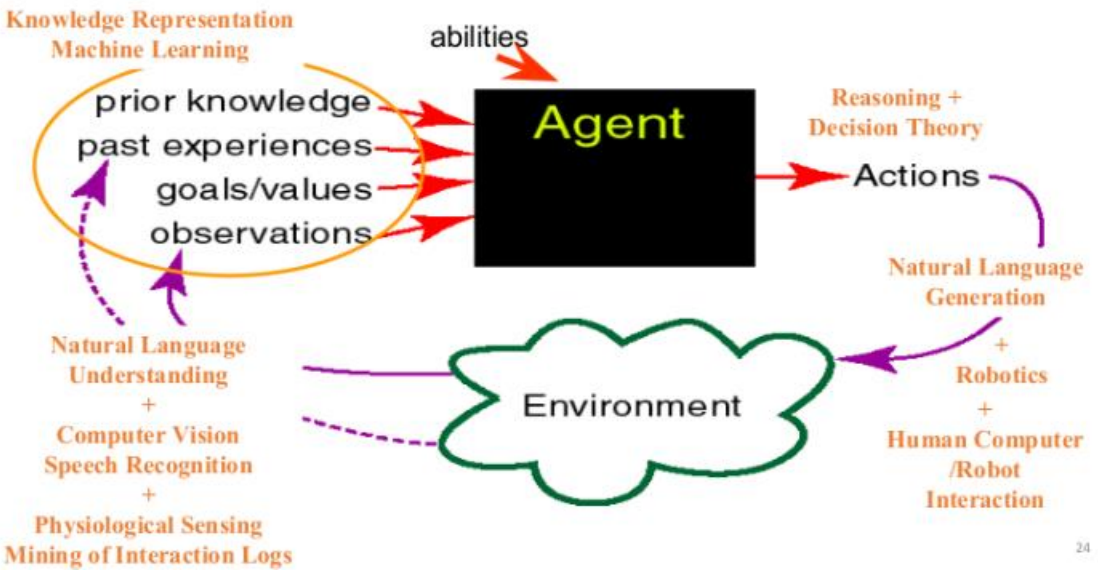
AI UNIT 1

At a glance

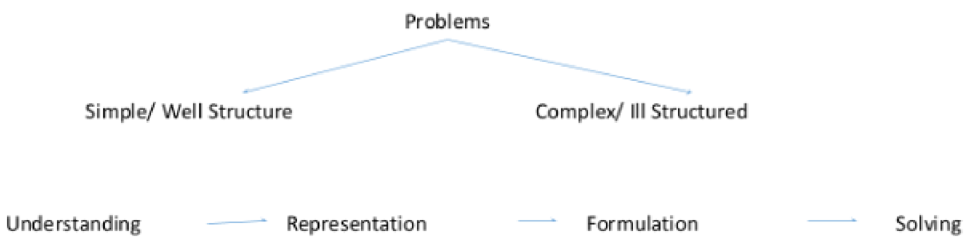
- Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans.
- It is the simulation of human intelligence processes by machines, especially computer systems.
- **Four views of AI**
    - Think like human
    - Think Rationally
    - Act like humans
    - Act Rationally
- **Major components of AI**
    - NLP (communicate in language)
    - Knowledge representation (to store what it knows)
    - Automated reasoning (use knowledge to answer questions and draw
    - conclusions)
    - Machine learning (adapt to new circumstances; prediction)
    - Vision
    - Robotics
- **Agent (An entity that interacts with its environment)**



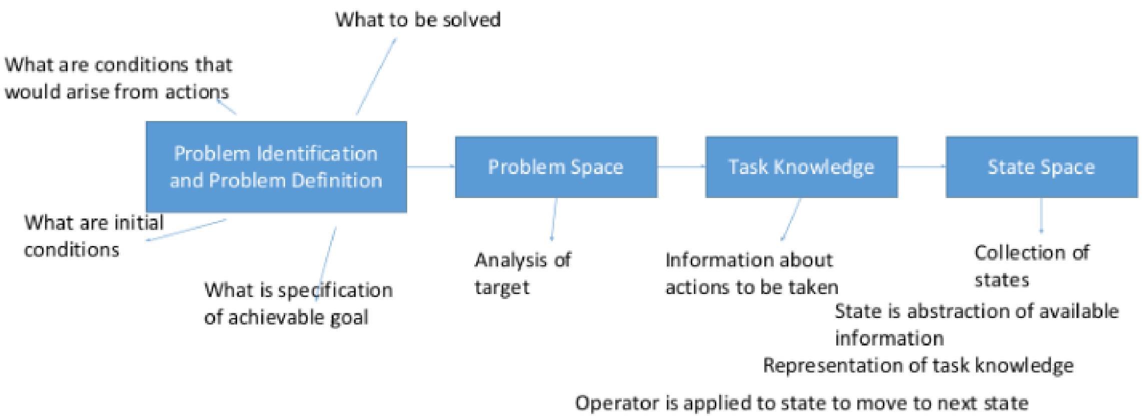
- Robots (Artificial agents having physical presence)
- Software agents (Agents without a physical presence)
- Rational agents (They sense something and do react on it)

- **PEAS (Performance measure, Environment, Actuators, Sensors)**
    - Must first specify the setting for intelligent agent design
    - Example Medical diagnosis system, Part-picking robot, Interactive English tutor
- **Environment (Surrounding)**
    - **Fully observable (accessible)**
        - Agent sensor detects all aspect of environment relevant to choose of action
        - Example Chess
    - **Partially observable (inaccessible)**
        - Environment could be partially observable due to noisy, inaccurate or missing sensors or inability to measure everything which is needed.
        - Example Driving
    - **Deterministic**
        - Deterministic environment is one in which any action has a single guaranteed effect
        - There is no uncertainty
        - Example Internet shopping
    - **non-deterministic (stochastic)**
        - Non-deterministic environments present greater problems for the agent designer
        - Example Taxi Driver
    - **Static**
        - Static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent
        - Example Crossword puzzle
    - **Dynamic**
        - Dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.
        - The physical world is a highly dynamic environment
        - Example Part picking robot
    - **Discrete**
        - An environment is discrete if there are a fixed, finite number of actions and precepts in it
        - Example Chess game
    - **Continuous**
        - Continuous environments have a certain level of mismatch with computer systems
        - Example taxi driving
    - **Single agent**
        - An agent operating by itself in an environment in single agent.
        - Example Part picking robot
    - **Multiagent**
        - Multi agent is when other agents are present (Two or more agents)
        - Example Game playing like chess

- **Problem Solving:**



- General Problem-solving techniques involve:
  - Problem Identification
  - Problem analysis and representation
  - Planning
  - Execution
  - Evaluation of solution
  - Consolidating gains
- Problem Formulation



- Problem Analysis and Representation
  - Performance of solution depends on Problem Representation
- PROBLEM DEFINITION should satisfy:
  - Compactness (must be able to restrict and define boundaries clearly)
  - Utility (must be able to restrict and define boundaries clearly)
  - Soundness (should not report false)
  - Completeness (should not lose any information)
  - Generality (should be able to capture maximum instance of problem)
  - Transparency (reasoning with representation efficiently)

- **State Space**
  - A SET, consisting of all possible CONFIGURATIONS of a SYSTEM
  - Can be a finite / infinite set
  - Goes from (start state → goal state)
  - One State can change to its NEIGHBORING State with 1 VALID MOVE
  - Valid moves change a state to its neighbouring state

State Space

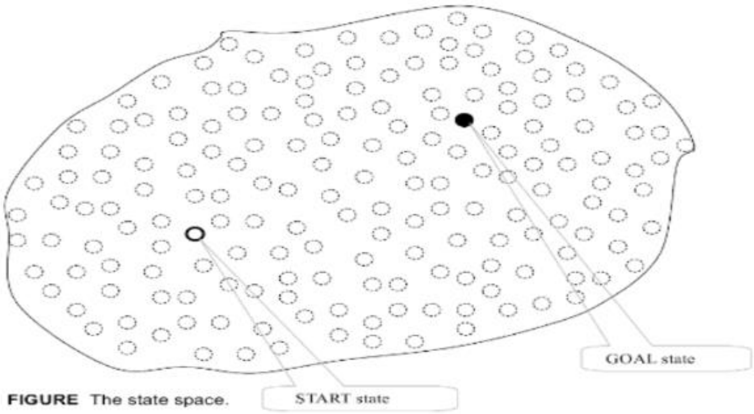


FIGURE The state space.

- We can apply this state space to games like Missionaries and Cannibals, Water jug problem

AI UNIT 2

At a glance

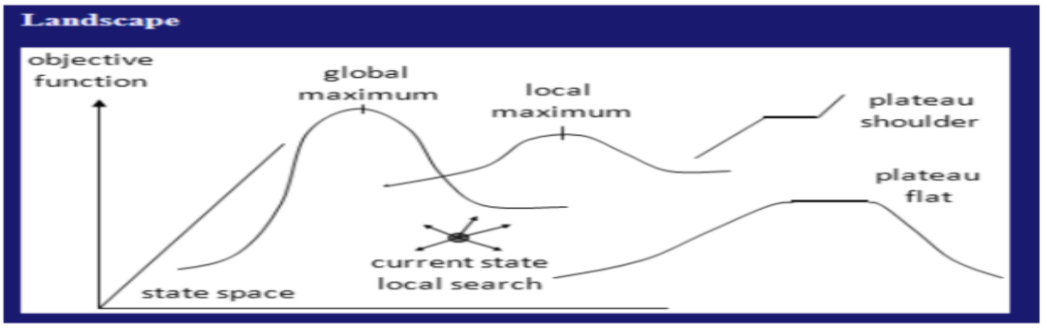
- Evaluating Search Strategies
  - Completeness
  - Time Complexity
  - Space Complexity
  - Optimality/Admissibility
- Uninformed Search (easy, very inefficient in most cases of huge search tree)
  - Breadth-First search (Expand shallowest unexpanded node)
    - Complete? → Yes, it always reaches goal (if b is finite)
    - Time complexity → $O(b^d+1)$
    - Space complexity → $O(b^d+1)$
    - It is optimal only space increases as size of problem increases
  - Depth-First search (Expand deepest unexpanded node)
    - Complete? → No: fails in infinite-depth spaces
    - Time complexity → $O(b^m)$ with m=maximum depth, terrible if m is much larger
    - Space complexity → $O(bm)$
    - It's not optimal
  - Uniform-Cost search (queue ordered by path cost Equivalent to breadth-first)
    - Complete? → Yes, if step cost ≥ ε
    - Time complexity → O of nodes with path cost ≤ cost of optimal solution.
    - Space complexity → O of nodes with path cost ≤ cost of optimal solution.
    - It is optimal for any step cost.
  - Depth-First Iterative Deepening search (avoids the infinite depth problem of DFS)
    - Searches nodes up to a depth
    - Complete? → Yes
    - Time complexity → $O(b^d)$
    - Space complexity → $O(bd)$
    - It is optimal if step cost = 1 or increasing function of depth.
  - Bi-Directional Search
    - Time and space complexity are $O(b^{d/2})$

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

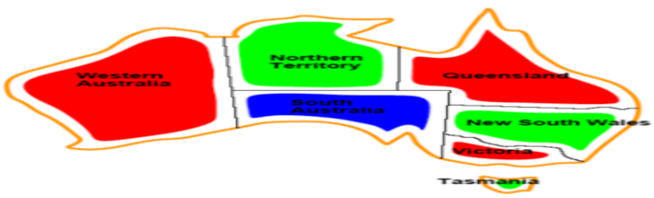
- Solution to repeated states (do not create paths containing cycles (loops), never generate a state generated before)
- Informed Search (uses problem-specific information to reduce the search tree into a small one)
  - Best First Search
    - Uses an evaluation function, f(n)
    - The path cost g is one of the examples, h(n) is required
    - Note f(n) = g(n) + h(n)
    - less space, faster
    - Types
      - **A* Search** (A Star algorithm is a best first graph search algorithm that finds a least cost path from a given initial node to one goal node)
        - Evaluation function → f(n)=g(n)+h(n)
        - It stops when f(n) == 0
        - If there is solution A* will find it, which makes this search algorithm complete
        - It is more accurate than greedy search
        - Use case: Disneyland Paris, 8 Puzzle problem
        - Time Complexity → Exponential with path length
        - Space Complexity → It is a problem as it keeps generating the nodes
      - **Greedy Search**
        - Tries to expand the node
        - Just evaluates the node n by heuristic function: f(n) = h(n)
        - It is good ideally, poor practically
        - Similar to depth-first search, not optimal, incomplete, suffers from the problem of repeated states
        - Time complexity depends on h(n) in general it is O(bm).
        - O(bm) is also space complexity
  - Hill climbing (Hill climbing search is a local search problem. The purpose of the hill climbing search is to climb a hill and reach the topmost peak/ point of that hill)
    - From the current state it moves to adjacent states going uphill and the algorithm ends when it reaches a peak (local or global maximum).
    - Types of HC
      - Simple hill climbing search
        - The task is to reach the highest peak of the mountain. Here, the movement of the climber depends on his move/steps.
        - If he finds his next step better than the previous one, he continues to move else remain in the same state.
      - Steepest-ascent hill climbing
        - Unlike simple hill climbing search, It considers all the successive nodes, compares them, and choose the node which is closest to the solution.
        - Similar to best-first search

    - Stochastic hill climbing
        - Stochastic hill climbing does not focus on all then odes. It selects one node at random and decides whether it should be expanded or search for a better one.
    - Random-restart hill climbing
        - Random-restart algorithm is based on try and try strategy.
        - It iteratively searches the node and selects the best one at each step until the goal is not found.
    - Limitations of Hill climbing algorithm
        - Local Maxima (Stuck at local maxima)
        - Plateau (If it is stuck at plateau, it is hard for algo to decide where to go)
        - Ridges (Two or more local maxima)
    - Applications of HC
        - Marketing
        - Robotics
        - Job Scheduling

    - Beam search
    - Algorithm A
- Heuristics-Local Search Algorithms (where the path cost does not matter, and it only focus on solution-state needed to reach the goal node)
    - It uses a single search path of solutions, not a search tree.
    - Local Search often needs to start from an initialized solution

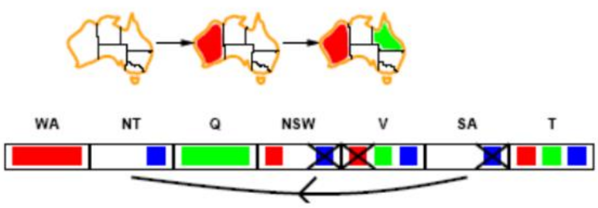


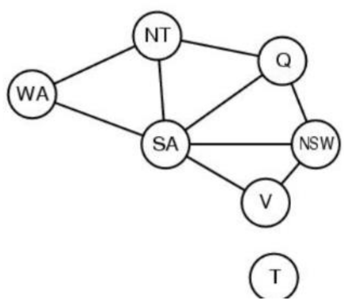
- Optimization-Hill Climbing
- Constraint Satisfaction Problem
    - An assignment is complete when every variable is mentioned.
    - A solution to a CSP is a complete assignment that satisfies all constraints.
    - Example: map colouring



- Solutions are assignments satisfying all constraints, e.g. {WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}

- Applications:
    - Scheduling the time of observations on the Hubble Space Telescope
    - Airline schedules
    - Cryptography
    - Computer Vision
- Types of CSP
    - Discrete variables (Finite domains; size d ⇒$O(d^n)$ complete assignments)
    - Infinite domains (Infinite solutions exist)
- Varieties of constraints
    - Unary constraints (involve a single variable)
    - Binary constraints (involve pairs of variables)
    - Higher-order (constraints involve 3 or more variables)
- CSP can be expressed as
    - a standard search problem
    - Commutative problem
- Backtracking search for CSP
    - Similar to Depth-first search
    - Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign
- Its efficiency can be improved by heuristics
    - Forward checking (Forward checking idea: keep track of remaining legal values for unassigned variables)
    - Minimum remaining values (choose variable with the fewest legal moves)
- Constraint propagation
    - Techniques like CP and FC are in effect eliminating parts of the search space
    - Constraint propagation goes further than FC by repeatedly enforcing constraints locally
    - It has techniques like
        - Arc consistency
            - Arc consistency detects failure earlier than FC





        - K-consistency (As all the inconsistencies are not detected by Arc consistency)
- Trade off of consistency checks
    - Takes more time
    - will reduce branching factor

    - Further improvements
        - Checking special constraints
        - Intelligent backtracking
        - Local search for CSPs
- Games vs. Search Problems
    - specifying a move for every possible opponent reply
    - unlikely to find goal
- Mini-Max Terminology
    - Utility (function the function applied to leaf nodes)
    - Backed-up value (Min position or max position)
    - Procedure (search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move)
- Minimax (Perfect play for deterministic games)
    - Logic

```
function MINIMAX-DECISION(state) returns an action
   v ← MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v
-------------------------------------------------------------
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s))
   return v
-------------------------------------------------------------
function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s))
   return v
```

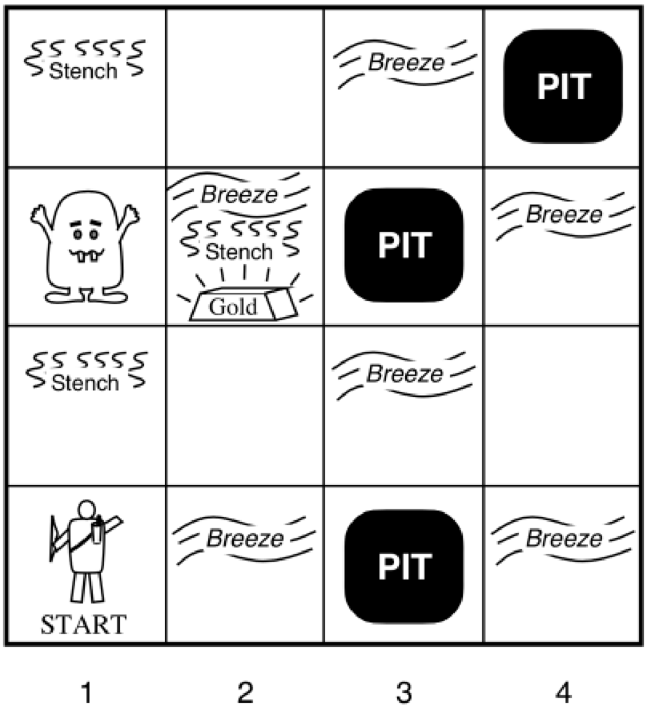
    - Properties of Minimax
        - Tree generated is finite
        - Optimal against an optimal opponent
        - Time complexity → $O(b^m)$
        - Space complexity → O(bm) (depth-first exploration)
    - Alpha-Beta Procedure
        - The alpha-beta procedure can speed up a depth-first minimax search.
            - Alpha: a lower bound on the value that a max node may ultimately be assigned ($v > \alpha$) else we don’t consider that nodes
            - Beta: an upper bound on the value that a minimizing node may ultimately be assigned ($v < \beta$) else we don’t consider that nodes

        - Pruning does not affect final result.
        - With "perfect ordering," time complexity = $O(b^{m/2})$, doubles depth of search
    - Additional Refinements
        - Waiting for Quiescence (continue the search until no drastic change occurs from one level to the next)
        - Secondary Search (after choosing a move, search a few more levels beneath it to be sure it still looks good)
        - Book Moves for some parts of the game keep a catalogue of best moves to make.
- Samuel's Checker Player
    - In learning mode (Computer acts as 2 players: A and B)
        - A adjusts its coefficients after every move
        - B uses the static utility function
        - If A wins, its function is given to B
    - A can change function by
        - Coefficient replacement
        - Term Replacement
- Kalah
    - To move, pick up all the stones in one of your holes, and put one stone in each hole, starting at the next one, including your Kalah and skipping the opponent's Kalah.
    - The winner is the player who has the most stones in his Kalah at the end of the game.
- Games
    - Single Agent Games
    - Two-Agent Games
    - Optimal Decisions in Games Min-Max Algorithm Pruning
    - Stochastic Games
- Games of Chance
    - We use minimax here as well
    - Instead of $O(b^m)$, it is $O(b^m n^m)$ where n is the number of chance outcomes.
    - Since the complexity is higher (both time and space), we cannot search as deeply.

AI Unit 3

At a glance

- **Logical Agents** (Agents that can form representations of a complex world, use a process of inference to derive new representations about the world)
- **Knowledge-Based Agents**
  - The central component of a knowledge-based agent is its knowledge base, or KB (set of sentences).
  - Each sentence is expressed in a language called a knowledge representation language.
  - When the sentence is taken as being given without being derived from other sentences, it is called as axiom
  - They can do two operations to give inference
    - Tell (Add new sentences)
    - Ask (Query what is known)
- **The Wumpus World**



  - **PEAS**
    - Performance measure (gold +1000, death (eaten or falling in a pit) -1000, -1 per action taken, -10 for using the arrow. The games ends either when the agent dies or comes out of the cave)

        - **Environment:**
            - Breeze is near pit
            - Stench is near Wumpus
            - 1000 Gold
        - **Actuators:**
            - Left turn, Right turn, Forward, Grab, Release, Shoot
        - **Sensors:**
            - Stench, Breeze, Glitter, Bump, Scream
    - **Wumpus World properties**
        - Partially observable, Static, Discrete, Single-agent, Deterministic and Sequential
- Knowledge Representation and Knowledge Base
    - Example Marcus is a man → man (Marcus)
    - **A good knowledge base has**
        - Representation adequacy (Ability to represent all kinds of knowledge that are needed in the domain)
        - Inferential adequacy (Ability to manipulate representational structures such that new knowledge can be derived/inferred from the old)
        - Inferential efficiency (Ability to incorporate additional information into an existing knowledge base that can be used to focus the attention of inference mechanisms in the most promising direction)
        - Acquisitional efficiency (Ability to easily acquire new information)
    - **Approaches to knowledge Representation**
        - Simple Rational Knowledge
            - Provide weak inferential ability
        - Inheritable knowledge
            - Objects are organized into classes and classes are organized in a generalization hierarchy.
            - Inheritance is a powerful form of inference, but not adequate.
        - Inferential knowledge
            - Facts represented in a logical form, which facilitates reasoning
            - An inference engine is required
        - Procedural knowledge
            - Representation of "how to make it" rather than "what it is".
            - May have inferential efficiency, but no inferential adequacy and acquisitional efficiency
        - Issues in KR
            - Important Attributes: Isa and instance attributes.
            - Relationships among attributes (inverses, existence in a hierarchy, single-valued attributes, techniques for reasoning about values)
            - Choosing the Granularity (High-level facts may not be adequate for inference. Low-level primitives may require a lot of storage)
            - Representing Set of Objects
            - Finding the right structure as needed
- Logic, Propositional Logic: A Very Simple Logic
    - **Logics** (are formal languages for representing knowledge to extract conclusions)
        - Syntax (defines well-formed sentences in the language)
        - Semantic (defines the truth or meaning of sentences in a world)

- **Propositional logic**
  - Preposition → declarative sentence
  - Properties of statement:
    - Satisfiability (a sentence is satisfiable if there is an interpretation for which it is true)
    - Contradiction (if there is no interpretation for which sentence is true)
    - Validity (a sentence is valid if it is true for every interpretation)
  - Inference Rules
    - Commutativity: p ∧ q = q ∧ p, p ∨ q = q ∨ p
    - Associativity: (p ∧ q) ∧ r = p ∧ (q ∧ r), (p ∨ q) ∨ r = p ∨ (q ∨ r)
    - Identity element: p ∧ T rue = p, p ∨ True = True
    - ¬(¬p) = p
    - p ∧ p = p, p ∨ p = p
    - Distributivity p ∧ (q ∨ r) = (p ∧ q) ∨ (p ∧ r), p ∨ (q ∧ r) = (p ∨ q) ∧ (p ∨ r)
    - p ∧ (¬p) = False and p ∨ (¬p) = True
    - DeMorgan's laws ¬ (p ∧ q) = (¬p) ∨ (¬q), ¬ (p ∨ q) = (¬p) ∧ (¬q)

Modus Ponens	$p \implies q$ p $\therefore q$	Modus Tollens	$p \implies q$ $\sim q$ $\therefore \sim p$
Elimination	$p \vee q$ $\sim q$ $\therefore p$	Transitivity	$p \implies q$ $q \implies r$ $\therefore p \implies r$
Generalization	$p \implies p \vee q$ $q \implies p \vee q$	Specialization	$p \wedge q \implies p$ $p \wedge q \implies q$
Conjunction	p q $\therefore p \wedge q$	Contradiction Rule	$\sim p \implies F$ $\therefore p$

  - Given an implication p → q
    - converse is: q → p
    - contrapositive is: ¬q → ¬p
    - inverse is: ¬p → ¬q

- **Convert to CNF**
  - Eliminate implications and biconditionals using formulas
  - Apply De-Morgan's Law and reduce NOT symbols so as to bring negations before the atoms
  - Use distributive and other laws & equivalent formulas to obtain Normal forms

Q. Convert into CNF : $((P \rightarrow Q) \rightarrow R)$

Solution:

Step 1: $((P \rightarrow Q) \rightarrow R) \implies ((\neg P \vee Q) \rightarrow R)$

$\implies \neg(\neg P \vee Q) \vee R$

Step 2: $\neg(\neg P \vee Q) \vee R \implies (P \wedge \neg Q) \vee R$

Step 3: $(P \wedge \neg Q) \vee R \implies \underbrace{(P \vee R) \wedge (\neg Q \vee R)}_{\text{CNF}}$

- **Resolution in propositional logic**
  - Proof by Refutation / contradiction
    - Say we have to prove proposition A Assume A to be false i.e., TA
    - Continue solving the algorithm starting from 7A
    - If you get a contradiction (F) at the end it means your initial assumption i.e., TA is false and hence proposition A must be true.
    - Clause: disjunction of literals is called clause
- **First-Order Logic** (Theorem deciding is semi decidable)
  - Can Represent → Objects and quantification
  - Ex Everyone loves john → ∀x: loves (Everyone, x) where ∀ is universal quantifier (Represents all)
  - Also ∃ represents some ex somebody kills xyz → ∃x kills (x, xyz)
  - Nested quantifier example
    - everybody loves somebody → ∀x: ∃y: loves (x, y)

- DeMorgan's rules

$$
\begin{array}{llll}
\neg\exists x \; P \equiv \forall x \; \neg P & \quad & \neg(P \lor Q) \equiv \neg P \land \neg Q \\
\neg\forall x \; P \equiv \exists x \; \neg P & \quad & \neg(P \land Q) \equiv \neg P \lor \neg Q \\
\forall x \; P \equiv \neg\exists x \; \neg P & \quad & P \land Q \equiv \neg(\neg P \land \neg Q) \\
\exists x \; P \equiv \neg\forall x \; \neg P & \quad & P \lor Q \equiv \neg(\neg P \land \neg Q).
\end{array}
$$

- Knowledge Engineering in First-Order Logic Inferences:
  - IDENTIFY THE QUESTIONS.
  - ASSEMBLE THE RELEVANT KNOWLEDGE.
  - DECIDE ON A VOCABULARY OF PREDICATES, FUNCTIONS, AND CONSTANTS.
  - ENCODE GENERAL KNOWLEDGE ABOUT THE DOMAIN.
  - ENCODE A DESCRIPTION OF THE PROBLEM INSTANCE.
  - POSE QUERIES TO THE INFERENCE PROCEDURE AND GET ANSWERS.
  - DEBUG AND EVALUATE THE KNOWLEDGE BASE.
- Inference in First-Order Logic (Reduce first-order inference to propositional Inference)
  - Universal Instantiation/ Elimination
    - In general, the rule of Universal Instantiation says that we can infer any sentence obtained by substituting a ground term (a term without variables) for a universally quantified variable.
  - Existential Instantiation/ Elimination
    - Basically, the existential sentence says there is some object satisfying a condition, and applying the existential instantiation rule just gives a name to that object.
    - The new name is called a Skolem constant.

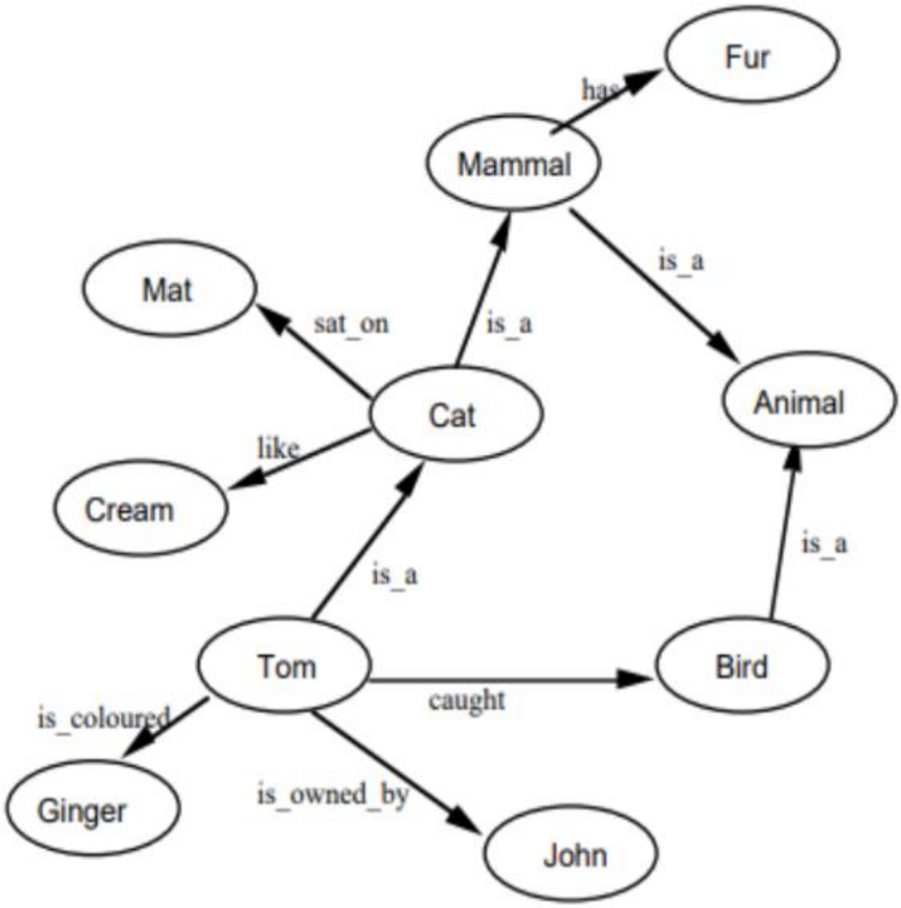
- Existential Introduction

- example, we no longer need
$\exists$ x Kill(x, Victim)
- once we have added the sentence
Kill(Murderer, Victim)

- Generalized Modus Ponens (modified form of Modus ponens)
    - we use a single inference rule called Generalized Modus Ponens for the inference process.
    - "P implies Q, and P is declared to be true, hence Q must be true," summarizes Generalized Modus Ponens.
- Inference Principles
    - Unification & Resolution
        - ✓ Matching procedure that compares two literals and discovers whether there exists a set of substitutions that can make them identical
    - Forward Chaining Rules (Bottom-up process)
        - ✓ From the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.
        - ✓ The process repeats until the query is answered
    - Backward Chaining Rules
        - ✓ Starting from the Goals, chaining through all the rules.
        - ✓ Top-down process

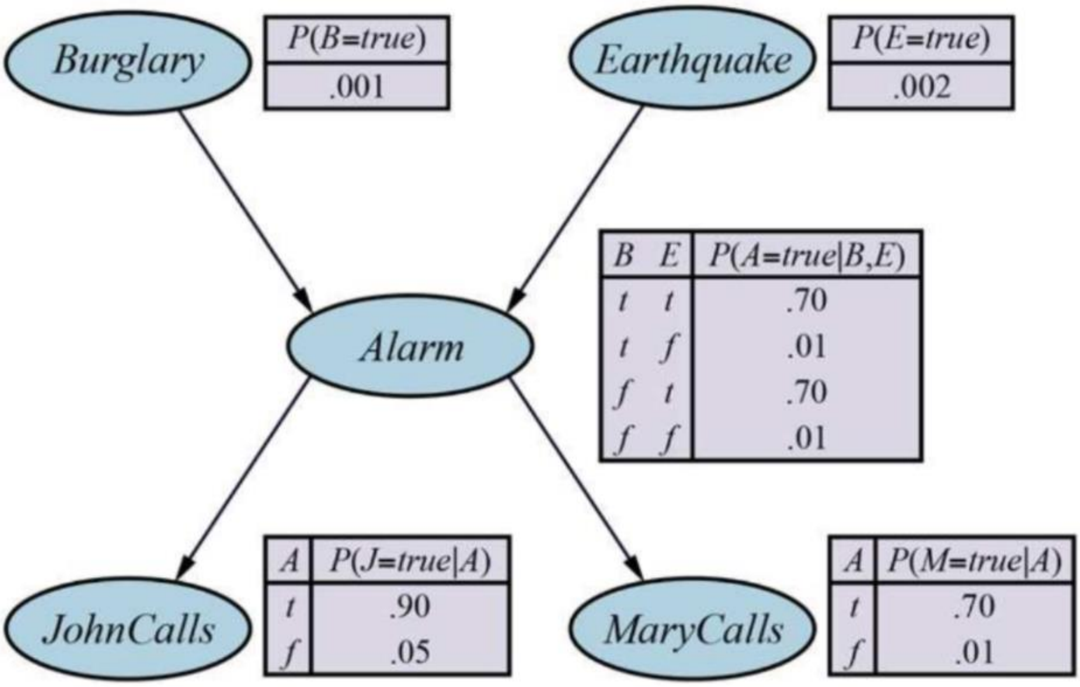
- Steps to be followed for Resolution Inference
    - Eliminate existential quantifiers
    - STANDARDIZE VARIABLES
    - SKOLEMIZE (Skolemization is the process of removing existential quantifiers by elimination)
    - DROP UNIVERSAL QUANTIFIERS
    - DISTRIBUTE ∨ OVER ∧
- Instance and Isa relationship
    - "Marcus is a man" can be written as man (Marcus) or instance (Marcus, man)
    - “All Pomerians were Romans” can be written as ∀x: Pomerians(x) → Romans(x)
    - So, instance ∀x instance (x, Pomerians) → instance (x, Romans)
    - This can Isa (Pomerians, Roman)

- **Representation-Production**
  - **Logical Representation**
    - Hierarchical task networks for reasoning about plans
    - Bayesian networks for reasoning with uncertainty
    - Markov models for reasoning over time
    - Deep neural networks for reasoning about images, sounds, and other data
  - **Semantic Networks**
    - Idea is that we can store our knowledge in the form of a graph, with nodes representing objects in the world, and arcs representing relationships between those objects.
    - Semantic net allows us to perform inheritance reasoning as all members of a class, will inherit all the properties of superclass additional relation.



  - **Production Rules**
    - Like in TOC
  - **Frames Representation**
    - Means of representing common sense knowledge. Knowledge is organized into small packets called “Frames”.
    - Frame can be defined as a structure that has slots for various objects & a collection of frames consists of expectation for a given situation.
    - Frame are used to represent two types of knowledge viz. declarative/factual and procedural, declarative & procedural Frames

- **Bayesian Belief Net**
  - A Bayesian network is a directed graph in which each node is annotated with quantitative probability information.



  - A Bayesian network is a directed graph in which each node is annotated with quantitative probability information.

- **Dempster – Shafer Theory (DST)**
  - DST is a mathematical theory of evidence based on belief functions and plausible reasoning. DST offers an alternative to traditional probabilistic theory for the mathematical representation of uncertainty.
  - Bayesian methods are sometimes inappropriate so DST used
  - Belief ≤ Plausibility
  - Mass function m(K) (It is an interpretation of m ({K or B}) i.e.; it means there is evidence for {K or B} which cannot be divided among more specific beliefs for K and B.)
  - Belief in K (It is an interpretation of m ({K or B}) i.e.; it means there is evidence for {K or B} which cannot be divided among more specific beliefs for K and B)
  - Plausibility in K (It is the sum of masses of set that intersects with K)