgan's rules and double-negation:

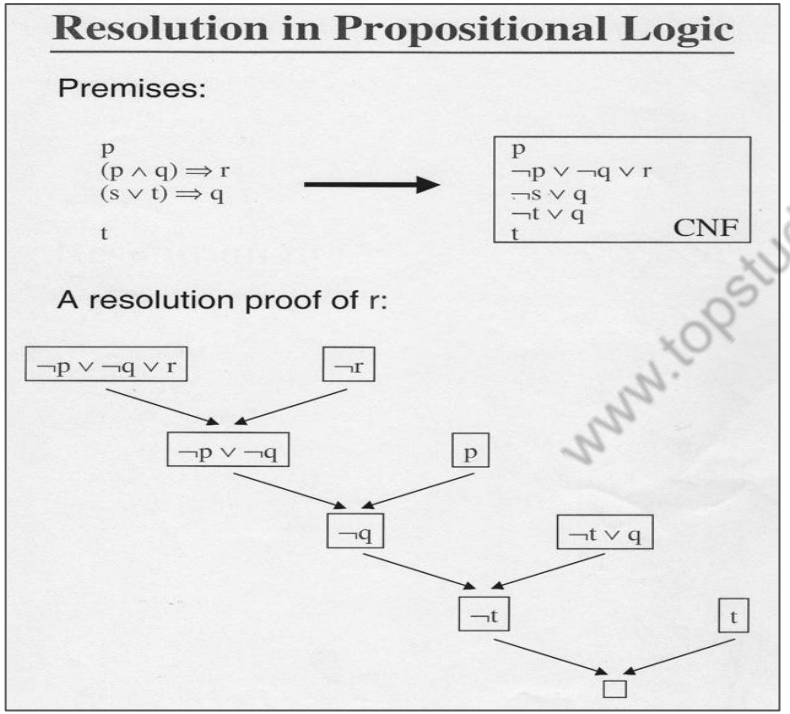
¬(avb)

¬a∧¬b

4. Apply distributivity law (∧ over ∨) and flatten

Resolution example

The Resolution rule state that if PvQ and ¬ PvR is true, then QvR will also be true.



Resolution in Propositional Logic

1. Convert all the propositions of KB to clause form (S).
2. Negate α and convert it to clause form. Add it to S.
3. Repeat until either a contradiction is found or no progress can be made.
   a. Select two clauses (α ∨ ¬P) and (γ ∨ P).
   b. Add the resolvent (α ∨ γ) to S.

Reasoning with Horn Clauses

- Forward Chaining
  - For each new piece of data, generate all new facts, until the desired fact is generated
  - Data-directed reasoning
- Backward Chaining
  - To prove the goal, find a clause that contains the goal as its head, and prove the body recursively
  - Goal-directed reasoning

Forward Chaining

- It is a strategy of an expert system to answer the question,

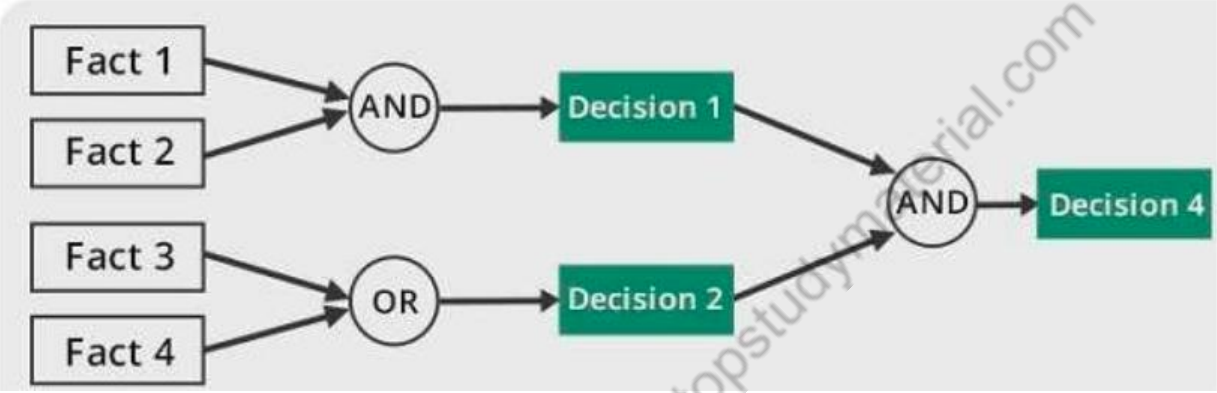
“What can happen next?”

- Data Driven

  Here, the **Inference Engine follows the chain of conditions and derivations and finally deduces the outcome.**

  It considers all the facts and rules, and sorts them before concluding to a solution.
- This strategy is followed for working on conclusion,result,or effect.
- For example, **prediction of share market status as an effect of changes in interest rates.**

Forward Chaining



Forward Chaining

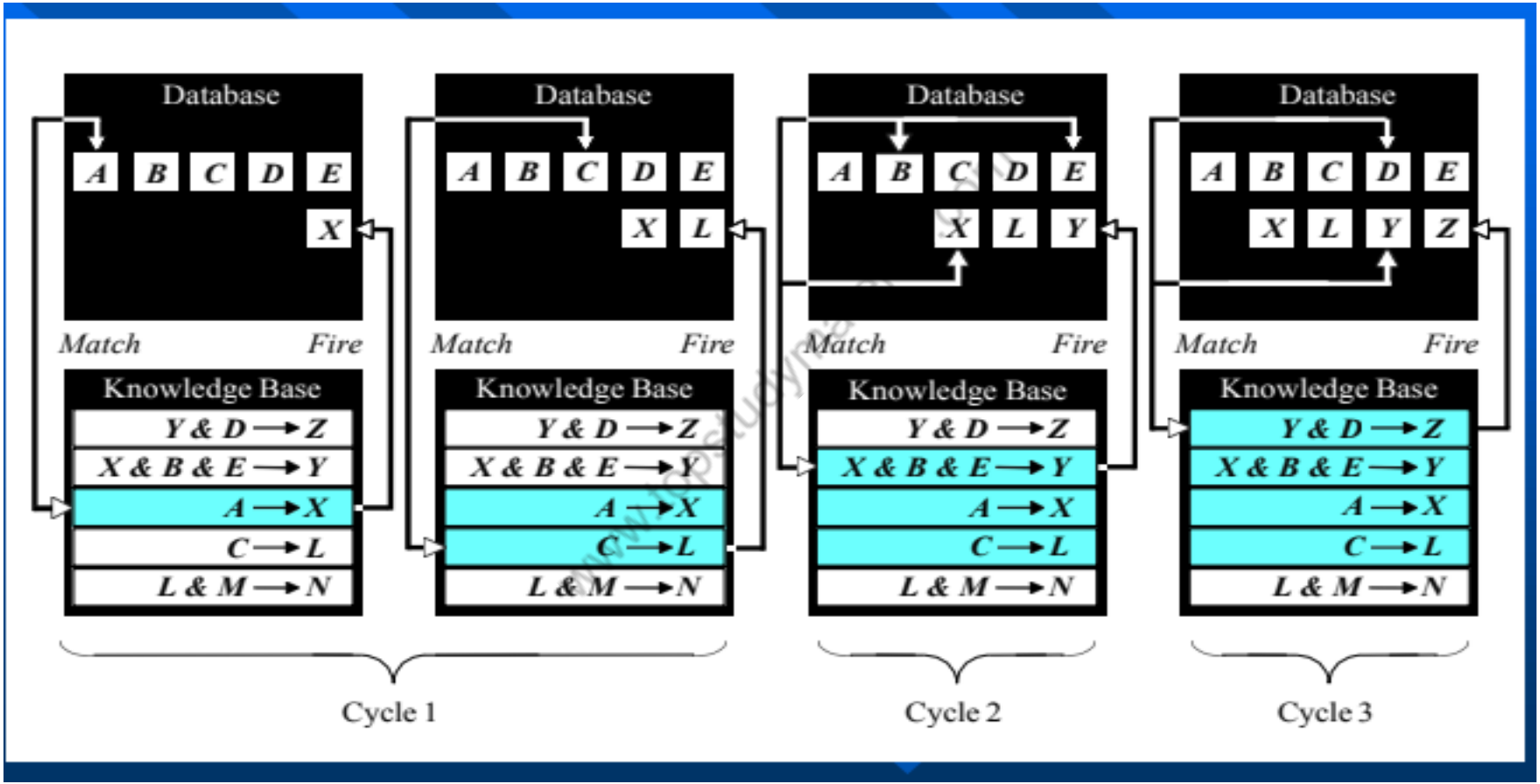
R1: IF A and C THEN E
R2: IF D and C THEN F
R3: IF B and E THEN F
R4: IF B THEN C
R5: IF F THEN G

Given facts:
A is true
B is true
What can be concluded?

Cycle through rules, looking for rules whose premise matches the working memory.

	Working memory
	A, B
R4 fires: assert new fact C	A, B, C
R1 fires: assert new fact E	A, B, C, E
R3 fires: assert new fact F	A, B, C, E, F
R5 fires: assert new fact G	A, B, C, E, F, G

Forward Chaining



Forward Chaining

A practical example will go as follows;

Tom is running (A)

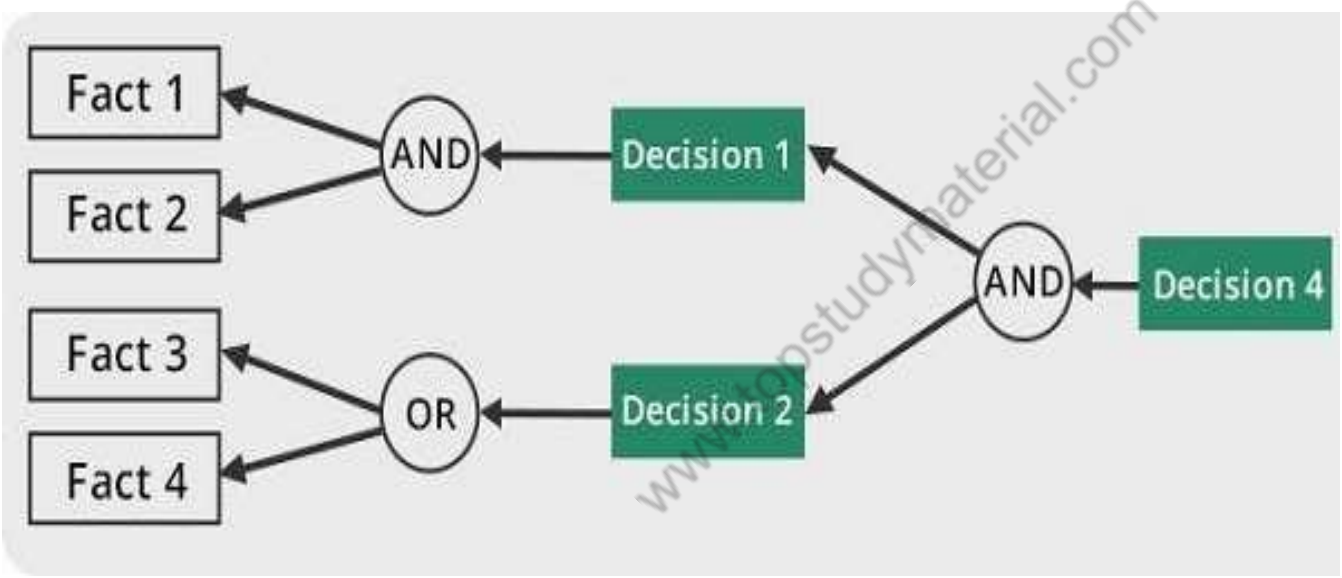
If a person is running, he will sweat (A->B)

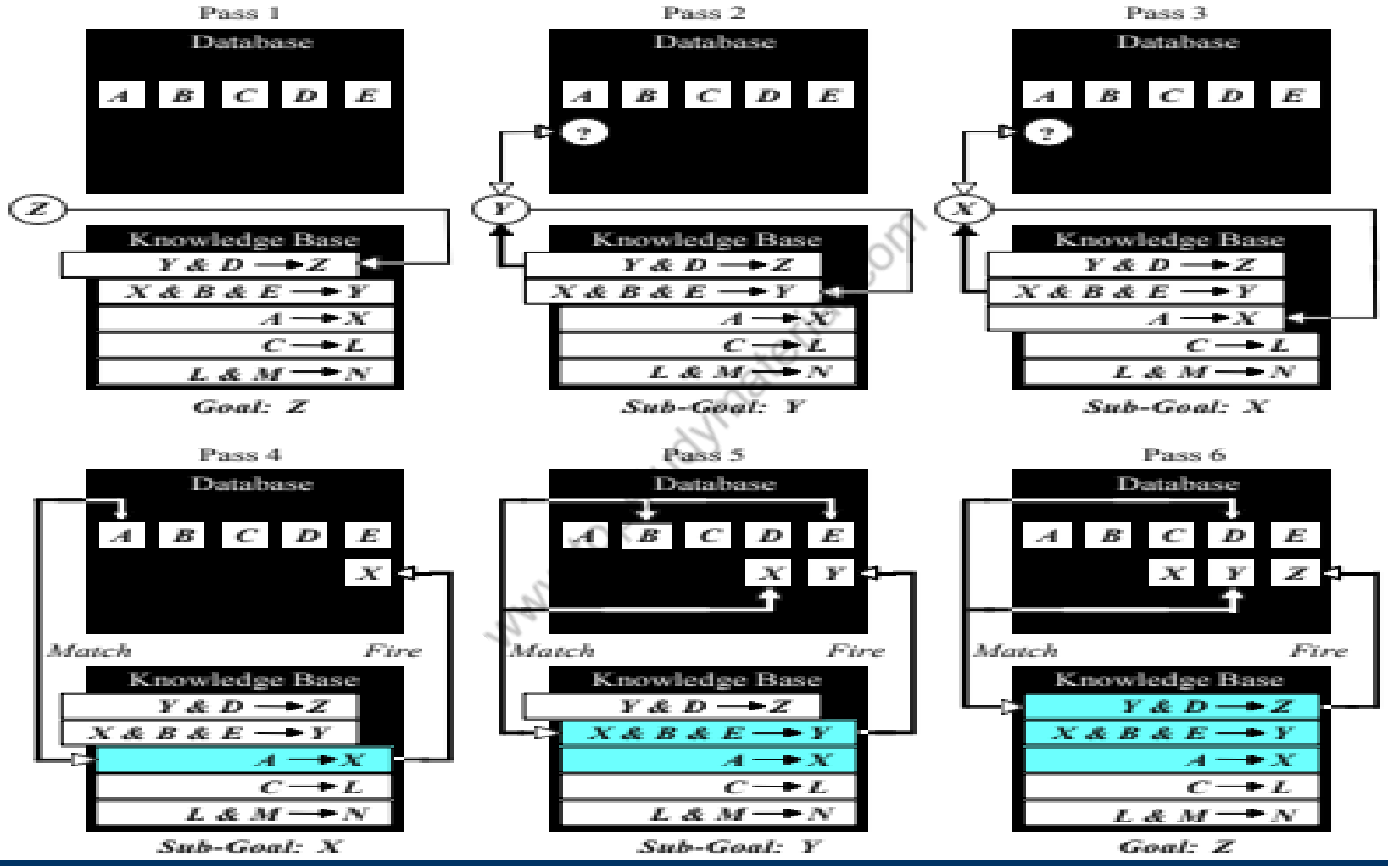
Therefore, Tom is sweating. (B)

Backward Chaining

- With this strategy, an expert system finds out the answer to the question, “Why this happened?”
- Goal Driven
- On the basis of **what has already happened,** the Inference Engine tries to find out which conditions could have happened in the past for this result.
- This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.

Backward Chaining




Pass 1
Database
A B C D E
Z
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Goal: Z
Pass 2
Database
A B C D E
?
Y
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Sub-Goal: Y
Pass 3
Database
A B C D E
?
X
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Sub-Goal: X
Pass 4
Database
A B C D E
X
Match
Fire
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Sub-Goal: X
Pass 5
Database
A B C D E
X Y
Match
Fire
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Sub-Goal: Y
Pass 6
Database
A B C D E
X Y Z
Match
Fire
Knowledge Base
Y & D → Z
X & B & E → Y
A → X
C → L
L & M → N
Goal: Z

Backward Chaining

A suitable sequence can be as follows:

- The patient has a bacterial infection.
- The patient is vomiting.
- He/she is also experiencing diarrhea and severe stomach upset.
- Therefore, the patient has typhoid (salmonella bacterial infection).

The MYCIN expert system uses the information collected from the patient to recommend suitable treatment.

The recommended treatment corresponds to the identified bacterial infection. In the case above, the system may recommend the use of ciprofloxacin.

Model Checking in Propositional Logic

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
  - *Early termination*: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}

      SAT solver can stop with partial models; no need to assign all variables (can assign arbitrarily if a complete model is needed).

    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false, B=false}

      Stop when a conflict is found. Similar to backtracking algorithm for general CSPs.

Model Checking in Propositional Logic

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
  - *Early termination*
  - *Pure symbols*: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true

Claim: If a sentence has a model to satisfy it, then it has a model in which the pure symbols are assigned values that make their literals true. Why?

W.l.o.g., assume symbol A shows up in all clauses as A. Assume there is a model satisfies the sentence with A=false. Then construct a new model with A=true and everything else the same. Since there are no opposite sign literals, making A=true that could make any clause be false.

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
  - *Early termination*
  - *Pure symbols*: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true

Note: In determining the purity of a symbol, the algorithm can ignore clauses that are already known to be true in the model constructed so far

DPLL Algorithm

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern SAT solvers
- Essentially a backtracking search over models with several tricks:
  - *Early termination*
  - *Pure symbols*
  - *Unit clauses*: A unit clause is a clause in which all literals but one are already assigned false by the model (i.e., left with a single literal that can potentially satisfy the clause). Set the remaining symbol of a unit clause to satisfy it.
    - E.g., if A=false and the sentence (in CNF) has a clause $(A \vee B)$, then set B true

      Similar to Generalized Forward Checking (nFC0) for general CSPs

    - Unit propagation: Assigning values to the symbol in a unit clause can lead to new unit clauses. Iteratively find unit clauses until no more remains.

      Similar to Constraint Propagation for general CSPs

DPLL Algorithm

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false

Early termination

P, value ←FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols–P, model∪{P=value})

P, value ←FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols–P, model∪{P=value})

P ← First(symbols)
rest ← Rest(symbols)

return or(DPLL(clauses, rest, model∪{P=true}),
DPLL(clauses, rest, model∪{P=false}))

Essentially backtracking

DPLL Algorithm

Clauses:

$\neg a \vee b \vee c$

$a \vee c \vee d$

$a \vee c \vee \neg d$

$a \vee \neg c \vee d$

$a \vee \neg c \vee \neg d$

$\neg b \vee \neg c \vee d$

$\neg a \vee b \vee \neg c$

$\neg a \vee \neg b \vee c$

Assign $a = true$

Assign $b = true$

Find unit clause $\neg a \vee \neg b \vee c$, so $c = true$

Find unit clause $\neg b \vee \neg c \vee d$, so $d = true$

Local Search Algorithms for SAT

- WALK-SAT
  - Randomly choose an unsatisfied clause
  - With probability p, flip a randomly selected symbol in the clause
  - Otherwise, flip a symbol in the clause that maximizes the # of satisfied clauses

WALKSAT

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a model or *failure*

inputs: *clauses*, a set of clauses

p, the probability of choosing to do a random walk, typically around 0.5

max_flips, number of flips allowed before giving up

model ← a random assignment of *true*/*false* to the symbols in *clauses*

for i = 1 **to** *max_flips* **do**

if *model* satisfies *clauses* **then return** *model*

clause ←a randomly selected clause from *clauses* that is *false* in *model*

with probability *p* flip the value in *model* of
a randomly selected symbol from *clause*

else flip whichever symbol in *clause* maximizes the # of satisfied clauses

return *failure*

Local Search Algorithms for SAT

- WALK-SAT
  - Randomly choose an unsatisfied clause
  - With probability p, flip a randomly selected symbol in the clause
  - Otherwise, flip a symbol in the clause that maximizes the # of satisfied clauses

- GSAT [Selman, Levesque, Mitchell AAAI-92]
  - Similar to hill climbing but with random restarts and allows for downhill/sideway moves if no better moves available

GSAT

function GSAT(*sentence*, *max_restarts*, *max_climbs*) **returns** a model or *failure*

for i = 1 **to** *max_restarts* **do**

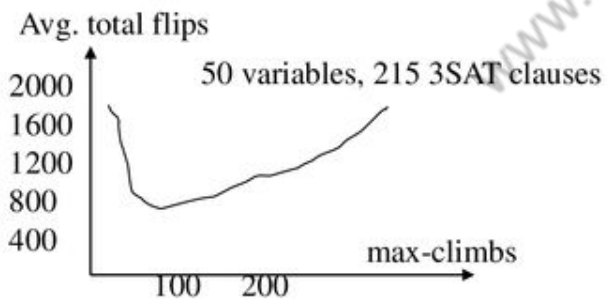
model ← a random assignment of *true*/*false* to the symbols in *clauses*

for j = 1 **to** *max_climbs* **do**

if *model* satisfies *sentence* **then return** *model*

model ← randomly choose one of the best successors

return *failure*



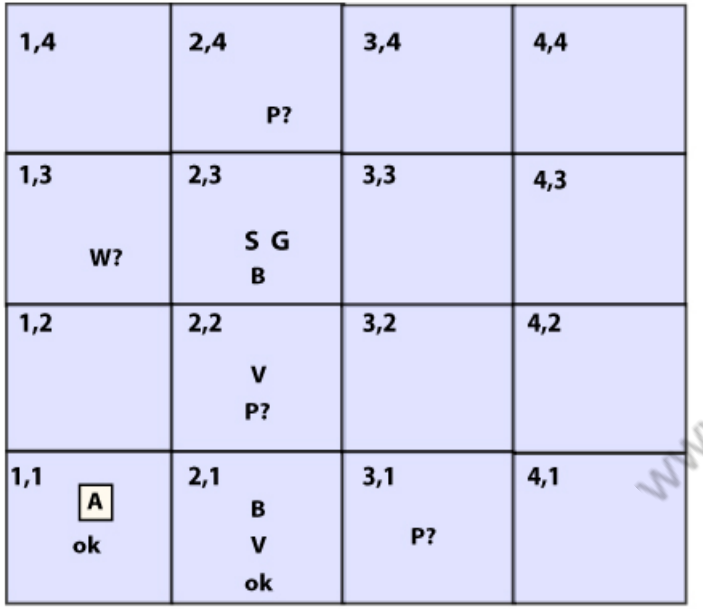
Greediness is not essential as long as climbs and sideways moves are preferred over downward moves.

Propositional Logic

Agents based on Propositional Logic

- To enable the agent to deduce, to the extent possible, the state of the world given its percept history. This requires writing down a complete logical model of the effects of actions.
- How the agent can keep track of the world efficiently without going back into the percept history for each inference.
- How the agent can use logical inference to construct plans that are guaranteed to achieve its goals

The current state of the world



Atomic proposition variable for Wumpus world:

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

The current state of the world

Some Propositional Rules for the wumpus world:

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

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

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

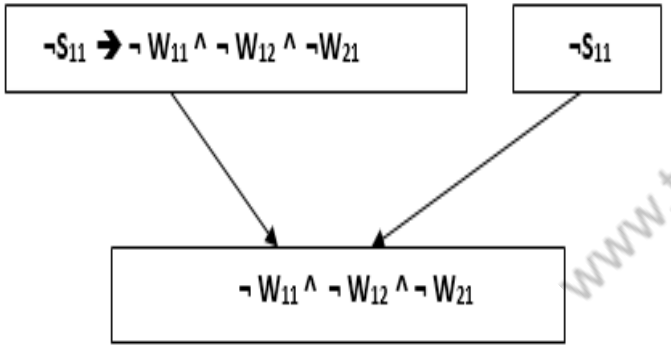
(R4) $S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

Making plans by propositional inference

Prove that Wumpus is in the room (1, 3)

We can prove that wumpus is in the room (1, 3) using propositional rules which we have derived for the wumpus world and using inference rule.

Apply Modus Ponens with ¬S11 and R1:



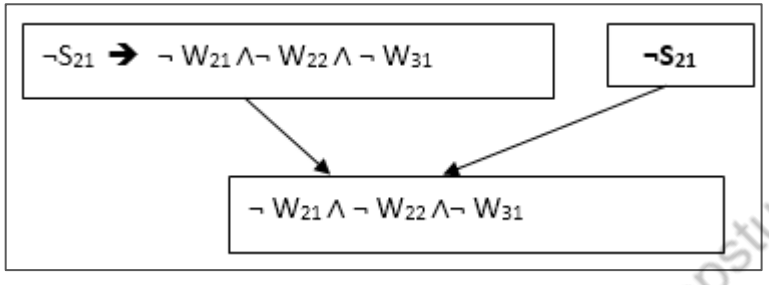
Apply And-Elimination Rule:

After applying And-elimination rule to ¬ W_{11} ∧ ¬ W_{12} ∧ ¬ W_{21}, we will get three statements:

¬ W_{11}, ¬ W_{12}, and ¬W_{21}.

Prove that Wumpus is in the room (1, 3)

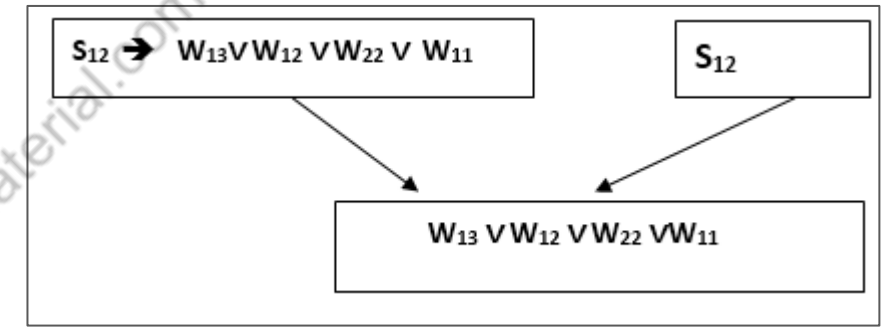
- **Apply Modus Ponens to $\neg S_{21}$, and R2:**



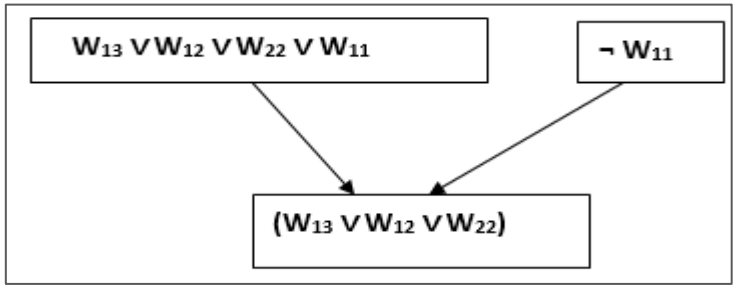
- **Apply And -Elimination rule:**

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

- **Apply MP to S_{12} and R4:**

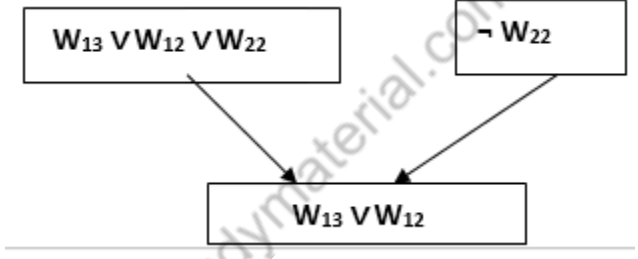


- **Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$:**

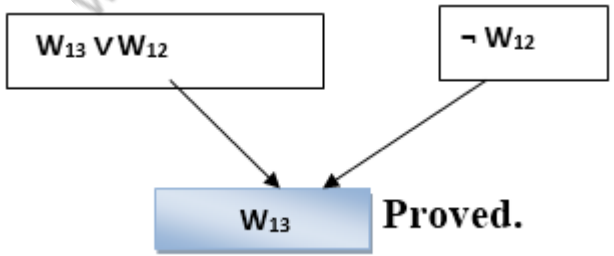


Prove that Wumpus is in the room (1, 3)

- **Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22}$ and $\neg W_{22}$:**



- **Apply Unit Resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$:**



First order Logic

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

"Some humans are intelligent", or

"Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

Example

- Express “Socrates is a man” in
- Propositional logic
  - MANSOCRATES - single proposition representing entire idea
- First-Order Predicate Calculus
  - Man(SOCRATES) - predicate representing property of constant SOCRATES

First Order Logic

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...

Syntax for First-Order Logic

Sentence → AtomicSentence
| Sentence Connective Sentence
| Quantifier Variable Sentence
| ¬Sentence
| (Sentence)
AtomicSentence → Predicate(Term, Term, ...)
| Term=Term
Term → Function(Term,Term,...)
| Constant
| Variable
Connective → ∨ | ∧ | ⇒ | ⇔
Quantifiers → ∃ | ∀
Constant → A | John | Car1
Variable → x | y | z |...
Predicate → Brother | Owns | ...
Function → father-of | plus | ...

First-Order Logic

- Term
  Anything that identifies an object
  Function(args)
  Constant - function with 0 args
- Atomic sentence
  Predicate with term arguments
  Enemies(WilyCoyote, RoadRunner)
  Married(FatherOf(Alex), MotherOf(Alex))
- Literals
  atomic sentences and negated atomic sentences
- Connectives
  (&), (v), (->), (<=>), (~)
  Quantifiers
  Universal Quantifier ∀
  Existential Quantifier ∃

First-Order Logic

- **Constant symbols** (which refer to the "individuals" in the world) E.g., Mary, 3
- **Function symbols** (mapping individuals to individuals) E.g., father-of(Mary) = John, color-of(Sky) = Blue
- **Predicate symbols** (mapping from individuals to truth values) E.g., greater(5,3), green(Grass), color(Grass, Green)
- Everyone likes someone: (Ax)(Ey)likes(x,y)
- Someone is liked by everyone: (Ey)(Ax)likes(x,y)

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

Properties of Quantifiers:

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

First Order Logic

1. **All birds fly.**

   ∀x bird(x) →fly(x).

1. **Every man respects his parent.**

   ∀x man(x) → respects (x, parent).

1. **Some boys play cricket.**

   ∃x boys(x) → play(x, cricket).

1. **Not all students like both Mathematics and Science.**

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

1. **∀x) student(x) → smart(x) "All students are smart"**
2. **(∃x) student(x) ∧ smart(x) "There is a student who is smart"**
3. **Everyone likes someone: (∀x)(∃y) likes(x,y)**
4. **Every gardener likes the sun. ∀x gardener(x) → likes(x,Sun)**
5. **You can fool some of the people all of the time.∃x ∀t person(x) ∧time(t) → can-fool(x,t)**
6. **All purple mushrooms are poisonous. ∀x (mushroom(x) ∧ purple(x)) → poisonous(x)**

Use of Quantifiers

- Universal quantification naturally uses implication:

$\forall x$ Whale(x) $\wedge$ Mammal(x)

Says that everything in the universe is both a whale and a mammal.

- Existential quantification naturally uses conjunction:

$\exists x$ Owns(Mary,x) $\Rightarrow$ Cat(x)

Says either there is something in the universe that Mary does not own or there exists a cat in the universe.

- $\forall x$ Owns(Mary,x) $\Rightarrow$ Cat(x)

Says all Mary owns is cats (i.e. everything Mary owns is a cat). Also true if Mary owns nothing.

- $\forall x$ Cat(x) $\Rightarrow$ Owns(Mary,x)

Says that Mary owns all the cats in the universe.

Also true if there are no cats in the universe.

Nesting Quantifiers

- The order of quantifiers of the same type doesn't matter

  $\forall x \forall y(Parent(x,y) \wedge Male(y) \Rightarrow Son(y,x))$

  $\exists x \exists y(Loves(x,y) \wedge Loves(y,x))$

- The order of mixed quantifiers does matter:

  1. $\forall x \exists y(Loves(x,y))$: Says everybody loves somebody, i.e. everyone has someone whom they love.

  2. $\exists y \forall x(Loves(x,y))$: Says there is someone who is loved by everyone in the universe.

  3. $\forall y \exists x(Loves(x,y))$: Says everyone has someone who loves them.

  4. $\exists x \forall y(Loves(x,y))$: Says there is someone who loves everyone in the universe.

Variable Scope

- The **scope** of a variable is the sentence to which the quantifier syntactically applies.
- As in a block structured programming language, a variable in a logical expression refers to the closest quantifier within whose scope it appears.

$\exists x\ (Cat(x) \wedge \forall x(Black\ (x)))$

The x in Black(x) is universally quantified Says cats exist and everything is black

- In a **well-formed formula** (**wff**) all variables should be properly introduced:
- $\exists x P(y)$ not well-formed
- A **ground** expression contains no variables.

Relation Between Quantifiers

- Universal and existential quantification are logically related
- to each other:
- $\forall x \neg Love(x,Saddam) \Leftrightarrow \neg \exists x\ Loves(x,Saddam)$
- $\forall x\ Love(x,Princess\text{-}Di) \Leftrightarrow \neg \exists x \neg Loves(x,Princess\text{-}Di)$
- General Identities
- $\forall x \neg P \Leftrightarrow \neg \exists x\ P$
- $\neg \forall x\ P \Leftrightarrow \exists x \neg P$
- $\forall x\ P \Leftrightarrow \neg \exists x \neg P$
- $\exists x\ P \Leftrightarrow \neg \forall x \neg P$
- $\forall x\ P(x) \wedge Q(x) \Leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
- $\exists x\ P(x) \vee Q(x) \Leftrightarrow \exists x P(x) \vee \exists x Q(x)$

Equality

- Can include equality as a primitive predicate in the logic, or require it to be introduced and axiomatized as the **identity relation**.
- Useful in representing certain types of knowledge:

1) $\exists x \exists y(Owns(Mary, x) \wedge Cat(x) \wedge Owns(Mary,y) \wedge Cat(y) \wedge \neg(x=y))$

- Mary owns two cats. Inequality needed to insure x and y are distinct.

2) $\forall x \exists y\ married(x, y) \wedge \forall z(married(x,z) \Rightarrow y=z)$

Everyone is married to exactly one person. Second conjunct is needed to guarantee there is only one unique spouse.

Higher-Order Logic

- FOPC is called **first-order** because it allows quantifiers to range over objects (terms) but not properties, relations, or functions applied to those objects.
- **Second-order** logic allows quantifiers to range over predicates and functions as well:

$\forall x \forall y [(x=y) \Leftrightarrow (\forall p\ p(x) \Leftrightarrow p(y))]$: Says that two objects are equal if and only if they have exactly the same properties.

$\forall f \forall g [(f=g) \Leftrightarrow (\forall x\ f(x) = g(x))]$: Says that two functions are equal if and only if they have the same value for all possible arguments.

Third-order would allow quantifying over predicates of predicates, etc.

For example, a second-order predicate would be Symmetric(p) stating that a binary predicate p represents a symmetric relation.

Propositional vs. Predicate Logic

- In propositional logic, each possible atomic fact requires a separate unique propositional symbol.
- If there are *n* people and *m* locations: representing the fact that some person moved from one location to another requires nm^2 separate symbols.
- Predicate logic includes a richer **ontology.**
- **Ontology:** A rigorous and exhaustive organization of some knowledge domain that is usually hierarchical and contains all the relevant entities and their relations.

Predicate logic / Ontology

- Predicate logic requires.
  - Objects (terms)
  - Properties (unary predicates on terms)
  - Relations (*n*-ary predicates on terms)
  - Functions (mappings from terms to other terms)
- Allows more flexible and compact representation of knowledge.
- Move(x, y, z) for person x moved from location y to z.

First-Order Logic: Terms and Predicates

- Objects are represented by **terms**:
  - **Constants**: Block1, John
  - **Function symbols:** father-of, successor, plus
  - An *n*-ary function maps a tuple of *n* terms to another
  - term: father-of(John), succesor(0), plus(plus(1,1),2)
- Terms are simply names for objects.
- Logical functions are not procedural as in programming languages.
- They do not need to be defined, and do not really return a value.
- Allows for the representation of an infinite number of terms.

First-Order Logic: Terms and Predicates

- Propositions are represented by a **predicate** applied to a tuple of terms.
- A predicate represents a property of or relation between terms that can be true or false:
- Brother(John, Fred), Left-of(Square1, Square2)
- GreaterThan (plus(1,1), plus(0,1))

First order Logic

In propositional logic, we can only represent the facts, which are either true or false.

PL is not sufficient to represent the complex sentences or natural language statements.

The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

"Some humans are intelligent", or

"Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

Example

- Express “Socrates is a man” in
- Propositional logic
  - MANSOCRATES - single proposition representing entire idea
- First-Order Predicate Calculus
  - Man(SOCRATES) - predicate representing property of constant SOCRATES

First Order Logic

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of

Syntax for First-Order Logic

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \\
& | & (\ \textit{Sentence Connective Sentence}\ ) \\
& | & \textit{Quantifier Variable}, \ldots\ \textit{Sentence} \\
& | & \neg\ \textit{Sentence} \\
\\
\textit{AtomicSentence} & \rightarrow & \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
& | & \textit{Constant} \\
& | & \textit{Variable} \\
\\
\textit{Connective} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{Before} \mid \textit{HasColor} \mid \textit{Raining} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{Mother} \mid \textit{LeftLeg} \mid \cdots
\end{array}
$$

First-Order Logic

- **Term**
  Anything that identifies an object
  Function(args)
  Constant - function with 0 args
- **Atomic sentence**
  Predicate with term arguments
  Enemies(WilyCoyote, RoadRunner)
  Married(FatherOf(Alex), MotherOf(Alex))
- **Literals**
  atomic sentences and negated atomic sentences
- **Connectives**
  (&), (v), (->), (<=>), (~)
- **Quantifiers**
  Universal Quantifier ∀
  Existential Quantifier ∃

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

Properties of Quantifiers:

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

First Order Logic

1. **All birds fly.**

   ∀x bird(x) →fly(x).

1. **Every man respects his parent.**

   ∀x man(x) → respects (x, parent).

1. **Some boys play cricket.**

   ∃x boys(x) → play(x, cricket).

1. **Not all students like both Mathematics and Science.**

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

1. **∀x) student(x) → smart(x) "All students are smart"**
2. **(∃x) student(x) ∧ smart(x) "There is a student who is smart"**
3. **Everyone likes someone: (∀x)(∃y) likes(x,y)**
4. **Every gardener likes the sun. ∀x gardener(x) → likes(x,Sun)**
5. **You can fool some of the people all of the time.∃x ∀t person(x) ∧time(t) → can-fool(x,t)**
6. **All purple mushrooms are poisonous. ∀x (mushroom(x) ∧ purple(x)) → poisonous(x)**

Use of Quantifiers

- Universal quantification naturally uses implication:

$\forall x$ Whale(x) $\wedge$ Mammal(x)

Says that everything in the universe is both a whale and a mammal.

- Existential quantification naturally uses conjunction:

$\exists x$ Owns(Mary,x) $\Rightarrow$ Cat(x)

Says either there is something in the universe that Mary does not own or there exists a cat in the universe.

- $\forall x$ Owns(Mary,x) $\Rightarrow$ Cat(x)

Says all Mary owns is cats (i.e. everything Mary owns is a cat). Also true if Mary owns nothing.

- $\forall x$ Cat(x) $\Rightarrow$ Owns(Mary,x)

Says that Mary owns all the cats in the universe.Also true if there are no cats in the universe.

Nesting Quantifiers

- The order of quantifiers of the same type doesn't matter

  $\forall x \forall y(Parent(x,y) \wedge Male(y) \Rightarrow Son(y,x))$

  $\exists x \exists y(Loves(x,y) \wedge Loves(y,x))$

- The order of mixed quantifiers does matter:

  1. $\forall x \exists y(Loves(x,y))$: Says everybody loves somebody, i.e. everyone has someone whom they love.

  2. $\exists y \forall x(Loves(x,y))$: Says there is someone who is loved by everyone in the universe.

  3. $\forall y \exists x(Loves(x,y))$: Says everyone has someone who loves them.

  4. $\exists x \forall y(Loves(x,y))$: Says there is someone who loves everyone in the universe.

Relation Between Quantifiers

- Universal and existential quantification are logically related
- to each other:
- $\forall x \neg Love(x,Saddam) \Leftrightarrow \neg \exists x\ Loves(x,Saddam)$
- $\forall x\ Love(x,Princess\text{-}Di) \Leftrightarrow \neg \exists x \neg Loves(x,Princess\text{-}Di)$
- General Identities
- $\forall x \neg P \Leftrightarrow \neg \exists x\ P$
- $\neg \forall x\ P \Leftrightarrow \exists x \neg P$
- $\forall x\ P \Leftrightarrow \neg \exists x \neg P$
- $\exists x\ P \Leftrightarrow \neg \forall x \neg P$
- $\forall x\ P(x) \wedge Q(x) \Leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
- $\exists x\ P(x) \vee Q(x) \Leftrightarrow \exists x P(x) \vee \exists x Q(x)$

Inference in First order logic

In First-Order Logic, inference is used to derive new facts or sentences from existing ones.

Substitution:

Substitution is a basic procedure that is applied to terms and formulations. It can be found in all first-order logic inference systems. When there are quantifiers in FOL, the substitution becomes more complicated. When we write F[a/x], we are referring to the substitution of a constant "a" for the variable "x."

Equality:

In First-Order Logic, atomic sentences are formed not only via the use of predicate and words, but also through the application of equality. We can do this by using equality symbols, which indicate that the two terms relate to the same thing.

Example: Brother (John) = Smith.

In the above example, the object referred by the Brother (John) is close to the object referred by Smith. The equality symbol can be used with negation to portray that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

Inference in First order logic

FOL inference rules for quantifier:

First-order logic has inference rules similar to propositional logic, therefore here are some basic inference rules in FOL:

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

TABLE 2 Rules of Inference for Quantified Statements.

Rule of Inference	*Name*
$\forall x P(x)$ $\therefore P(c)$	Universal instantiation
$P(c)$ for an arbitrary c $\therefore \forall x P(x)$	Universal generalization
$\exists x P(x)$ $\therefore P(c)$ for some element c	Existential instantiation
$P(c)$ for some element c $\therefore \exists x P(x)$	Existential generalization

Let $P(x)$ be a predicate in a universe U. It will be helpful for Exercise to recall that **universal instantiation** is the formal rule which tells us that

$$(\forall x)P(x) \implies P(a) \text{ for any particular } a \in U$$

and that **universal generalization** is the formal rule which tells us that

$$P(a) \text{ for an arbitrary } a \in U \implies (\forall x)P(x)$$

Further recall that **existential instantiation** is the formal rule which tells us that

$$(\exists x)P(x) \implies P(a) \text{ for some particular } a \in U$$

where we further require that a is a *name that has not been used yet* (otherwise, we risk commiting the fallacy of *equivocation*). Finally, recall that **existential generalization** is the formal rule which tells us that

$$P(a) \text{ for some particular } a \in U \implies (\exists x)P(x).$$

Inference in First order logic

Universal Generalization

- Universal generalization is a valid inference rule that states that if premise P(c) is true for any arbitrary element c in the universe of discourse, we can arrive at the conclusion x P. (x).
- It can be represented as

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

- If we want to prove that every element has a similar property, we can apply this rule.
- x must not be used as a free variable in this rule.

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

Inference in First order logic

Universal Instantiation:

A valid inference rule is universal instantiation, often known as universal elimination or UI.

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

Let's take a example,
"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL: ∀x king(x) ∧ greedy (x) → Evil (x),
We can infer any of the following statements using Universal Instantiation from this information:

- King(John) ∧ Greedy (John) → Evil (John),
- King(Richard) ∧ Greedy (Richard) → Evil (Richard),
- We can infer any phrase by replacing a ground word for the variable, according to UI
- King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),

Inference in First order logic

Existential Instantiation:

- Existential instantiation is also known as Existential Elimination, and it is a legitimate first-order logic inference rule.
- It can only be used to replace the existential sentence once.

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 known as Skolem constant.
- The Existential instantiation is a special case of Skolemization process.

Inference in First order logic

Existential introduction

- An existential generalization is a valid inference rule in first-order logic that is also known as an existential introduction.
- This rule argues that if some element c in the universe of discourse has the property P, we can infer that something in the universe has the attribute P.
- Example: Let's say that, **"Priyanka got good marks in English."** **"Therefore, someone got good marks in English."**

Inference in First order logic

FOL inference rules for quantifier:

Universal generalization is a valid inference rule that states that if premise P(c) is true for any arbitrary element c in the universe of discourse, we can arrive at the conclusion x P. (x).

It can be represented as:

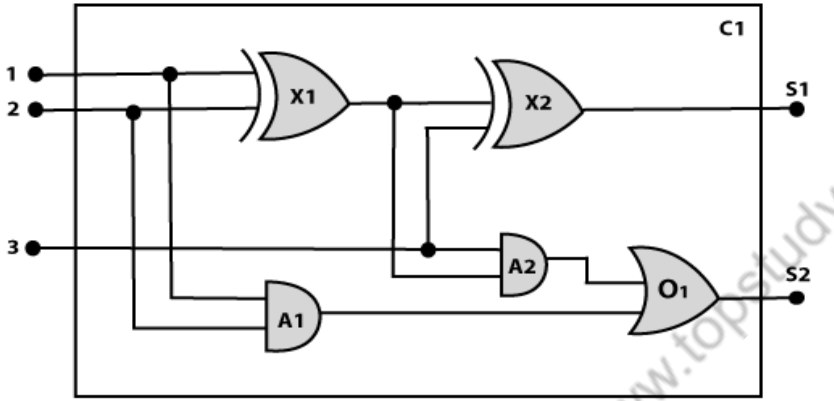
$$\frac{P(c)}{\forall x\, P(x)}$$

- If we want to prove that every element has a similar property, we can apply this rule.
- x must not be used as a free variable in this rule.

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

Knowledge Engineering in First-order logic

The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering.



1. Identify the task:

- Does the circuit add properly?
- What will be the output of gate A2, if all the inputs are high?
- Which gate is connected to the first input terminal?
- Does the circuit have feedback loops?

Knowledge Engineering in First-order logic

2. Assemble the relevant knowledge:

In the second step, 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 vocabulary:

- The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.
- The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**.
- Circuits will be identified by a predicate: **Circuit (C1)**.
- For the terminal, we will use predicate: **Terminal(x)**.
- For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.
- The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.
- 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:

  ∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2).

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

  ∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.

5. Encode a description of the problem instance

The given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be

1. For XOR gate: Type(x1)= XOR, Type(X2) = XOR
2. For AND gate: Type(A1) = AND, Type(A2)= AND
3. For OR gate: Type (O1) = OR.

Knowledge Engineering in First-order logic

6. Pose queries to the inference procedure and get answers

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?

1. ∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3 ∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.