These notes are not complete. Please visit https://tutorialsduniya.com/notes for updates in these notes.

Contributor: Riya Goel [KMV CS(H)]

Visit https://www.tutorialsduniya.com for Compiled books, Notes, books, programs, question papers with solutions etc.

Facebook: https://www.facebook.com/tutorialsduniya
Youtube: https://www.youtube.com/user/TutorialsDuniya
LinkedIn: https://www.linkedin.com/company/tutorialsduniya
Prolog - language used in practicals.

Programming in logic?

Science through which we make our machines intelligent so that they behave like humans.

Artificial intelligence.

Basic idea is initial thinking process?

1. Knowledge representation - ways to provide initial knowledge to machine.

2. Natural language processing - The language used by all parties (client, server) must be compatible.

3. Logic building - We have certain rules to build logic. Without knowledge, a system can't be intelligent. E.g., distributive law, transitive law.

4. Adoptability to current environment - The machine adopts to changing situations, using sensors (e.g., camera of robots), activators.

Artificial - something which we create by ourselves.
Scheme on which AI is based:

<table>
<thead>
<tr>
<th>Thinking (humanly)</th>
<th>Acting (machine)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinking (machine)</td>
<td>Acting humanly</td>
</tr>
</tbody>
</table>

Note:

First test in AI was initiated by Alan Turing

Turing Test (no visual contact between A & B)

(test incorporating intelligence)

With any visual contact, if A is able to distinguish whether the system/machine is human or machine, then AI test fails.

If A can't distinguish whether B is human or machine, then AI test is successful.

A → Interrogates B. Basic test to test AI.

Total Turing Test?

When we have a visual representation (e.g., cameras)
among A & B, then it becomes a total
turing test.

* **Captcha** → used to check intelligence.

{ **soft bot** } → internally robots but structure is like humans.

used to differentiate how much good is our intelligence.

Captcha are provided in combination of no; letters, words for good websites.

provides combination of random states.

* **Agent** - work on AI, sensed acts on a given environment

The computer must possess the following capabilities to behave intelligently:

1. Natural language processing.

2. Knowledge representation.

3. Automated reasoning (feeding logic)

4. Machine Learning (remembering the logic)

Agent → program that behaves by sensing a particular condition.
(broad category).

AI

Machine Learning

Deep Learning

To know behaviour of an agent, we need to know behaviour of a national agent.

Ideal agent (produces ideal behaviour)
Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from [https://www.tutorialsduniya.com](https://www.tutorialsduniya.com)

- C and C++
- Programming in Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- DataBase Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- Programs: [https://www.tutorialsduniya.com/programs](https://www.tutorialsduniya.com/programs)
- Question Papers: [https://www.tutorialsduniya.com/question-papers](https://www.tutorialsduniya.com/question-papers)
- Python Notes: [https://www.tutorialsduniya.com/python](https://www.tutorialsduniya.com/python)
- Java Notes: [https://www.tutorialsduniya.com/java](https://www.tutorialsduniya.com/java)
- JavaScript Notes: [https://www.tutorialsduniya.com/javascript](https://www.tutorialsduniya.com/javascript)
- JSP Notes: [https://www.tutorialsduniya.com/jsp](https://www.tutorialsduniya.com/jsp)
- Microprocessor Notes: [https://www.tutorialsduniya.com/microprocessor](https://www.tutorialsduniya.com/microprocessor)
- OR Notes: [https://www.tutorialsduniya.com/operational-research](https://www.tutorialsduniya.com/operational-research)
Ch-2

Sensors & activators

Intelligent Agent perceives & acts on things.

- Percept: basically a sensing process, i.e., sensing the environment and building up a perception and an action is built.

When we gain multiple percepts, it becomes percept sequence.

- History of percepts gained throughout life.

- Function: partial implementation.
- Program: complete implementation.

Combining multiple percepts → agent function

And when we combine multiple agent functions, we get an agent program.

Agent senses environment with sensors then acts with activators.
Agent as vacuum cleaner?

We have 2 environments (A and B) with dust particles.

If A → dirt ⇔ Clean action

percept for vacuum cleaner:

A → clean ⇔ move down
B → dirt ⇔ clean.

Each agent here has a specific task to perform so we didn't make a generic agent.

When we talk of a percept and a percept sequence, then we get a specific agent.

eg: for an automated car

Environment - roads (artificial/natural)
Activators - brakes, accelerators
Sensors - cameras
Performance measure - mileage.

Contributor: Riya Goel
Agent depends on environment.

P A S E

environment

agents performance action sensors

values

defines characteristics of agent.
Software - Prolog

?- \[\text{blink (system is ready to be questioned)}\]

Prolog - deals with sentences and converts into high-level language.
- It is a high-level language
- Prolog defines relationship between objects.

  \[\text{eq - relation between father and son.}\]

- Finding relationship and objects in a sentence -

Basic sentence:
\[x \text{ is a father of } y\]

\[\text{Ram, Shyam.}\]

relationship between \(x\) and \(y\).

- Define relationship name starting with small letter

\[
father (x, y)\]

\[
\text{dot is end of clause name of particular statement}\]

\[
\exists x \text{ and } y \text{ are objects}\]

\[\iff father (\text{Ram, Shyam})\]

Contributor: Riya Goel
Prolog works on:

1. Facts: universal truths. No need for predefined objects to check its validity.
   - Who are grandchildren of Ram?
   - Relationships already defined is in terms of parent only.

2. Questions can be in terms of facts & rules.

   \[
   \begin{align*}
   x & \rightarrow y \\
   y & \rightarrow z
   \end{align*}
   \]

   We have clubbed some or more sentences, then finding out other rules, using if and else condition to our facts.

   ? - [blink]

1. First step: add knowledge

   File

   Consult

   New

   File pl

   Oval Icon.

   Write in notepad and save with extension .pl.

Contributor: Riya Goel
Notepad

file.pl

father (ram, shyam). // clause

This is predefined knowledge that
system will use.

2. System (questions are always asked from
command prompt)

?- []
father (ram, shyam) // returns True

True

?- father (ram, m)

false

Knowledge Base is predefined information
fed in system.

2 languages for AI

LISP

Prolog

(programming in logic)

Notepad

file.pl

- father (ram, shyam).
- father (sohan, mohan).
- father (par, shyam).

Question: (1) whether Ram is father of Shyam or not.
Simply define relationships with entities.

\[ \text{Who is child of Ram.} \]

**Note:**
Use capital letters for defining variables in Prolog and small letter for objects.

- \[ \text{father (}X, \text{ Shyam)} \]
  - \( \text{capital letter} \)
  - \( \text{read as \_ who is father of Shyam} \)

Output \( \rightarrow X = \text{Ram} \) // can display Ram or bar
  - but first display relationship.

**Note:**
To display all values of \( X \), use `;`

- \[ \text{after execution: } X = \text{ram} \]
  - press `;`\[ X = \text{bar} \]

- \( \text{trace} \rightarrow \text{Graphical Trace (for debugging)} \)
  - \( \text{trace, father (}X, \text{ Shyam)} \)
    - \( \text{variable object} \)
    - \( \text{father (} \text{ram, Shyam} \) \)

- \( \text{trace} \rightarrow \text{command prompt debugging} \)
father (ram, shyam). $\Rightarrow$ terminate.

* Notebook:

father (ram, shyam).
father (par, shyam).

parent (ram, shyam).
parent (par, shyam).

// Ram’s parent of shyam.

grandparent (x, z) :-
parent (x, y),
parent (y, z) ;

If generating a new fact using two predefined facts then that new fact is not fact it is defined as a predicate in terms of rule. In fact, can’t be defined.

First program -> define family relationships and define rule.

grandparent (x, z) :- parents (x, y),
parent (y, z).

sequence matters?

We can define 1 or 2 objects entities directly.
for multiple objects, we use predicate.

- command

grandparent (sita, ram). // false.
grandparent (ram, sita). // True.

father (ram)

Error // 1 (no. of argument)
// 2 (need 2 arguments)
<table>
<thead>
<tr>
<th>Subject</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>C and C++</td>
<td>Computer System Architecture</td>
</tr>
<tr>
<td>Programming in Java</td>
<td>Discrete Structures</td>
</tr>
<tr>
<td>Data Structures</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>Computer Networks</td>
<td>Algorithms</td>
</tr>
<tr>
<td>Android Programming</td>
<td>DataBase Management Systems</td>
</tr>
<tr>
<td>PHP Programming</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Theory of Computation</td>
</tr>
<tr>
<td>Java Server Pages</td>
<td>Operational Research</td>
</tr>
<tr>
<td>Python</td>
<td>System Programming</td>
</tr>
<tr>
<td>Microprocessor</td>
<td>Data Mining</td>
</tr>
<tr>
<td>Artificial Intelligence</td>
<td>Computer Graphics</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Data Science</td>
</tr>
</tbody>
</table>

- Programs: [https://www.tutorialsduniya.com/programs](https://www.tutorialsduniya.com/programs)
- Question Papers: [https://www.tutorialsduniya.com/question-papers](https://www.tutorialsduniya.com/question-papers)
- Python Notes: [https://www.tutorialsduniya.com/python](https://www.tutorialsduniya.com/python)
- Java Notes: [https://www.tutorialsduniya.com/java](https://www.tutorialsduniya.com/java)
- JavaScript Notes: [https://www.tutorialsduniya.com/javascript](https://www.tutorialsduniya.com/javascript)
- JSP Notes: [https://www.tutorialsduniya.com/jsp](https://www.tutorialsduniya.com/jsp)
- Microprocessor Notes: [https://www.tutorialsduniya.com/microprocessor](https://www.tutorialsduniya.com/microprocessor)
- OR Notes: [https://www.tutorialsduniya.com/operational-research](https://www.tutorialsduniya.com/operational-research)
- **Rational Agent**: Ideal behavior shown by an agent. If the outcome of an environment is known by an agent, it is a rational agent.

The agent which implements things in terms of designs, etc. correctly, with reference to a performance measure.

- **To implement performance measure**:

1. **Environment**: The location in the real world for which an agent is defined. Without environment, an agent cannot act.

First thing of agent on a new environment is to perceive things through its sensors.

- Perception sequence mapping of process to corresponding action

- **Types of Environment**

- **Categories / properties of task environment**: (Accessible vs. Inaccessible)

- (a) Fully Observable

- (b) Partly Observable

As for each environment, we have dedicated agent.

If an agent's sensor gives it access to the complete state of the environment
at each pt. in time, then this is known as fully observable environment, whereas in partially observable environment, an agent might not be able to keep the track of all the changes because of noisy or inaccurate sensors. A vacuum agent is an example of fully observable agent & automated taxi driving is an example of partially observable environment.

Vacuum cleaner has only 2 stages - room & dirt, then it continuously captures changes.

When a car in front of us, suddenly applies brakes & dirt, then it might not be able to capture these changes & movement cannot be captured by sensors.

(b) Deterministic Vs Stochastic :-

(determined beforehand on basis of previous knowledge)

If next state of the environment is completely determined by the current state and action executed by the agent, then this comes under "deterministic task environment" otherwise it will be "stochastic".

Next state = Current State + work by agent.

eg: automated taxi driving is a
Stochastic process

e.g.- vacuum cleaner & crossword are deterministic process.

(c) episodic v/s sequential

Flow maintained

(at random).

diff. episodes are based on diff. themes.

An episodic environment the agents experience is divided into atomic episodes. Each episode consists of the agent's perception and the corresponding single action, whereas in the sequential task environment, it doesn't depend on a single performed action.

(d) Static v/s Dynamic

While

of the environment can change an agent is deliberately doing something. It comes under the dynamic environment otherwise the environment is static. The action done by an agent causes a change in environment. An environment becomes dynamic.

e.g.- Taxi driving as performed action of 1 car changes environment for other cars. Also, diff. dir of roads.
eg - cross word is a static process. (environment not changing).

(e) discrete v/s continuous -

In a discrete environment, the percepts and the actions are not moving acc. to the time whereas in continuous environment, both things are moving parallelly.

perceive → perform → perceive → perform →

continous (action acc. to only talking of time frame, no use of each moment single or multiple tasks. of time, there must be immediate action & percept.

eg:- applying brakes if driver in front of us applies brakes (in terms of reflex actions).

eg:- google assistant.

(f) single v/s multiagent

Single agent multiple agents in one
in one environment eg - taxi driving.

eg - vacuum cleaner

(automated)
PEAS task environment

P - Performance measures
E - Environment
A - Activators
S - Sensors

Table of PEAS
(Read book)

PEAS for a Taxi driver

Agent Type - Taxi driver

Performance measures - (1) Safety
(2) Fast
(3) Legal (corresponds to licenses and other protocols)
(4) Comfortable trip
(5) Max. mileage
(6) Reach to the destination

Environment - (1) Roads
(2) Traffic
(3) Pedestrians

Activators - (1) Steering
(2) Brakes
(3) Accelerator
(4) Horns
(5) Lights (front & back)

Sensors - (1) Camera
(2) Speedometers
(3) GPS
(4) Odometers
(5) Engine sensors.

- PEAS for Vacuum Cleaner

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Performance</th>
<th>Environment</th>
<th>Activators</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum Cleaner</td>
<td>Suction power</td>
<td>Room, Dirt</td>
<td>Clean, Dirt</td>
<td>Cameras</td>
</tr>
<tr>
<td></td>
<td>Electricity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consumption</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Structure of Agent

Agent = Hardware + Software (programs according to the environment, depending on the nature of the agent and environment)

* Models to develop structure of an agent:

1. Simple Reflex model - (based on degree of perceived intelligence and capability)

Block Diagram

eq. 6: vacuum cleaner.
If A → dirt = clean.
After analysing what we sense, an if else condition is mapped which generates an action.

(2) Model Based Reflex Agents

As soon as what is performed, it maps to both options and acts according to if and else.

Is action already stored and based on that action is performed? If so, action depends on both if else and internal state.
If an agent is for cleaning, we see how much effective it is in cleaning dust & whether agent positively grows in environment or negatively grows. and thus happiness can be determined.

First a decision is taken & then we satisfy happiness & we keep on going through this cycle until happiness increases.

Goal Based models (base of programming) target to be achieved.

In all the 3 models, we haven’t checked about the goal achieved but in goal based models, we check whether the agent moves positively to contributing the environment.
NOTE: In all these models, agent cannot exist as there is one process which is lacking i.e., learning process.

To check worth of an action, we have a process

Performance element/critics is used to check whether an action is correct or not?
Learning Agent

Agent = hardware + software.

models to build Agent:

1. Simple Reflex
2. Model Based
3. Goal based
4. Utility based

Learning agents refer to learning how to carry out tasks for reducing time to carry out those tasks.

Percept

Continuous evaluation

Performance elements

modify change

Knowledge

Learning element

Critic

Problem generator

{ based on 4 models }

First of all, environment generates percept and then it goes to performance element where its action is decided by the 4 models.

Critic continuously evaluates this action with corresponding standards.

Contributor: Riya Goel
If action is not accurate, critics will inform learning element which in turn triggers the performance element to change the action.

- Problem generator → possible ways to incorporate same things

{performance element can consult this and find out another way?}

4 main components:

Critics

- Problem generator
- Performance element
- Learning element

Note:
- Each model has some function
  percept → argument to a function,

- Problem generator provides us with all the possible solutions.

function TABLE-DRIVEN-AGENT(percept)

returns an action

Static:

percept or sequence, initially empty table,

a table of actions, indexed by percepts sequence

Initially fully specified.

append percept to end of percept action —
lookup (percept, table)
return action.

Problem: if data is increasing, percept is changing continuously with time.

let P be possible percepts, and let T be the lifetime of agent. no. of percepts?

$$\sum_{t=1}^{T} P(t)$$

no. of entries in lookup table.

From book:

30 frames per sec., 640 x 480 pixels with 24
this gives a lookup table with over
$$10^{250,000,000,000}$$ entries for an hour's driving.

? see fig 2.9 - functions of all models?

- Practical (Prolog):

  operators:

  • / - commenting
  • min/ max - built-in words for binary comparison or more
  • compare - comparison
    returns 3 type of values:
      inferior, equal, superior


- \( = =, \ 1 = \) - equality, inequality.
- \( \text{is} \rightarrow \text{variable assignment} \)
- \( \text{strings only? for computation purpose?} \)
- \( \rightarrow \) numerical values and as well as strings.
  - Declaration?
  - assigning body in head

- \( = =, \ 1 = \) - string equality

- **functions of list**

  - `number`
  - `last`
  - `append`
  - `length`
  - `remove`
  - `min/max`
  - `sort/sublist`

  - `mod` (for remainder division)
  - `modulus` (modulo of 3/2/6)

  - `a := b, c` - end of statement.

- Head | Body

  - `program consists of procedure definitions`...

  - multiple clauses are separated by commas.

  - Evaluation & from LHS to RHS i.e. first `be` is evaluated and then c
A procedure is a resource of evaluating something.

\[ a \leftarrow b, c \quad \text{or} \quad a \leftarrow b \land c \]

? a is true if b is true and c is true?

This is a program clause with head 'a' and body b, c.

\[ ? \rightarrow a, d, e \quad \text{// initial query} \]

\[ ? \rightarrow \text{signifies a query.} \]

- multiple answers \rightarrow directed backtracking.

? - happy(chris), likes(chris, bob).

\[ \rightarrow \text{sum of 2 nos.} \]

? - \( x, y \in (2, 3) \).

This won't work.

We need to write reading and writing statements separately.

? \( x, y \) - enter and.

\[ xy \in \text{read}(x), \text{read}(y), \text{write}(x+y). \]
\[ \begin{align*}
\text{some variable result} \quad \text{xyz} (2, 3, k) \\
\text{xyz} (x, y, z) & \leq \text{z} \leq x+y.
\end{align*} \]
Searching Techniques

Knowledge representation

logic | Knowledge to be fed in computer (agent).

Book → P.W Peterson
    (Ch-4, 5, 6, 7)

Ch-4
Knowledge representation

1. Proposition Logic (PL)
2. First order predicate logic (FOPL)

(meaningful proposition → contains atomic meaning
atomic sentences that can’t be broken).

eg - It is raining // atomic sentence can’t be further broken.

→ To represent atomic sentences, we have notation in AI

KR (Knowledge representation)

Symbols R, X, Y, S

X = It is raining

Symbol assignment may vary.

atomic sentences which are independent of any action can be
denoted by single capital letter.

To show dependent sentences

\[ \text{use functions} \]
\[ R(x) \]
\[ R(x) \]

eq: it is raining in New York
can't be represented using only \( x \)

\[ x \rightarrow \text{raining} \]
\[ N \rightarrow \text{New York} \]

[dependency/relationship changes acc. to diff. junct. used]

\[ N(x) \]
\[ X(N) \]

2 possibilities.

\[ A \rightarrow B \]
\[ B \rightarrow C \]
\[ A \rightarrow C \]

New proposition.

Note:

We can't define functions and quantities.

We have only atomic sentences, but 2 or more atomic sentences can be combined to form a compound sentence.
To join two atomic sentences:

\[ \land \rightarrow \text{And} \]
\[ \lor \rightarrow \text{OR} \]
\[ \neg \rightarrow \text{Not (Unary connector)} \]
\[ \rightarrow \rightarrow \text{Implication} \]
\[ \leftrightarrow \rightarrow \text{if and only if} \]

While writing any sentence, we must keep in mind:

- **Syntax**
  - \( \text{sentence in a meaningful sequence} \)

- **Semantics**
  - \( \text{meaningful sentences} \)

Example:
- It is raining \( \Leftrightarrow \) Valid sentence
- As we are able to get full information and it is meaningful.

But the T/F value isn't determined & hence is not taken in consideration.
(Dealing only with validity & not the truth value)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A \land B</th>
<th>A \lor B</th>
<th>A \rightarrow B</th>
<th>A \leftrightarrow B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Note:
- 'All' can't be denoted by PL logical reference rules.
- \( \text{it is a part of POPL} \)
Given a representation \[(P \land \sim Q) \rightarrow R\] check its validity.

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>\sim Q</th>
<th>R</th>
<th>P \land \sim Q</th>
<th>P \land \sim Q \rightarrow R</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

- Properties of Statement -

1. **Satisfiable** - A statement is satisfied if there is some interpretation for which it is true.
   - Final column of truth table has at least one T.

2. **Contradiction** - A sentence is contradictory if there is no interpretation for which it is true.
   - Opp. of Satisfiable.

3. **Valid** - A sentence is valid iff. it is true for every interpretation (it is also known as tautology).
(4) **Equivalence** - Two sentences are equivalent if they have the same truth value under every interpretation.

(5) **Logical Consequence** - A sentence is a logical consequence of another if it is satisfied by all interpretations which satisfy the first one.

\[(P \land Q)\]

P is logical consequence of P and Q.

\[\text{I.e. if } P \land Q \text{ is valid, } P \text{ is valid if } P \land Q \text{ is contradictory, } P \text{ is contradictory.}\]

* **Inference Rules** (Equivalent Rules)

1. \[P \lor P = P \quad \text{\textit{Idempotent Law}}.\]
   \[P \land P = P\]

2. \[(P \lor Q) \lor R = P \lor (Q \lor R) \quad \text{\textit{Associativity}}.\]
   \[(P \land Q) \land R = P \land (Q \land R)\]

3. \[P \lor Q = Q \lor P \quad \text{\textit{Commutativity}}.\]
   \[P \land Q = Q \land P\]
   \[P \leftrightarrow Q \leftrightarrow P\]

4. \[P \lor (Q \lor R) = (P \lor Q) \lor (P \lor R) \quad \text{\textit{Distribution}}.\]
   \[P \lor (Q \land R) = (P \lor Q) \land (P \lor R)\]

5. \[\neg (P \lor Q) = \neg P \land \neg Q \quad \text{\textit{De Morgan’s}}.\]
   \[\neg (P \land Q) = \neg P \lor \neg Q \quad \text{\textit{Law}}.\]

Contributor: Riya Goel
\[ p \rightarrow q = \sim p \lor q \] \text{ conditional.}\\
\[ p \leftrightarrow q = (p \rightarrow q) \land (q \rightarrow p) \] \text{ biconditional.}\\

\text{Proof:}\\
\begin{array}{cccccccc}
p & q & p \rightarrow q & q \rightarrow p & p \rightarrow q \land q \rightarrow p & p \leftrightarrow q \\
T & T & T & T & T & T \\
T & F & F & T & F & T \\
F & T & T & T & F & F \\
F & F & F & F & F & T \\
\end{array}

\text{equal}

* Inference Rules of PL*

(1) Modus Ponens

\[ p \]

\[ p \rightarrow q \]

\[ q \]

(2) Chain Rule

\[ p \rightarrow q \text{ and } q \rightarrow r \]

\[ p \rightarrow r \]

(3) Simplification

\[ p \land q , \text{ Infer } p \]

(4) Conjunction

\[ p \text{ and } q , \text{ Infer } p \land q \]

(5) Transposition

\[ p \rightarrow q , \text{ Infer } \sim q \rightarrow \sim p \]

\[ \text{Monkey-Banana Problem} \]

\[ \text{famous AI problem} \]
First order predicate logic (FOPL)

building new things with help of facts.

1. PL is not sufficient for connecting multiples sentences
2. PL doesn't represent quantity (for all, some)

drawbacks of PL overcome by FOPL

FOPL offers keywords of quantity

\( \text{(All, Some)} \)

\{ All, Not All \} \quad \{ For Some, For Not Some \}

Quantifiers

\( \forall \) (for all) \quad \exists \) (existential quantifier)

* Syntax of FOPL - (1) Connectors (same in PL)
(2) **Quantifiers**

Universal quantifier

Existential quantifier (for some)

(3) **Constants**

- Use of small letters/letters + numbers
  - *e.g.* 101, John, al23

(4) **Variables**

- Denoted with help of words and capital letters are used

(5) **Functions** - Symbols that denote the relationship defined on a Domain 'D'

- *e.g.* All employees of company are at programme

(6) **Predicates** - Predicate symbols are used to denote relations or functional mapping from the element of a domain D to the values True / False. Capital letters or capital words are used to represent predicates.

Constants, variables, and functions are referred as "terms" and predicates are referred as "atomic formulas"
(1) All employees earning $1400 or more per year pay taxes.

Quantifier: All

(2) Some employees are sick today.

Well-formed formula (WFF): \( \forall x (E(x)) \rightarrow S(x) \)

(3) No employee earns more than the President.

\[ \forall x (\lnot (E(x) \land GE(\ell(x), 1400))) \rightarrow T(x). \]

\( x \) is an employee. Income value to be compared.

Both conditions must be satisfied.

If(\( \ell(x), 1400 \) -> represents income of employee greater than 1400.

\[ \forall x y (E(x) \land P(y)) \rightarrow GE(\ell(x), \ell(y)). \]

Can't use same variable for employee and president.

We need to define a function where income of employee < income of president.
TutorialsDuniya.com

Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from https://www.tutorialsduniya.com

- C and C++
- Programming in Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- Database Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- Compiled Books: https://www.tutorialsduniya.com/compiled-books
- Programs: https://www.tutorialsduniya.com/programs
- Question Papers: https://www.tutorialsduniya.com/question-papers
- Python Notes: https://www.tutorialsduniya.com/python
- Java Notes: https://www.tutorialsduniya.com/java
- JavaScript Notes: https://www.tutorialsduniya.com/javascript
- JSP Notes: https://www.tutorialsduniya.com/jsp
- Microprocessor Notes: https://www.tutorialsduniya.com/microprocessor
- OR Notes: https://www.tutorialsduniya.com/operational-research
First order predicate logic (FOPL)

Complex sentences are possible using connectors.

Predicate
Relationships

Well formed formula (wff)
Properly connecting sentences using connectors

Semantics - (meaning)
Using truth tables, if we say that wff is defined semantically.

A given wff $\exists x \ p(x) \rightarrow q(x)$.

Domain is needed to define quantity.

Domain, interpretation is not possible as quantity is not known.

Formal System - A formal system is a set of axioms $S$ (proven facts/atomic sentences) and set of inference rules $L$ which can be represented as

$\langle S, L \rangle$ // also known as knowledge base.
This representation is also known as knowledge base for a particular system.

- **Soundness** - The inference procedure $L$ is sound if and only if any statements (s) can be derived from the formal system as a logical consequence.

- **Completeness** - The inference procedure $L$ is complete if and only if any sentence's can be logically implied using the procedures defined in a formal system.

**Properties of FOPL**

Equivalent:

\[ P \rightarrow Q \equiv \neg P \lor Q \]

Ce rules:

\[ P \leftrightarrow Q \equiv (\neg P \lor Q) \land (\neg Q \lor P) \]

(all other rules are same as that of PL)

Conversion to the clausal form.

Representation of symbols only, no inference rules, keywords are included.

A clause is defined as the disjunction of the no. of literals (predicates).

Ground Clause - A ground clause is one that
contains no variables in the given expression.

 Horn clause: It is defined as a clause that contains at least one positive literal.

- Steps to convert to clausal form

**Step-1** - Eliminate all implications and equivalence symbols.

\[ P \rightarrow Q \text{ or } P \equiv Q \]

To be removed

To remove them, we can use their implied form.

**Step-2** - Move negation symbols into individual atoms.

\[ \forall x \, p(x) \lor q(x) \]

To make it individual.

**Step-3** - Rename variables if necessary so that all the remaining quantifiers have different variable signs.
\[ \forall x \exists y \text{ creates ambiguity} \]

1. Rename it as \( \forall x \) and \( \exists y \).

**Step 4:** Replace existential quantifier / quantified variables with special functions and eliminate the corresponding quantifiers.

Skolexmization - process to remove existential quantifiers using functions.

**Step 5:** Drop all universal quantifiers and put the remaining expressions into CNF form.

Configuration Normal Form.

**Step 6:** Drop all conjunction symbols, writing each clause previously connected by a connective on a separate line.

- To eliminate existential quantifiers from the expression (Step 4).

- Few variables / special functions are used. This process is known as "skolemization".

1. If the leftmost quantifier in an expression is an existential quantifier, then replace all the occurrences of the variable it quantifies with a constant not appearing anywhere in the expression and delete the quantifier then.
2. For each existential quantifier that is preceded by one or more universal quantifiers, then replace all occurrences of the existentially quantified variable by a function symbol not appearing elsewhere in the expression. The arguments assigned to that function should match to all the variables appearing in each universal quantifier which exists before the existential quantifier. Then delete the corresponding existential quantifier.

\[ \forall(x) \exists(y) \]

Here only one existential quantifier is present, thus the function symbol which will be made will have one parameter, as only one existential quantifier is present.

Given expression:

\[ \exists x \forall y \left( \forall z \ P(f(x)), y, z \right) \rightarrow \]

\[ \left( \exists u \ Q(x, u) \land \exists v \ R(y, v) \right). \]

1. Remove \( \rightarrow \) or \( \iff \) symbol.

\[ P \rightarrow Q = \sim P \lor Q \]

\[ \exists x \forall y \left( \sim \forall z \ P(f(x)), y, z \right) \lor \exists v \ R(y, v) \]

Negation is only inside brackets as ( ) denotes dependent values and thus
Fx $\forall y$ is independent of whole expression.

2. Remove negation and make them into individual atoms.

$$Fx \forall y \ (\neg \exists z \sim p(f(x)), y, z) \lor (Fx \exists (x, y) \& Fx \exists (y, v)).$$

$\neg A z$ becomes $A z$ and negative sign shifts to $p$.

3. Checking for ambiguity & renaming variables if necessary.

If it had been $Fx \forall y \ (Fx \sim p(f(x)), y, z)$

$x$ won't be replacing here as it is not dependent on any value inside $( )$.

Same as of 2

a. Implementing Skolemization -

It's has $Fx$ but also $Fz$ is preceded by $\forall y$.

Step 1:

Both steps will be carried.

Step 2:

$$Fx \forall y (Fz)$$

func parameter $z$ is 1 value.
Removal of $F_1$

$\forall y \exists z \sim P(y(a), y, z) \land (\exists u, Q(a, u) \land \exists v, R(y, v))$

replacing $x$
by constant $a$
which must not be
available throughout exp.

Now on leftmost we have $\forall y$

Now we remove $F_2$

no. of arguments $= 1$ as $\exists$ precedes it.

$\forall y, (\sim P(f(a), y, g(y)) \land (Q(a, h(y)) \land R(y, l(y)))$

name of functions
be made using small
symbols but shouldn't
occur in exp.

universal has

$\forall y \exists z$ combination of
3 steps.
(i) remove $F_2$
(ii) remove $F_1$
(iii) remove $F_0$

move from RHS to LHS.
5. Remove ∀ quantifier

\((\sim P(f(a)), y, q(y)) \lor (q(0, h(y)) \land R(y, l(y)))\)

If by default, no quantifier is given, always consider ∀ and not \(\forall\).

\([\forall P \lor (Q \land R)]\)

We can apply distributive law.

\((P \lor Q) \land (P \lor R) \rightarrow \text{CNF form}\)

Remove \(\lor\)

\((P \lor Q) \land (P \lor R) \rightarrow P \lor Q \land P \lor R \rightarrow P \lor Q\)

Final form.

Contributor: Riya Goel
CNF, DNF and PNF

- \( c_1 \lor c_2 \lor c_3 \lor c_4 \) \( \rightarrow \) DNF \( \{ c_1, c_2, c_3, c_4 \} \)
- \( c_1 \land c_2 \land c_3 \land c_4 \) \( \rightarrow \) CNF (clauses)
  
  (can be converted using laws or inference rules only).

\[ \rightarrow \] PNF

(prenex normal form)

- It is the form in which all quantities are present at the L.H.I.S of the clause.

- Steps to convert to PNF:

1. Eliminate all occurrences of implication or double implication from given formula.
2. Move all negations to inward such that in the end negations will only appear as part of literals.
3. Rename the variables if necessary.
4. The PNF can now be formed or obtained by moving all quantifiers to front of the formula.

\( c_\xi \) convert into PNF:

\( \forall x (\exists y \in R(x, y) \land \forall y \in s(x, y)) \rightarrow \neg (\exists y \in k(x, y)) \)

Contributor: Riya Goel
1. Remove implications using \( p \rightarrow q \equiv \neg p \lor q \)

\[ \forall x (\neg \exists y R(x, y) \lor \forall y \neg S(x, y)) \lor \neg (\exists y R(x, y) \land p) \]

2. Move negations to literals.

\[ \forall x (\forall y \neg R(x, y) \lor \exists y S(x, y)) \lor \exists y \neg R(x, y) \lor \exists y S(x, y) \]

3. Rename the variables to remove ambiguity.

\[ \begin{cases} \forall y = \exists y \\ \forall y = \exists y \end{cases} \]

\[ \forall x \forall y, \neg R(x, y) \lor \exists y S(x, y) \lor \exists y \neg R(x, y) \lor \exists y S(x, y) \]

4. Moving all quantifiers to front of the formula.

\[ \forall x \forall y, \exists y_2 \exists y_3 (\neg R(x, y_1) \lor S(x, y_2) \lor \neg R(x, y_3) \lor S(x, y_3) \lor \neg p) \]

**NOTE:**

Representing expression using quantifiers, variables.

\[ \begin{cases} \text{constants} \rightarrow a, b, c \\ \text{variables} \rightarrow x, y, z \end{cases} \]

Formal:

- **Unification:** (one/single)

  \[ \text{mother} (\text{eita, xyz}) \quad \text{Clauses} \]

  \[ \text{Def (any substitution that makes 2 or more in expressions equal is called unifier for expression)} \]

Contributor: Riya Goel
converting 2 clauses to a single form using either substitution or replacement is unification.

\[ \text{unify} \quad \frac{\text{term}}{\text{variable}} \rightarrow \text{(any predicate)} \]

**NOTE:**
A given clauses need not to be unified every time.

- **Unification** - process through which we are unifying 2 clauses through substitution or replacement.

- **Resolution** - (process)
  
  used to build check validity of a clause obtained by some other clauses.

  \[ \text{Prove by contradiction} \]
  
  \{ add negation of some clause and try to prove that \}
  
  2) proved \( \Rightarrow \) contradiction (hence not valid).

Contributor: Riya Goel
In exam (Given) 5 General Statements

1. Convert into WFF
2. Then clausal form
3. Then resolution.

Resolution principle is a syntactic inference procedure in which a set of clauses or axioms are added to the information which we need to prove.

OR

After applying the resolution, if we are getting a unsatisfiable set or empty set, then this process is known as "reputation".

* There are 4 types of resolution:

Example: \( \sim P \lor Q, \sim Q \lor R \Rightarrow \sim P \lor \sim R \) given clauses.

New clause obtained is resolvent

If a literal present in normal & one in negation, then only resolvent can be formed.

Resolvent \( \rightarrow \sim P \lor R \)

Remove this \( \sim P \lor Q \lor R \)

Literal to form resolvent

(literals can use as well as Lee)
Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from https://www.tutorialsduniya.com

- C and C++
- Computer System Architecture
- Programming in Java
- Discrete Structures
- Data Structures
- Operating Systems
- Computer Networks
- Algorithms
- Android Programming
- DataBase Management Systems
- PHP Programming
- Software Engineering
- JavaScript
- Theory of Computation
- Java Server Pages
- Operational Research
- Python
- System Programming
- Microprocessor
- Data Mining
- Artificial Intelligence
- Computer Graphics
- Machine Learning
- Data Science

- Compiled Books: https://www.tutorialsduniya.com/compiled-books
- Programs: https://www.tutorialsduniya.com/programs
- Question Papers: https://www.tutorialsduniya.com/question-papers
- Python Notes: https://www.tutorialsduniya.com/python
- Java Notes: https://www.tutorialsduniya.com/java
- JavaScript Notes: https://www.tutorialsduniya.com/javascript
- JSP Notes: https://www.tutorialsduniya.com/jsp
- Microprocessor Notes: https://www.tutorialsduniya.com/microprocessor
- OR Notes: https://www.tutorialsduniya.com/operational-research
(i) Type 1

**Binary Resolution**

dealing with 2 clauses.

\[ \sim P(x, a) \lor Q(x) \]

\[ \sim Q(b) \lor R(x) \]

Check for literal present in both form.

Yes, \( Q \)

but \( Q(x) \) and \( Q(b) \)

use unification \( \rightarrow \) term/variable.

\[ \sim P(x, a) \lor R(x) \]

\( \therefore b/x \text{ not } x/b \)

substitute \( Q(b/x) \)

\[ \times \]

(ii) Type 2

**Unit Resolution**

\( \exists UR \)

more than 2 clauses? \( \exists \) multiple clauses?

\[ \sim \text{married}(x, y) \lor \sim \text{mother}(x, z) \lor \text{father}(y, z) \]

\[ \text{married}(Sue, Joe) \]

\[ \sim \text{father}(Joe, Bill) \]

3 clauses
Here, \{ married & married \} is present in both form.

\[
\text{father}
\]

\[
\text{married (x)} \quad \text{sue/x, joey/y)
\]

\[
\text{skip this step & \sim PVP = 1?}
\]

\[
\text{married (sue, joe)}
\]

\[
\text{get cancelled}
\]

Replace all x by sue/x and y by y/joe, z by bill/z.

Final clause \( \sim \text{mother (sue/x, bill/z)} \).

(3) **Linear Resolution** (Type-3)

When each resolved clause \( c_i \) is a parent to the clause \( c_{i+1} \), this process is known as linear resolution.

\[
c_i \rightarrow c_1 \rightarrow \text{substitute D_0 in } c_1 \text{ to obtain } c_2
\]

\[
c_{i+1} \rightarrow c_2
\]

\( \therefore \) New clause is obtained by substituting in previous clause.

(getting clauses in sequence)

\[
\therefore c_1 \rightarrow c_2 \rightarrow c_3 \quad \cdots
\]

(4) **Linear Input Resolution** (Type-4)

If one of the parents in linear resolution is always from the original set of clauses, then we have implemented linear input resolution.

Contributor: Riya Goel
Ch-7

Semantic Network & Frames.

- Enhancement of knowledge representation
- Providing info. to agent through which it draws out new inferences & acts on environment.

Defining relationship with help of objects:

\[
\begin{align*}
\text{father (ram, xyz)}; \\
\text{father (ram, def)}; \\
\text{father (ram, pqr)}.
\end{align*}
\]

We need to define as many times as there are no. of same definitions in relationshipt.

To overcome this problem, we have Semantic N/W

Combining multiple entities into a single meaningful unit. or

\[
\begin{align*}
\text{xyz} \quad \text{arc} \quad \text{label} \\
\text{ram} \quad \text{arc} \\
\text{def} \quad \text{pqr}
\end{align*}
\]
We are able to combine common entity b/w multiple objects.

We need to define arc label as the last object at end of arrow as arcs are directed edges.

We can define different relationships using a common object.

Semantic N/1/W is also known as "Associative N/1/W".

Association b/w 2 entities.

Example: (Book) Tweety is a bird of yellow color design it in terms of semantic N/1/W.

1. Identify Entities.
2. Identify arc labels.

Entities - has some particular chrns./behaviour.
Always take a constant/real word as an entity.

Benefit of semantic N/w is importance of "Inheritance".

(central entity) 

Human \text{(generalized class)} 

\text{twohands} 

Ram \text{(specific class)} 

Though Ram has one hand, but generalized class entity is human, it will have a hand(since it has a defined relationship with some other entity.)

Can't restrict an entity to a particular characteristic if it is linked to some other entity.

Representation is not always a down arrow (\downarrow). Semantic N/w is just mapping. (\uparrow) is also possible in semantic N/w.

For the above example, 

Blind \text{kind-of} tweety \text{yellow} \text{color}

has pants \text{wings}

\text{in book} \text{tweety} \text{color} \text{yellow}
Here, we can also represent some hidden info. like can fly, has wings.

Color: It is an entity as it has some diff. chr. i.e. in terms of values i.e. diff. colours.

If arrows converge to common entity, then relationship will never be represented.
From central entity, we always draw out the arrows.

\[ \text{eg (in book) \text{ Ex7.1) \text{ Read?}}.} \]

Conceptual Graphs:
- Type of representation scheme.

- Types of Relationships available:
  1. Generic-generics (5 parts)
  2. Super set & Sub set
  3. Subset & Superset

eg - Fighting ships - Battleships.

This can also be a superclass/supedomain of some other entity.
Agni — fighting — battle

fighting ship has other entities.

→ A k O ( A kind of ) conceptual containment.

One automatically included in other.

eg - figure circle inheritance.

→ Role Value Restrictions.

eg - an elephant trunk is a cylinder 1.3 m in length.

defining a role of entity with its restriction
role of trunk is carrying weight but restriction is on length of 1.3 m.

cylinder is a characteristic and not a restriction.

(2) Generic - individual relationships

→ Abstraction.

- Things declared as abstract are compulsory to provide implementation.

- In next class, we define the specific implementation of that generic method.

( in programming terms)
TutorialsDuniya.com

Contributor: Riya Goel

Area: Rectangle

Circle: Triangle

defining area specifically.

→ Set membership.

conceptual containment.

defining entity has its own values and also it will inherit all characteristics of its superclass.

Human

↓

Ram

Two hands

↓

One hand

generic to specific

generic to generic

Conceptual Graphs -

Semantic N/W is a directed one, thus it can't traverse back from leaf node.

This disadvantage is overcome by conceptual graphs.

- A conceptual graph is a graphical representation of mental perception which consists of basic or primitive concepts & the
- Relationships that exist b/w the concepts.
- Concepts are enclosed in boxes, and relationships b/w the objects are enclosed in ovals. The direction of the arrow corresponds to the order of the arguments in the relation they connect.
- Concept symbols referred to entities, actions, properties or elements in real world.

- A concept may be individual or generic.
  
  **eg.-** Joe is eating soup with a spoon.

**Note:** Individual concepts have a type field followed by a referent field. (Generic version)

- First concept — Joe.
  
  (Individual concept)

- Generic version [Referent version of Joe = Person.]

- 2nd — soup (individual)
- 3rd concept — spoon.

  generic — food.

  specific field

  (generic — utensil).

→ Relations

includes some fixed versions (eating & using)
If for relations, we talk b/w specific specific, we have a fixed relationship b/w them, thus knowledge representation of every person will be same.

This is not the case in semantic NLw.

**Notes:** Relations may not be same, differ acc to person.

Use/using:

But if relation gets fixed, same info is derived by 2 diff agents i.e. reversing order by reading in-order remains same.

**Person:** Joe

**Food:** Soup

**Utensil:** Spoon

*Person:* Joe

Generic to specific.

as spoon has diff types (same case with soup) But Joe is specific, a type of entities is possible.

From where it is referred (Generic term)
Person: Joe

Eat [food: soup]

Utensil: Spoon

In conceptual graphs, instead of arc labels, we have fixed relation term depicted by ( ).

Joe [agent] Eat [object] Soup

Instrument

The keywords agent, object, instrument are placed depending upon entity.

Agent — used with generic & relation.

Note: Here, arc label will be same for all agents as relation name are fixed.
boxes → concepts
circles → relationships
ovals → are labels/mappings.

- frames: same info in terms of boxes.

  car (superframe).

  / engine \ brakes
  \ clutch / characteristics.

  and

  engine characteristics.

- Script: 2 dialogues in terms of concepts and relations.

  \{ Ch 7 + Ch 4 \} → 20 marks
  (Back ex.)

Contributor: Riya Goel
Feb 19

Conceptual Graphs

Consists of concept and relation nodes.

Concept nodes represent entities, attributes, states, events.

Relation nodes show how concepts are interconnected.

Joe -> agent -> eat -> action
entity used with an entity & an event/action

Q: Draw conceptual graph for -

1. John is between a rock & a hard place.
2. John is going to Boston by bus.

(1) person: John

between

rock

place

attribute

hand

can't use like hard & hard place.

We can indirectly use (hand is char. of that place) but it is wrong interpretation in graph.
Person: John  →  agent  →  Goal  →  Instrument  →  Bus

City: Boston

going with help of a bus

The operator is testing earth resistance using a megger

Person: Operator  →  Agent  →  Test  →  Object  →  Parameter:

Earth Resistance

Concepts:

megger

Earth resistance.

Other symbols used in conceptual graphs:

(On RHS of colon)

(1) ?  →  which

Ram is going to a house.

own house  / someone’s house.
(Specific cond is known)

[House : ?] → which house
generic term/concept.

(2) * → signifies a variable or unspecified house.

eg: Some house

[House : *]

eg: on house → specific concept referent field is known
→ no unknown quantity, no need to use special symbols

[House : on]

(3) ∀ (universal quantifier) - used for "for all"

(4) @ - it is used for quantity

(5) # - used to refer a specific value.

eg: house no. 226.

[House : #226] ←

[House : 226] ×
Inference Rules/Cond. for Conceptual Graphs

There are 4 operators available for graphical inference:

(1) **Copy** - produces a duplicate copy of conceptual Graph.

(2) **Restrict** - it modifies a graph by replacing a type label of a concept with a subtype or a specialisation from generic to individual by inserting a referent of the same concept type/content type.

*Example:* Figure ➔ Rectangle. (Rectangle is type of figure)

(3) **Join** - it combines two identical graphs i.e. CG1 and CG2 by attaching all relation arcs from CG2 to CG1 and then erasing CG2.

(4) **Simplify** - it eliminates one of 2 identical relations in a conceptual graph when all connecting arcs are of the same name/connecting arcs are also the same.

works on relations, whereas in join we merge whole of the graph.
TutorialsDuniya.com

Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from https://www.tutorialsduniya.com

- C and C++
- Programming in Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- DataBase Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- Compiled Books: https://www.tutorialsduniya.com/compiled-books
- Programs: https://www.tutorialsduniya.com/programs
- Question Papers: https://www.tutorialsduniya.com/question-papers
- Python Notes: https://www.tutorialsduniya.com/python
- Java Notes: https://www.tutorialsduniya.com/java
- JavaScript Notes: https://www.tutorialsduniya.com/javascript
- JSP Notes: https://www.tutorialsduniya.com/jsp
- Microprocessor Notes: https://www.tutorialsduniya.com/microprocessor
- OR Notes: https://www.tutorialsduniya.com/operational-research
Knowledge Representation

- **Frame** (Structured):
  - Frames - A data structure that captures the implicit connections of info. from explicitly organised DS.
  - Used to represent info. in slots & values.
  - Giving a name to a particular frame.

  ![Frame Example]

  - Slot (character of frame):
    - Name, Rno, address, sem...
  - Values:
    - Student (Frame)
      - Riya, 9.14, Sri Nagar, 6th class

- Default values can also be given to attributes.

**NOTE:** We can define a sub-frame of a frame.

  - Marks
    - Hindi
    - Maths
    - Eng
Giving name to a frame

Hotel bed

Superclass: Bed
Use: Sleeping
Size: King
Part: (Mattress, Frame)

Mattress
Superclass: Cushion
Firmness: Firm

Pair of slot & values:

Slot:
Value:

Create a frame named Student.

First give a name to student.

Student
Id: 1
Name: Riya
Specialization of:
Person
Address: Rani Bagh
Grade: A
Contains: (Marks, Course)

Through frames, inheritance is possible

Capital letter (no use of -)

Used to define subframes

Course
Name: CS
Degree: Graduation
Sem: 6
Year: 2019

Contributor: Riya Goel
Frames are general record-like structures which consist of a collection of slots and slot values. The slots may be of any size and type.

Slots typically have names & values called "facets".

* Scripts & Conceptual Dependency

aim is to moving towards standardization using scripts & new names i.e. talking of physical entities.

* 5 parts to represent sentences into structures

There are 5 different types of components that are ontological building blocks.

(State of being), next step of AI.

(i) Entities (in standard terms)

PA (picture aiders)  PP (Picture producers)

Picture producers (PP) are actors or physical objects e.g.: - Joe.
- Picture aids are supporting properties/attributes of producers (PP), e.g.- spoon.

(2) **Actions** - 2 categories

- **AA** (action aids)
- **PA** (primitive actions)

- Supports PA.

(3) **Conceptual Cases** - (written on arrows)

Types available are:

- **Objective**
- **Instrumental**

(2) **Directive**

**9mb.**

**Primitive Acts of CD Theory** :- (PA)

(1) **TRANS** - Transfer of an abstract relation

(location not changed) Ship (i.e. give).

(2) **TRANS** - Transfer of physical location of

(location changes) an object (i.e. go).

(3) **PROPEL** - Application of physical force to

an object (e.g.: push)

(4) **MOVE** - Movement of a body part by its

owner (e.g.: kick)

(5) **GRASP** - Grasping of an object by an

action (e.g.: throw)
(6) INGEST - ingesting of an object by an animal (eg - eat)

(7) EXPEL - expulsion of something from body of an animal (eg - dry)

(though changes)

(8) mTRANS - transfer of mental info. (eg - tell)

(9) mBUILD - building new info. out of old (eg - decide)

(10) SPEAK - production of sounds (eg - say)

(11) ATTEND - focusing of sense organ towards a stimulus (eg - listen)

(4) Conceptual Tenses -

Talking in present, past or future.

Tenses available are:

- Conditional (c)
- Continuing (k)
- Future (f)
- Past (p)
- Present (nul)
- Transition (t)
- Am terrogaative (q)

Can't use 'i' as 'is' is used in conceptual case (amtronomical)
→ Negative (\(\neg\))
→ Timeless (\(\delta\))
→ Start Transition (\(ts\))
→ Finish Transition (\(tf\)).

Conceptual dependency:

- has fixed rules
- how is one concept dependent on another

**Rule 1:**
\[ PP \leftrightarrow \text{ACT} \]
Action & entity are directly related.

eg - John ran

under PP

John \(\xrightarrow{\text{PTTRANS}}\) physical transformation.

denotes past tense.

John \(\xleftarrow{\text{PTTRANS}}\) physical transformation.

Use of double arrow.

**Rule 2:**
\[ \text{ACT} \leftarrow PP \]

eg - John pushed the bike.
John $\leftrightarrow$ PROPEL $\leftrightarrow$ bike.

Conceptual objective

John pushed.

Thing to be pushed.

(under Rule 1)

Rule 2 says a PP is related to contribute an action.

ACT $\leftarrow$ PP

Rule 3: PP $\leftrightarrow$ PP (both are entities)

generic-specific / specific-generic.

e.g. John is a doctor.

CD Rep: John $\leftrightarrow$ doctor.

Blank i.e. present tense.

no use of

\[ \downarrow \]

diff. from

Rule 4: PP $\leftrightarrow$ PP

not generic-specific / specific-generic.

both entities are diff. & not dependent.
eq - John's dog.

Rule 5: PP ↔ PA

eg - John is fat

CD Rep: John ↔ weight (>80).

NOTE: For words like height, tall thin, for these values we also consider a reference value.

i.e. >80 fat person
<40 thin person.
>5' tall person.

V.V. Comp:

Rule 7: ACT ↔ PP (to)

eg - John took the book from mummy.

John ↔ recipient

ATRANS ↔ book

R → John

Contributor: Riya Goel
Script Structures:

- Sequence of dialogues represented by ED.
- Useful in describing certain stereotyped situations such as going to the theatre.
- It consists of a set of slots containing default values along with some information about type of values similar to frames.
- Event tends to occur in known patterns because of clausal relationships to occurrence of events.

Contributor: Riya Goel
Script Components:

(1) Entry Conditions - Initial context/goal.

(2) Results

(3) Props - Properties

(4) Track - Script name.

e.g.

- Track: Play in Theatre.
- Props: Tickets, seats, play
- Roles:
  - Person (who wants to see a play) - P
  - Ticket Distributor - TD
  - Ticket Checker - TC
- Entry Conditions:
  - P wants to see a play
  - P has money
  - (Here we have scenes: scene 1, -- -- -- )
- Results:
  - P saw a play
  - P has less money
  - P is happy (Optional if he liked the play)
  - TD has more money

 divide goals in individual tasks.

Contributor: Riya Goel
Scene 1: Going to Theatre

Scene 2: Buying Tickets

21 Feb 19

Script

Conceptual dependencies are needed to define a script.

Script Components (eq. from book)

Script Name: Food Market

Track: Supermarket

Role: Shopper

Main entities:

Entry Conditions: Shopper needs groceries

Props: Shopping cart

Conditions:

Put item in box
Track is specific version of script
Name?

NOTE:
Roles has those entities who produce a dialogue in script.

NOTE:

entry cond. → starting basic cond.

eg:
for a restaurant
entry cond = restaurant opened
person is hungry.

Next step is defining dialogues, 3 independent tasks make 1 scene?

Scene 1: Enter market.
Shopper PTRANS
Shopper into market

Scene has to defined in terms of conceptual dependencies

Scene 2: Shop for items
Shopper move shopper through our problem lanes.
Shopper ATTEND eyes to display items (used for move)
Shopper PTRANS items to shopping cart.

Scene 3: Check out
Shopper move shopper to checkout
Shopper WAIT shopper turn
Shopper ATTEND eyes to prices/charges.
Shopper ATRANS money to cashier (abstract Transfer)
(Here we have PP only)

**Shopper** PTRANS **Shopper**.

Here, we used shopper 2 times to show conceptual Dependency $pp \iff act$.

Here we needed to define an action for producer, thus we used shopper.

**Shopper** ATTEND eyes.

Both PP & PA are present so shopper twice won't be used.

**Scene 4:** Exit market.

**Shopper** PTRANS **Shopper** to exit.

**Results:** Shopper has less money. Cashier/Market has more money. Market has less items.

**Note:** Shopper PTRANS items to shopping cart.

Actual inference is customer needs that item.

**SAM** \rightarrow Script Analyzer mechanism.

**Note:** Main aim of KR is inferencing new info.
Ch-4

4.8 Non Deductive Inferences

not deducible

→ Abductive Inference

\[ p \rightarrow q \]

no complete knowledge upon which we can conclude things.

Deductive Inference (modus ponens)

Diff blw modus ponens & abductive inference

→ We add symbol 'c' where c represents a possible causal relationship (assumption)

\[ p \rightarrow q \]

Inductive Inference

\[ P(a_1), P(a_2), \ldots, P(a_k) \]

satisfy condition

\[ \forall x \ P(x) \]

For all cases P(x) is True
inductive inferences can be false for a particular case.

- Pigeon is a bird → Then we can assume that all birds can fly. But if we see that ostrich is a bird but it can’t fly.

→ Analogical inference. Given a condition, if structure matches in terms of condition, then we can assume that if \( a \) and \( b \) are true/related.

4.9 Representation using rules

If, then, else
monotonic info. - info got is fixed & can't be changed. or certainty

we have infinite ways of representing info.

system will always be inconsistent = non monotonic?

since uncertainty, its knowledge base (can't decrease).

To handle all uncertainties we have a system

 Truth maintain system

- for a new conclusion, it doesn't remove previous conclusions but instead saves a copy of that conclusion for further use.

Block Diagram & Imp.:

\{ network consists of trees/graphs as a cycle is maintained \}
Problem Solver

Inference Engine

Inference Engine has deduced things based on KB.

Tell

Ask

TMS

Knowledge Base

Consists of \(\langle s, i \rangle\)

Handles current belief set.

(What is current belief set)

(Consists of \(\langle s, i \rangle\))

On what basis inference engine has deduced things.

Thus, TMS updates its belief set & Knowledge Base changes.

Eg:

Initially \(p \rightarrow q\)

\(q\) True

Now \(\neg p\)

Inference engine deduces \(q\) is not possible & tells TMS about this.

TMS asks inference engine if it is satisfied, it changes its current belief set & knowledge base gets updated in terms whether system remains consistent.
used to make system consistent

DDB (dependency directed Back Tracking)

(retraction) conclusion is dependent on p & p → q

NOTE:

<s, l> inference rules, set of axioms

TMS is used to maintain the consistency of the knowledge being used by the problem solver & not to perform any inference functions.

The inference engine solves domain problems based on current belief set whereas TMS maintains the currently active belief set.

- Record maintainence in form of DDB & network.

The nodes in the network representing knowledge base entries such as premises, conclusions and inference rules are attached and justified with the help of justifications.

- Premises - it is a fundamental belief which is assumed to be always true. It can be directly inferred.

  eg - ostrich can't fly → ostrich is abird.
(supports conclusion)

- **Justification** - records which are used to prove another statement.
  There are 2 types of Justification records which are used to maintain records

  - **Support list**

    - **conceptual dependency** (diff from kk)
    - helps to build IMS

* Support list → it is a data structure that contains 2 lists with a name emlist & outlist and can be represented with help of structure -
  
  \[(\text{SL} < \text{in-list} > < \text{out-list} >)\]

  - name of support list:
  - set of stmts that make support
d  - set of stmts that are set of stmts that are off to conclusion
  - conclusion true

  **eg** - sybil is a non-flying bird (ostrich)
  Given statement. (KB)

  We are trying to identify multiple conclusions & then will segregate into premises or Justification.
<table>
<thead>
<tr>
<th>Node</th>
<th>Status</th>
<th>Meaning</th>
<th>Support / Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>IN</td>
<td>Sybil is a bird</td>
<td>premise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty as it info. from given sent</td>
<td>indirectly inferred</td>
</tr>
<tr>
<td>n2</td>
<td>OUT</td>
<td>can fly</td>
<td>justified belief</td>
</tr>
<tr>
<td></td>
<td></td>
<td>direct as well as indirect info. supports n2</td>
<td>contradicts</td>
</tr>
<tr>
<td>n3</td>
<td>IN</td>
<td>Sybil can't fly</td>
<td>unjustified belief</td>
</tr>
<tr>
<td></td>
<td></td>
<td>only I can contribute in system</td>
<td>opposite</td>
</tr>
<tr>
<td>n4</td>
<td>OUT</td>
<td>Sybil has wings</td>
<td>retracted premise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unjustified belief)</td>
<td></td>
</tr>
<tr>
<td>n5</td>
<td>IN</td>
<td>Sybil is an ostrich</td>
<td>premise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clear info. from given sentence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>We don't need to prove it using any supporting node</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-flying → no wings</td>
<td>no direct inference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Induced info. is not derived from cybil is a bird</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>info. is inferred premise</td>
<td></td>
</tr>
</tbody>
</table>

Contributor: Riya Goel
Now if a new node comes

- ostrich can fly

- system becomes inconsistent upon addition of a new node & thus, we need to redefine str. of support list

- i.e. changing inlist, outlist & status

* To represent a network

- There are 4 components/symbols that are used to represent belief N/1:

1. **Premise** - true proposition requiring no justification.

2. **Assumptions** - it is a current belief that could change.

3. **Datum** - datum is either currently assumed or IE derived belief.

4. **Justifications** - these are the nodes that support any datum or assumption to be believed as true or false.

Symbols used are
(Premise) × (Assumption) (Datum)

¿See from book?

* Conceptual dependency

represented using

\[(c.p. < \text{consequent}> < \text{inhypto} > < \text{outhypo} >)\]

Consequence (result)

- An aircraft will take off only if its crew members are complete.

\[\text{eq: } (N \text{ type (aircraft) = 737 } (S_L(n1))(n2))\]

\[\text{n1 IN } class(\text{crew}) = A \ (S_L(n8, n9, \ldots, (i)))\]

\[\text{n2 OUT type (aircraft) = L400}\]

\[\text{2nd aircraft doesn't contribute to system}\]

\[\text{will takeoff}\]

\[\text{if 737 doesn't take off}\]
\[ n_4 \ \text{IN} \ \text{contradiction.} \ (SL(n_1, n_3), \ (n_2)) \]

\[ n_5 \ \text{IN} \ \text{no good } n_1 \ \text{CP}(n_4(n_1, n_3), (1)) \]

Now \( n_1 \) gets status \text{OUT}.
\[ n_2 \] gets \text{IN} instead of \text{OUT} as \( n_2 \) is possible solution for \( n_1 \).

This is \text{DDB}. first go to root problem and then try to find some solution for it.

Default Reasoning & Close World assumptions

common sense reasoning.

If \( p \) is available
\[ \neg p \text{ is also } \]
Thus if we conclude \( \neg q \), then \( \neg q \) can also be concluded.
Date: 3/1/19

Ch-6

Uncertain

Prob. of uncertainty.

Prob. in real world is of its occurrence & not its failure.

Finding probability of an event given an event has already occurred.

Bayes' Theorem/ Cond. probability (dependency).

\[ P(A|B) \]

to be found given prob.

Prob. of A given that B has already occurred.

Set

State space - Set where we have all possible values of probability.

- Bayes' Theorem:

Bayes' Theorem states that for 2 events A & E with a probability \( P(E) > 0 \), then conditional probability of event A can be calculated...
given that $E$ has already occurred

$$P(H|E) = \frac{P(H \& E)}{P(E)} \quad (1')$$

$$P(E|H) = \frac{P(H \& E)}{P(H)} \quad (2)$$

For another cond., if we want to calculate prob. of $E$ where $H$ is given.

* **Freq distribution**

$$\gamma_f(H \& E) = \frac{No.(H \& E)}{n} \quad (3)$$

$$\gamma_f = \frac{No.(E)}{Total \ no.} \quad (4)$$

Relative freq.

$$\gamma_f(H \& E) = \frac{No.(H \& E)}{No.(E)} \quad (5)$$

Rewriting $(1')$,

$$P(H|E) = \frac{P(E|H) \ P(H)}{P(E)} \quad (6')$$

This eqn. expresses the probability of event $H$ is occurring when it is known that event $E$ is occurred in the same as probability of $E$ has occurred when $H$ is given.

Contributor: Riya Goel
Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from https://www.tutorialsduniya.com

- C and C++
- Programming in Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- DataBase Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- Compiled Books: https://www.tutorialsduniya.com/compiled-books
- Programs: https://www.tutorialsduniya.com/programs
- Question Papers: https://www.tutorialsduniya.com/question-papers
- Python Notes: https://www.tutorialsduniya.com/python
- Java Notes: https://www.tutorialsduniya.com/java
- JavaScript Notes: https://www.tutorialsduniya.com/javascript
- JSP Notes: https://www.tutorialsduniya.com/jsp
- Microprocessor Notes: https://www.tutorialsduniya.com/microprocessor
- OR Notes: https://www.tutorialsduniya.com/operational-research
Consider the problem of determining the probability that a patient has a certain disease, given that symptom E was observed. For randomly chosen patients, \( P(D_1) \) is given as 0.05, and \( P(E) \) is 0.15. Also, \( P(E \mid D_1) = 0.95 \), based on doctor's previous experience.

\[
P(D_1 \mid E) = \frac{P(E \mid D_1) \cdot P(D_1)}{P(E)}
\]

\[
= \frac{0.95 \times 0.05}{0.15} = 0.316
\]

\[
P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}
\]

\[
P(\neg H \mid E) = \frac{P(E \mid \neg H) \cdot P(\neg H)}{P(E)}
\]

\[
P(H \mid E) \mid P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E \mid H) \cdot P(H) + P(E \mid \neg H) \cdot P(\neg H)}
\]

\[
\frac{P(H \mid E)}{P(H)} = \frac{\text{likelihood of event}}{\text{defines odds of event}}
\]

Contributor: Riya Goel
How to represent real-world probabilities in Bayesian N\text{\#W}

\begin{align*}
\text{notion used:} & \quad x_i \rightarrow \text{an event} \\
& \quad P(x_i) \rightarrow \text{prob. of event } x_i
\end{align*}

\begin{align*}
\exists x_3 \text{ is dependent on } x_1? \\
\text{Event } x_1 \text{ is an independent event whose prob. can be found independently.}
\end{align*}

Bayesian N\text{\#W} (multiple events)

\[ P(x_1, x_2, \ldots, x_6) = P(x_6 | x_5) \]

\begin{align*}
\text{combined probability.} \\
\text{When writing on RHS, take events in such an order which has highest subscript as we need to move from highest subscript to root node.}
\end{align*}
Now for $x_5$:

$$P(x_5 | x_3, x_2)$$

$$P(x_1, ..., x_6) = P(x_6 | x_5) P(x_5 | x_3, x_2) P(x_4 | x_2, x_1) P(x_3 | x_1) P(x_2 | x_1) P(x_1)$$

Q (in exam) either make Bayesian Nw or given an expression, draw Bayesian Nw.

6.3 Possible Worlds Representation

Identifying multiple meanings from a sentence.

For every sentence, 2 worlds exist:

- (Event occurs) True: $P$
- False (Event doesn't occur): $1 - P$

- Consistent - prob. of an event to occur is known.
- Inconsistent - prob. of occurrence of an impossible event possible worlds representation is based on this.
For more than 1 sentence
\[ \downarrow \]
possible worlds
(possible values)
\[ 2^n \text{ no. of sentences} \]

For 3 sentences
\[ \text{P, Q, P} \rightarrow \text{Q} \]

1. Possible words = \(2^3 = 8\)

To find its consistency, we need to find Truth Table.

<table>
<thead>
<tr>
<th>Representation</th>
<th>(P)</th>
<th>(Q)</th>
<th>(P \rightarrow Q) ((\sim P \lor Q))</th>
</tr>
</thead>
<tbody>
<tr>
<td>on basis of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(P &amp; Q)</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

For 2 symbols we can't define 8 entries.

(P \(\rightarrow\) Q only depends on 2 simbols P \& Q \& \(\sim P \lor Q\) symbols can't be used) 

In real world, we can only have
\[ \begin{cases} 
\text{true} & \text{true} \\
\text{true} & \text{false} \\
\text{false} & \text{false} \\
\text{false} & \text{true} \\
\end{cases} \]

4 possible cases using 2 symbols.

\[ \begin{array}{c|c|c|c}
P & Q & P \rightarrow Q \\
\hline
T & T & T \\
T & F & F \\
F & T & F \\
F & F & F \\
\end{array} \]

\(\Rightarrow\) inconsistent/Impossible

Contributor: Riya Goel
Ch-6

Possible World Representation

\[ \text{consistent} \quad \text{inconsistent} \]

\[
\text{possible soln} \quad \text{(no possible soln)}
\]

<table>
<thead>
<tr>
<th></th>
<th>( P )</th>
<th>( Q )</th>
<th>( P \rightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

2. If the truth assignments are consistent, then \( k \) set of possible worlds can be modelled with the corresponding interpretations. The probability distribution is then defined over the possible worlds where \( p_i \) is the probability that \( w_i \) is the actual world and can be represented as:

\[
\sum_{i=0}^{k} p_i = 1
\]

We use a matrix to represent possible world with \( k \) columns and \( k \) rows to represent the truth values for a given sentence and it can be represented with...
\[ q = Vp \]

where \( q \) represents the sum of probabilities, and \( V \) & \( P \) is the product (vector) gained with the help of \( q' \).

* Matrix Representation*

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
p_4 \\
\end{bmatrix}
\]

\[
P_1 + P_2 + P_3 \\
P_1 + P_3 \\
P_1 + P_3 + P_4 \\
\]

Define sentence \( S_1, S_2, S_3 \), determine probabilistic truth values of \( S_1, S_2, S_3 \). Given \( p_{w1}, p_{w2}, p_{w3}, p_{w4} \).

Bayesian NW or Possible World Representation.

Contributor: Riya Goel
Ch-12
Natural Language Processing

- Characteristics of an Agent
  - Has actuators, sensors, machine learning, natural language processing (or implement)
  - Turing Test incorporates NLP, it builds reply & then glues / provides responses

- Parsing - used for mapping in NLP.
  - RTN, ATN (two techniques)

- Terminologies in NLP:

  - Levels of knowledge used in language understanding
    1. Phonological - This knowledge relates the sound to the corresponding words we recognize.
       - Phoneme is the smallest unit of the sound.
    2. Morphological - This knowledge relates the word with their construction from the basic units called 'morphemes'. (smallest unit)

- Example: friendly where friend is basic unit (morpheme) & 'ly' is suffix.

Contributor: Riya Goel
(3) **Syntactic** - It relates the knowledge to the corresponding words where they are put together to provide a structured or grammatically correct sentences.

(4) **Semantic** - It is concerned with meanings of words & phrases and how they can be combined to provide a meaningful sentence.

(5) **Pragmatic** - This is high level knowledge which relates to the usage of sentences in different context & how the context is going to affect the meaning of the corresponding sentence. Meaning of sentence changes depending upon situation?

(6) **World** - World knowledge relates to the (space, set, language) a user must have in order to understand and carry on a conversation.

*World is whole set space where every representation is present & agent acts according to it?*

*General Approaches to Natural Language Understanding*

There are 2 ways to represent/understand a language.

Contributor: Riya Goel
with use of keywords & pattern matching.
with combination of syntax or syntactic and
semantics
comparing & matching the inputs to real
world situations.

\[ \text{Identifying keywords from a given pattern} \]

mapping 2 languages through pattern matching.

eg. cam (keyword) can be derived from 9cam.

System to identify scripts & frames.

\[ \text{SAM (Script Analyse mechanism)} \]

Initial AI system → ELIZA
(identifies patterns & keywords).

\[ \text{cat → real world entity depicted as INGEST in scripts.} \]
(2) working with syntax & semantics to build meaningful sentences? use of parsing in this method?

* Grammar & Language *

A language 'L' can be considered as a set of strings of finite or infinite length where a string is constructed by concatenating basic elements called 'symbols'.

atomic

\[ L = (\text{set of strings}) \]

can be a single chr or combination of chrs.

basic atomic unit - symbol

The finite set of 'v' symbols of the language is called the alphabet or vocabulary.

A well-formed sentences are constructed using a set of rules called "grammar".

Grammar can be represented with the representation of

\[ G = (V_n, V_t, S, and P) \]

Contributor: Riya Goel
when $V_n$ represents non terminal symbols.

$v_t$ - Terminal symbol.  
(eg.- Ram, Seeta).

$S$ - Starting symbol 

$P$ - it is a finite set of productions or rewrite rules.

eg - To process $xyz$

We need to identify meanings of $x, y, z$

Rewriting in terms of Grammar G.

$xyz \rightarrow xtz$ 

$\{v_n \rightarrow v_t, v\}$ 

Providing $y$ with a fixed constant value.

eg - A simple Grammar $G$ can be constructed with vocabulary

$N = \{ S, NP, V, VP, \star, ART \}$

Vocabulary consist of non terminal symbols.

Start Symbol.
Now we provide $Q_T$ (terminal) values

$ART \rightarrow$ will consist of a, am, the.

$ART \rightarrow$ Ram $X$ (Ram is not an article).

$Q_T = \{ \text{boy, popsicle, frog, ate, kissed, flew, the, a} \}$

To build a sentence, we need to know production rules ($P$)

Prod. Rules for this example

$P_S = S \rightarrow NP VP$

we can't provide them terminal symbols directly.

To convert them to terminal symbols, we need prod. rules.

Prod. rule of NP.

NP $\rightarrow$ ART N

$\rightarrow$ combinations (the / a)

$\rightarrow$ sentence can be started using The or a.

N $\rightarrow$ boy / frog.
prod. rule of VP:

\[ VP \rightarrow VNP \quad \text{(Same as)} \]
\[ \quad \text{(V ARTN)} \]
\[ \quad \text{(NP)} \]

\[ N \rightarrow \text{boy, popsicle, frog.} \]
\[ V \rightarrow \text{ate, kissed, flew.} \]
\[ \text{ART} \rightarrow \text{the, a.} \]

* Now we build the sentence *↓

\[ S \rightarrow \text{NP VP} \]
\[ \rightarrow \text{ARTN VP} \]
\[ \rightarrow \text{ARTN V NP} \]
\[ \rightarrow \text{ARTN V ARTN} \]

- We terminate when we get terminal symbols i.e. they become leaf nodes.
- Now we can assign terminal values.

\{ the boy ate the popsicle. \}

1st sentence.

\{ Now the boy kissed the frog. \}

\{ not a representation in real world. (not a good soln) i.e not semantically correct \}

\{ multiple combinations \}

\{ the popsicle ate the boy. \}

Contributor: Riya Goel
Building a language is easy but implementing it using semantics is difficult.

Without a starting symbol, sentence won't be build.

Q. (In exam) parsing (topdown or bottom up approach).

Building sentence given prod. rules.

Q. (building N(w, RTN & ATN).

Contributor: Riya Goel
Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from https://www.tutorialsduniya.com

- C and C++
- Programming in Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- DataBase Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- Compiled Books: https://www.tutorialsduniya.com/compiled-books
- Programs: https://www.tutorialsduniya.com/programs
- Question Papers: https://www.tutorialsduniya.com/question-papers
- Python Notes: https://www.tutorialsduniya.com/python
- Java Notes: https://www.tutorialsduniya.com/java
- JavaScript Notes: https://www.tutorialsduniya.com/javascript
- JSP Notes: https://www.tutorialsduniya.com/jsp
- Microprocessor Notes: https://www.tutorialsduniya.com/microprocessor
- OR Notes: https://www.tutorialsduniya.com/operational-research
* Type Of Grammar :-

- Chomsky Grammar :-

1. Type 0 - Type 0 is known as Simple Grammar & it can be obtained by making a simple restriction that 'y' can't be empty while rewriting the sentences.

   \[ xyz \rightarrow xwyz \]

2. Type 1 - Type 1 is known as 'context sensitive Grammar' and it can be obtained by adding the restrictions that the length of string on the RHS must be at least as long as the string on LHS, i.e. the non terminal symbols will increase.

   \[ eq. - x_0y \rightarrow x \quad y \rightarrow a \]

   length increases upon rewriting.

   In production of \( xwyz \) from \( xyz \), \( y \) must be a non-terminal symbol.

3. Type 2 - Type 2 Grammar is 'context free Grammar'. It can be characterised with the help of \( \downarrow \)

   \[ \langle \text{symbol} \rangle \rightarrow \langle \text{symbol} \rangle - \langle \text{symbol} \rangle \]
where $n$ should be greater than or equal to 1. ($n \geq 1$)

Let us have a single non-terminal symbol, so that it can be rewritten.

No restriction on length as in case of Type 1?

Type 3 - It is known as 'Regular Grammar', 'Regular Grammar', or 'finite state grammar', and it can be represented with form of

$$A \rightarrow aB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

where capital letters are non-terminals & small letters are terminals.

- Structural Representation of a Sentence

Representation is in form of a list.

Same tree-like structure as in previous topic of sentence building.

$$S \rightarrow NP\ VP$$

This represents that $S$ has some elements in it. $S$ not terminated as $S$ contains $VP$ also.
Now we draw the structure in terms of a tree.

\[ S \rightarrow NP \rightarrow ART \text{ the} \ N \text{ boy} \rightarrow VP \rightarrow V \text{ ate} \ N \text{ popsicle} \]

**Structural representation of a sentence**

\[ \{PP- preposition phrase? \}

**BasicParsing Techniques**

\[ \{ How to analyse the meaning of each symbol/ word of a sentence? \}

Parsing is the process of analysing a sentence by taking it apart word by word and determining its structure from its constituent parts & subparts.
To determine the meaning of a word, a parser must have access to lexical or dictionary from where the possible word can be identified (including their semantic) to apply it.

* eg:- John ate is an input string. It (parser) looks into dictionary and rewrites it to an output string.

For John, there is no other word for it as it is a terminal symbol. For ate, it has similar words eat, ate, eaten and thus it starts to produce an output accordingly.

* Refer Table (in Book)

* Transition Networks :-


det | noun | verb

N1 ——— N2 ——— N3 ——— N4
how we are transitting from one form to another using grammar components

Transition NW:

if cond = True go to next node
else get up to the same position only.

A Transition Network consists of a no. of
nodes and labelled arcs. The nodes
represent different states in traversing a
sentence & the arcs represent rules or
Test conditions req. to make the transition
from the current state to the next state.

A Transition NW can be successfully
traversed, then only we have a recognized
permissible sentence structure.

Parsing Techniques

Top down: Bottom Up

1. Top down - parsers may be designed to
process a sentence either in Top down or
Bottom up approach where top down
parser begins by hypothesising a sentence
with a symbol 's' and successfully predicts
lower level constituents until individual
Terminal symbols can be identified i.e.
reaching up to leaf nodes.
Top-down approach

don't miss steps in b/w & only replace one symbol in one step

movement from non-terminals to terminals.

- for Bottom-Up &-

movement from Terminals to non terminals.

```
S → NP VP
    → Name VP
        → kathy V NP
            → kathy jumped NP
                → kathy jumped ART N
                    → kathy jumped the N
                        → kathy jumped the horse.
```
Get FREE Compiled Books, Notes, Programs, Books, Question Papers with Solution* etc of following subjects from [https://www.tutorialsduniya.com](https://www.tutorialsduniya.com)

- C and C++
- Computer System Architecture
- Programming in Java
- Discrete Structures
- Data Structures
- Operating Systems
- Computer Networks
- Algorithms
- Android Programming
- DataBase Management Systems
- PHP Programming
- Software Engineering
- JavaScript
- Theory of Computation
- Java Server Pages
- Operational Research
- Python
- System Programming
- Microprocessor
- Data Mining
- Artificial Intelligence
- Computer Graphics
- Machine Learning
- Data Science

 Programs: [https://www.tutorialsduniya.com/programs](https://www.tutorialsduniya.com/programs)
 Question Papers: [https://www.tutorialsduniya.com/question-papers](https://www.tutorialsduniya.com/question-papers)
 Python Notes: [https://www.tutorialsduniya.com/python](https://www.tutorialsduniya.com/python)
 Java Notes: [https://www.tutorialsduniya.com/java](https://www.tutorialsduniya.com/java)
 JavaScript Notes: [https://www.tutorialsduniya.com/javascript](https://www.tutorialsduniya.com/javascript)
 JSP Notes: [https://www.tutorialsduniya.com/jsp](https://www.tutorialsduniya.com/jsp)
 Microprocessor Notes: [https://www.tutorialsduniya.com/microprocessor](https://www.tutorialsduniya.com/microprocessor)
 OR Notes: [https://www.tutorialsduniya.com/operational-research](https://www.tutorialsduniya.com/operational-research)