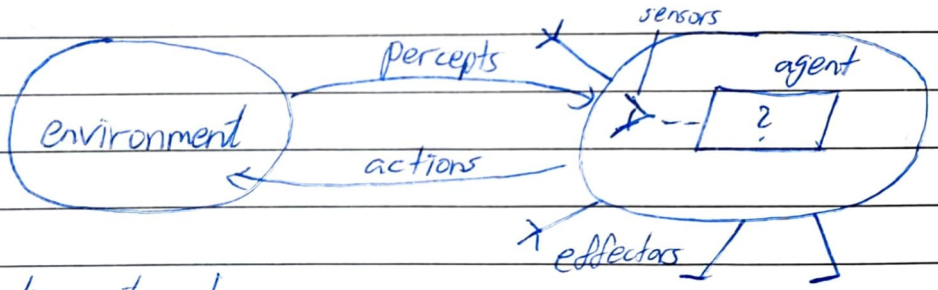
__/__/__

AI

☆ Intelligent Agents

- An intelligent agent perceives it's environment via sensors and acts rationally upon that environment with it's effectors
- Compared to a discrete agent which recieves percepts one at a time and maps this percept sequence to sequence of discrete actions.
- Properties: Autonomous, reactive to environment, goal-directed and interacts with other agents via the environment



- According to humans,

Sensors: Eyes (vision), ears (audio), skin (touch), nose (smell), neuromuscular system (sixth sense)

Percepts: At the lowest level, electrical signals from sensors. After preprocessing, objects in visual field and auditory streams.

Effectors: Limbs, digits, eyes, tongue.

Actions: Turn left/right, walk, run, lift a finger.

Ex: For an automated taxi ~~driver~~ driving system, PAGE analysis is as follows:

Percepts: Video, sonar, speedometer, odometer, engine sensors, GPS, microphone
Action: Steer left/right, accelerate, brake, horn
Goals: Maintain safety, reach destination, provide passenger comfort, maximize profits
Environment: Streets, freeways, traffic, pedestrians, weather, customers.

— Rationality (information gathering)

- A rational agent should do whatever action will maximize it's expected performance based on percept sequence and built-in and acquired knowledge.
- Performance measures include false alarm (FP) and false dismissal (FN) rates, speed, resource requirement, effect on environment, etc.

— Autonomy

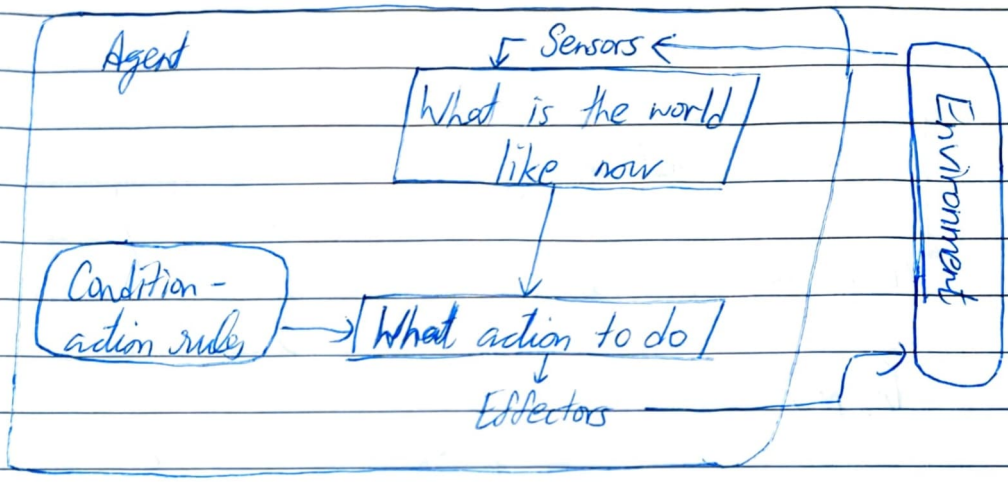
- It's own behaviour is determined by it's own experience not prior decisions it is designed with
- Agents must have enough built-in knowledge and the ability to learn, in order to survive.

— Types of agents

① Table-driven Agents

- Use a percept-sequence/action table in memory to find the next action (lookup table)
(optimal)

- Too big to generate and store, not adaptive to changes in the environment (requires updates to table) and lacks knowledge of non-perceptual parts of current state.

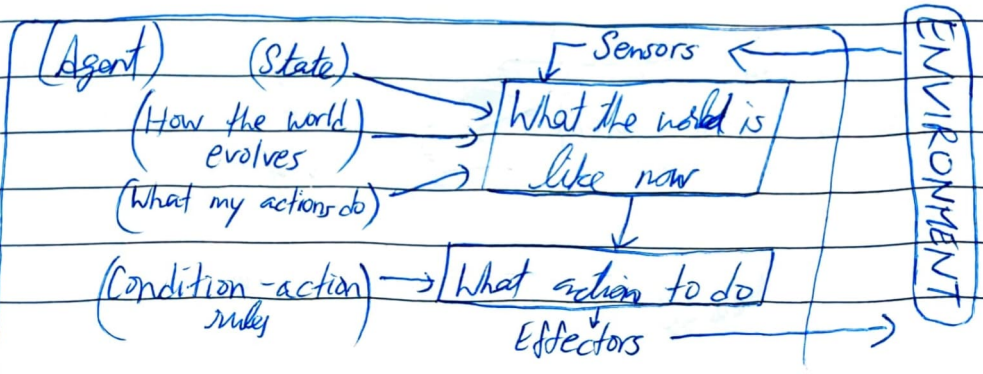


② Simple reflex agents

- Based on condition-action rules each handling a collection of percieved states.
- Even this cannot make actions conditional on previous states.

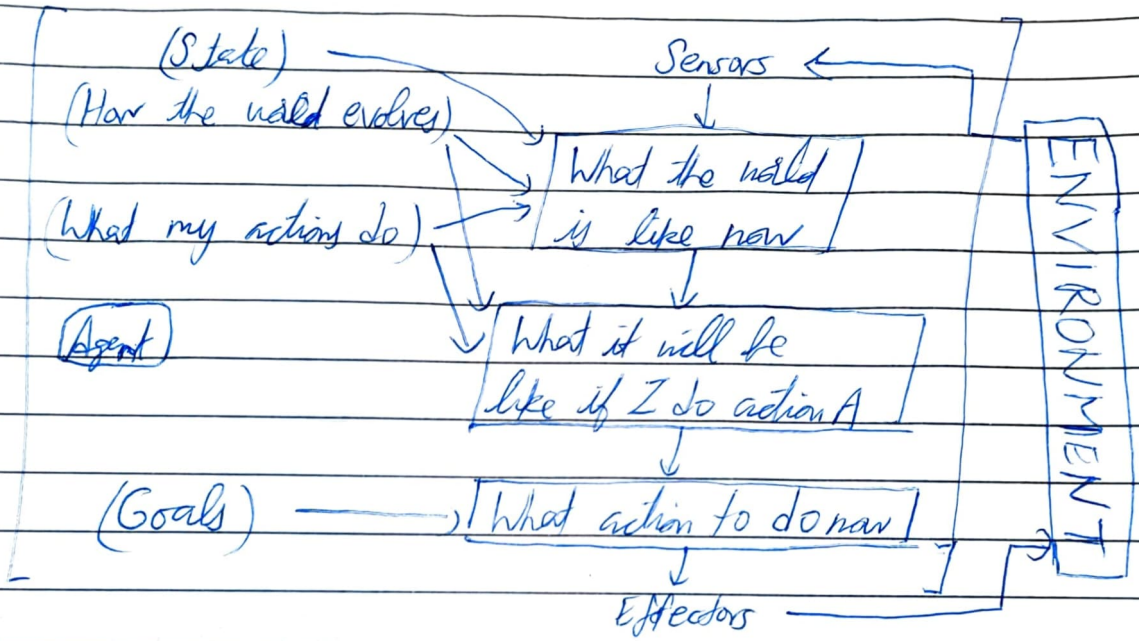
③ Agents with Memory (State-based)

- They maintain an internal state that keeps track of aspects of the world that cannot be seen in current percept. (Past percepts)



④ Agents with goals (Goal-based)

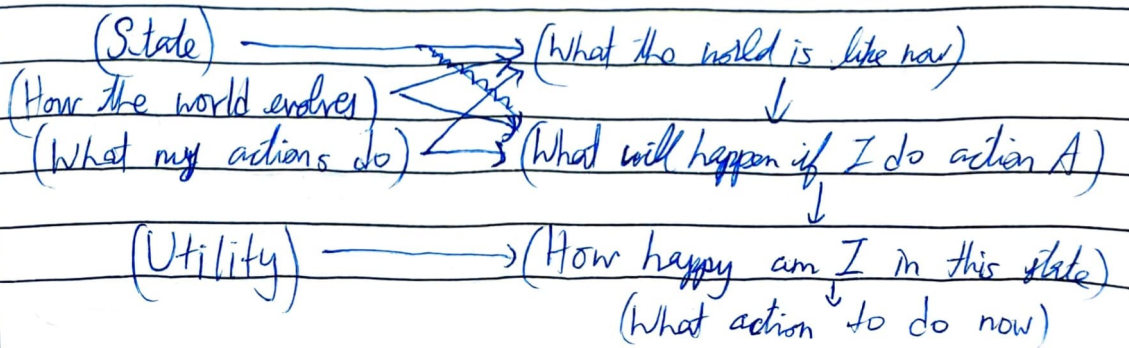
- In addition to state information, these agents have goal information that describe desirable situations (taking future events into consideration)



⑤ Utility-based agents

- Utility function U: State → Happy/Unhappy
- Optimizes choices based on preferences (degree of happiness)

Ex: If goal-based agent only cares if you reached the destination (goal), utility-based agents considers multiple factors like comfort, fuel efficiency, safety, time, etc.



Properties of Environments

- Fully observable, if agent's ~~state~~ sensors give it access to complete state of the environment to choose an action, else partially observable (need to keep track of changes in environment)

- Deterministic, if next state of environment is completely determined by current state of environment and action of agent, else stochastic (multiple unpredictable outcomes)

- Episodic, if subsequent episodes do not depend on what happened in previous episodes, else, sequential (series of connected episodes)

- Static, environment does not change with time while agent is thinking, else Dynamic

- Discrete, if no. of distinct percepts and actions are limited, else continuous

- Single agent / Mult-agent, agent needs to be concerned about strategic, game-theoretic aspects of environment (either cooperate / compete) if environment contains multiple intelligent agents.

__/__/__

PRODUCTION SYSTEMS

— Knowledge Base

- Collection of production rules (if-then statements that describe conditions and actions), facts and heuristics used to make decisions or solve problems.

Eg: For a medical diagnosis system that helps doctors identify possible diseases based on symptoms:

Rule 1: If a patient has fever and cough, then the patient may have a flu
Rule 2: If a patient has fever and sore throat, then the patient may have strep throat.
Rule 3: If a patient has fatigue and joint pain, then the patient may have rheumatoid arthritis.

— Working Memory

- Holds current state of the system, which includes facts gathered from the environment or input data.

Eg: Patient reports the following symptoms:
Fever = Yes, Cough = Yes, Sore Throat = No, Fatigue = No
These facts will be checked by the interference engine to see if which production rules apply.

— Interference Engine

- Applies the rules in the knowledge base to the facts

is the working memory, to make conclusions based on the rules and current state.

Ex Based on the patients reported symptoms, Rule 1 will be triggered, while Rule 2 and Rule 3 will not be triggered. Hence, we can conclude that the patient has d/u

- Control Strategy

- Determines how rules are applied.
- Conflict Resolution: If multiple rules are triggered, control strategy determines which rule to apply first based on rule priority or specificity.
- Recursion: Rules may trigger other rules (additional inference steps)
- Forward Chaining: System starts from current facts and applies rules to infer new facts.
- Backward Chaining: System starts with the goal and works backward to find which rules lead to that goal

Ex In the medical diagnosis system, forward chaining is used, starting with symptoms (facts) and applying rules to find a possible disease.

__/__/__

Ex (Practical Example : Medical Diagnosis System)

① Knowledge Base (Production Rules)

```
rules = [{'condition': ('fever', 'cough'), 'diagnosis': 'flu'},
         {'condition': ('fever', 'sore_throat'), 'diagnosis':
          'strep_throat'}, {'condition': ('fatigue' 'joint_pain'),
          'diagnosis': 'rheumatoid_arthritis'}]
```

② Working Memory (Facts)

```
working_memory = {'fever': True, 'cough': True,
                  'sore_throat': False, 'fatigue': False,
                  'joint_pain': False}
```

③ Inference Engine (Applying Rules)

```
def apply_rules(rules, working_memory):   (facts)
    for rule in rules:
        if all(facts.get(symptom) for symptom in
               rule['condition']):
            return "No diagnosis"
            return rule["diagnosis"]
    return "No diagnosis found"

diagnosis = apply_rules(rules, working_memory)
print(f"Diagnosis: {diagnosis}")   ⇒ Flu
```

④ Control Strategy

- Forward Chaining (simple example)
- If multiple possible diagnoses, then implement conflict

resolution strategy to prioritize certain diagnoses, or use backward chaining to check for symptoms needed to be present for disease to be confirmed.

```
def backward_chaining(goal, rules):
    for rule in rules:
        if goal in rule['diagnosis']:
            return rule['condition']
    return None
goal_diagnosis = 'flu'
symptoms = backward_chaining(goal_diagnosis, rules)
                                    => fever, cough.
```

★ Types of Production Systems

- Rule-Based Systems

• Use if-then rules to perform reasoning or decision-making

• Rules are applied based on conditions and actions and can be triggered by a control strategy to manage which rule is applied first.

- Case-Based Reasoning (CBR)

• Uses past experiences (cases) to solve new problems by finding similarities between previous cases and current situation (Retrieve, Reuse, Revise, Retain)

- Fuzzy-Logic Systems

• Can handle imprecision and uncertainty.

- Unlike traditional binary logic (T/F), fuzzy logic uses degree of truth, producing continuous output.

```
Def fuzzy_temperature_control(temp):
    if temp < 10:
        return "Cold"
    elif temp < 25:
        return "Warm"
    else:
        return "Hot"
```

(Or apply membership functions 60% warm, 40% cold)

— Expert Systems

- Designed to simulate human-like decision-making in a specific domain.

- Combines knowledge (production rules) with inference (reasoning) to mimic expert decisions (+ explaining facilities)

Ex: If stock market trend is upward and customer is risk-tolerant, suggest investing in stocks.
If stock market trend is downward and customer is risk-averse, suggest investing in bonds

— Genetic Algorithms

- Solutions evolve over time to get better (optimized)

- Works with population of solutions, not just one.
Best suited for optimization problems (robots)

- Based on Darwin theory of evolution, by choosing the best solutions (fittest) to reproduce and combining two solutions to make a new one (crossover) and making small changes to explore new possibilities.

— Production Systems with Backward Chaining

- Works backward from a goal to find the facts needed. Typically used when goal is known but facts are not.

— Planning & Control (Robotics)

- In robotics, production systems can be used for motion planning, path planning or task allocation
- These help robots make decisions based on their current state and environment
- Systems must adapt to dynamic environments

__/__/____

PROBLEM SOLVING

☆ Production Systems

- System that generates (produce) rules (states) to reach a solution. Consists of:

① Set of rules $(C_i \rightarrow A_i)$ (starting state → consequent state)
- C_i: condition
- A_i: Action

② One or more knowledge databases that contain relevant information

③ Control strategy that decides order in which rules must be applied to available database

④ Rule applies / interference engine

☆ State Space Search

- Process where we explore possible states of a problem ~~state~~ step-by-step, with the aim of reaching a goal state that satisfied the desired condition.

- Compared to traditional searching method where the search space is fully known and stored in memory, here to save memory, you generate states as you need them.

- From initial state, apply rules / actions to create next states until we reach the goal state. Once state is no longer needed, it can be discarded to save memory.

Ex: In tic-tac-toe, each move by a player forms a state space
Goal state → 3 similar consecutive symbols. (←, ↕, ↙)

Ex: (Water Jug Problem)
Jug A has max capacity (A_cap) = 4
Jug B has max capacity (B_cap) = 3
Goal capacity → 2 in either jug.

(Possible states / rules)

- Fill A : $(x, y) \rightarrow (A\_cap, y)$ if $x < A\_cap$
- Fill B : $(x, y) \rightarrow (x, B\_cap)$ if $y < B\_cap$
- Empty A : $(x, y) \rightarrow (0, y)$, $x > 0$
- Empty B : $(x, y) \rightarrow (x, 0)$, $y > 0$
- Pour A→B : moved = min $(x, B\_cap - y)$
  $(x, y) \rightarrow (x - moved, y + moved)$
- Pour B→A : moved = min $(A\_cap - x, y)$
  $(x, y) \rightarrow (x + moved, y - moved)$

→ [(0,0), (4,0), (4,3), (0,3), (3,0), (3,3), (4,2)]

Ex: Missionary - Cannibal Problem :

No. of cannibals <= No. of missionaries either side
Also, boat cannot move itself without people on board.

Tower of Hanoi: 3 pegs. Move entire stack of disks in ascending order ↓, such that no disk can be placed on top of smaller disk, from start → end peg.

__/__/____

✱ Informed Search

- Search strategy that uses heuristics (extra knowledge) about the problem to guide the search towards the goal efficiently.
- Does not only look at possible states, but also checks heuristic function $h(n)$ of each state (cost from node n to goal node) (decides what direction to search next)

— Greedy Best-First Search

- Uses heuristic function $h(n)$ (straight-line distance) to estimate cost from node to destination. Always expands the node that appears closest to the goal.

① Start from source, enqueue all the outward nodes.
② Examine each node from the queue. If goal, then stop. Else, evaluate $h(n)$ and select the node with the optimal $h(n)$ as the next 'S'.
③ Repeat ① and ②.

- Neither complete (can get stuck in loops) nor optimal (may find suboptimal solution since it ignores path cost)
- Only cares about heuristics value while finding optimal path, tries to find minimum heuristic value among nodes connected to each node. (priority queue)

Ex — heuristics = { 'A': 5, 'B': 4, 'C': 3, 'D': 2, 'E': 0 }

```
graph = { 'A': [('B', 3), ('C', 2)],
          'B': [('D', 5)],
          'C': [('D', 1)],
          'D': [('E', 3)], 'E': [] }
start_node = 'A'
goal_node = 'E'
```

Search ⇒ [(5, A, [A]) — ①
(3, B, [A,B]) — ③ (pop order)
(3, C, [A,C]) — ②
(2, D, [A,B,D]) — ⑤
(2, D, [A,C,D]) — ④
(0, E, [A,B,D,E])
(0, E, [A,C,D,E]) — ⑥ → end]

∴ A → C → D → E
visited = [A, B, C, D]

— A* Search

- Combines path cost (from graph) from start node (g(n)) and estimated / heuristic cost to goal node (h(n))
  f(n) = g(n) + h(n)

Search ⇒ [(5, 0, A, [A]) — ①
(7, 3, B, [A,B]) — ③
(5, 2, C, [A,C]) — ②
(5, 3, D, [A,C,D]) — ④
(9, 7, D, [A,B,D]) → skip ⑤
(6, 6, E, [A,C,D,E])] — ⑥ (end)

∴ A → C → D → E
Cost = 6
visited = { A: 0, C: 2, B: 3, D: 3 }

- Optimal and complete

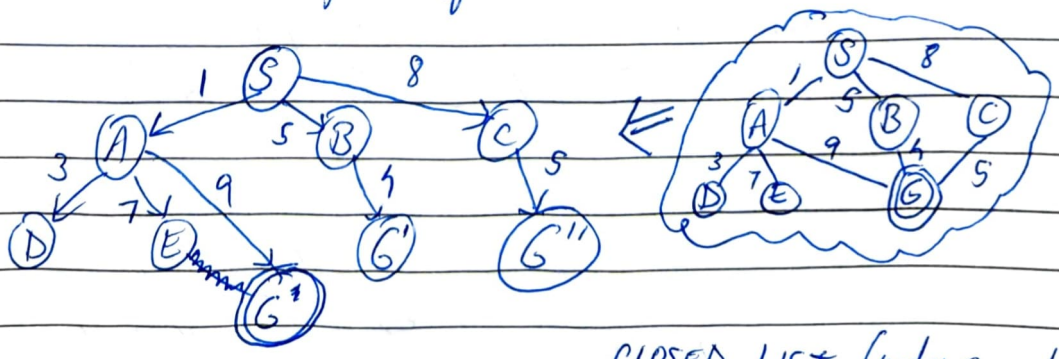
Uninformed Search (Blind)

- Use only information available in the problem definition without any knowledge about goal location (brute force) (Search until they find it)

BFS (Breadth-First Search)

- Operates as FIFO (first-in-first-out queue) also known as OPEN list.
- Explores level by level, expands all nodes at depth (d) before going to depth (d+1)
- Complete (will find a solution if it exists)
  Optimal (if all edges have same cost)
  High memory usage (stores all nodes at current level)
- Time Complexity $= O(b^d)$
  Space Complexity $= O(b^d)$
  $b \rightarrow$ branching factor (avg no. of child nodes)
  $d \rightarrow$ depth of shallowest solution

Exi-



Open List:	CLOSED LIST (nodes expanded)
{S}	{}
S → {A, B, C}	{S}
A → {B, C, D, E, G}	{S, A}

B	{C, D, E, G, G'}	{S, A, B}
C	{D, E, G, G', G''}	{S, A, B, C}
D	{E, G, G', G''}	{S, A, B, C, D}
E	{G, G', G''}	{S, A, B, C, D, E}
G	{G', G''}	{S, A, B, C, D, E, G}

For $(S \to A \to G)$, cost = 10, no. of nodes expanded = 7.

Note: CLOSED list is a search tree connected by backpointers.

— DFS (Depth First Search)

- Operates as LIFO (Last-in-first-out) stack as OPEN list.
- Always goes down one branch as deep as possible before backtracking.
- Not complete (can get stuck at ∞ depth)
  Not optimal (may find larger path)
  Low memory usage (only stores one path at a time)
- Space Complexity = $O(bd)$
  Time Complexity = $O(b^d)$ (exponential)

Ex: For previous example,

	OPEN	CLOSED
	{S}	{}
S	{A, B, C}	{S}
A	{D, E, G, B, C}	{S, A}
D	{E, G, B, C}	{S, A, D}
E	{G, B, C}	{S, A, D, E}
G	{B, C}	{S, A, D, E, G}

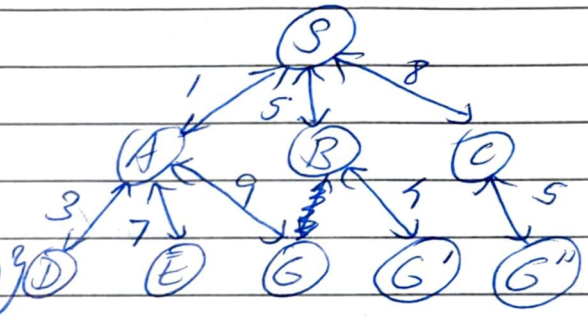
__/__/____

Here cost = 10 (S → A → G)
However no. of nodes expanded = 5

— Uniform Cost Search (UCS) (aka Dijkstra Algorithm)

- Expand node with least cost as first (g(n)).
- Here open list is priority queue (sorted by path cost)
- Complete (if costs are finite and positive)
  Optimal (always finds least-cost solution)
  Exponential time/space in worst case.

Ex:- OPEN LIST



{S}
S {A(1), B(5), C(8)}
A {D(4), B(5), C(8), E(8), G(10)}
D {B(5), C(8), E(8), G(10)}
B {C(8), E(8), G'(9), G(10)}
C {E(8), G'(9), G(10), G''(13)}
E {G'(9), G(10), G''(13)}
G' {G(10), G''(13)}

∴ S → A → G ⇒ S → B → G' (Cost = 9)
No. of nodes expanded = 7.

— Depth-First Iterative Deepening (DFID)

- Run DFS with depth limit = 0, 1, 2 until solution is found.

- Complete (tries all depths)
  Optimal (if step costs are equal like BFS)
  low memory use (like DFS)

- Time Complexity = $O(b^d)$, even though slightly more than BFS since it repeats exploration.
  (Space = bd)

— Bidirectional Search

- Search forward from start and backward from goal, stop when the two meet.

- Very efficient if branching factor (b) is high and solution is not too deep

- Works well if both start and goal ~~state~~ state are clearly defined and actions are reversible.

- Time Complexity = $b^{\frac{d}{2}}$
  Space Complexity = $b^{\frac{d}{2}}$

__/__/__

☆ Game Theory

- Branch of economics that views any multiagent environment as a game, provided impact of each agent on the others is significant, regardless of whether agents are cooperative or competitive.

- Formal definition of a game:
  - Initial state (starting position of game)
  - Successor function (returns list of (move, state) pairs)
  - Terminal test (determines when game over)
  - Terminal states (states where game ends)
  - Utility function (give numeric value for terminal states)
    (number that represents desirability of outcome for player)
    (Ex: +1 for win, 0 for draw, -1 for lose)

- ~~White~~
- Search Problem: Goal is to find path / solution
  (single agent) No opponent trying to interfere
  Only complication may be due to large search space / time.

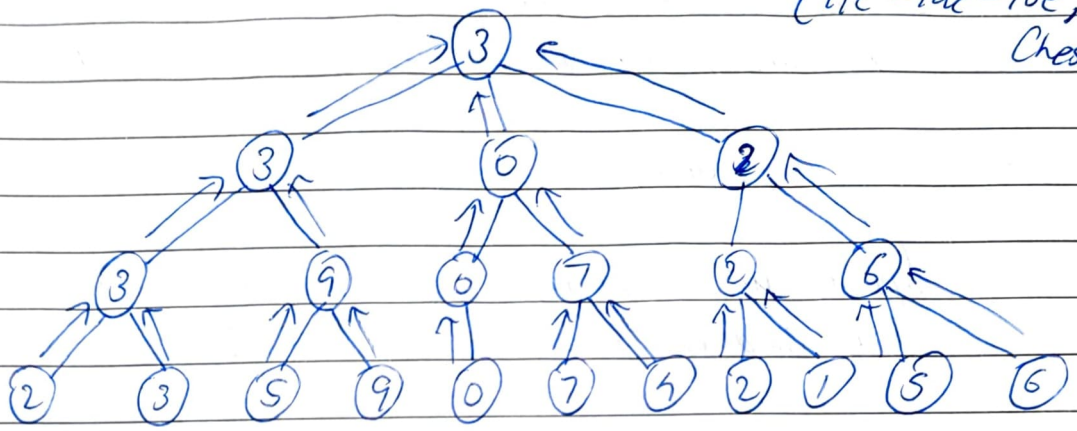
- Game Problem (multi-agent): Opponent is actively working against the agent, considering all possible moves before deciding an action. Constraints include ~~expand~~ unpredictable opponent moves and time limits.

- Typical case assumptions include two-player, turn taking (players move alternatively), deterministic (no randomness), zero sum (one player's gain is exactly another player's loss)

and both players have full knowledge of current state.

— Minimax Algorithm

- Perfect play for deterministic, 2-player game. (Tic-Tac-Toe, Chess, etc)



- Complete (if tree is finite)
  Optimal (against optimal opponent)
  Time Complexity = $O(b^m)$ — b → branching factor
  Space Complexity = $O(bm)$ — m → max depth of tree

- Limitations: Time constraints, not always feasible to traverse entire tree.

- This algorithm operates on the principle of two opposing players:

→ Max ⇒ player trying to achieve highest possible score.
  Min ⇒ Opponent, who tries to achieve lowest possible score for Max (thereby maximizing their own outcome)

- Goal is to identify the best move for Max, assuming Min will also play optimally to counter it.

__/__/____

Note — In the previous tree structure, after both players play optimally MAX can guarantee a payoff of 3. given that MIN plays optimally to minimize.

- Since Minimax explores the entire game tree to decide on the final payoff, which may be time consuming, unnecessary nodes are pruned.

— α-β Pruning

- Speeds up Minimax by skipping useless branches. thus reduces no. of nodes evaluated.

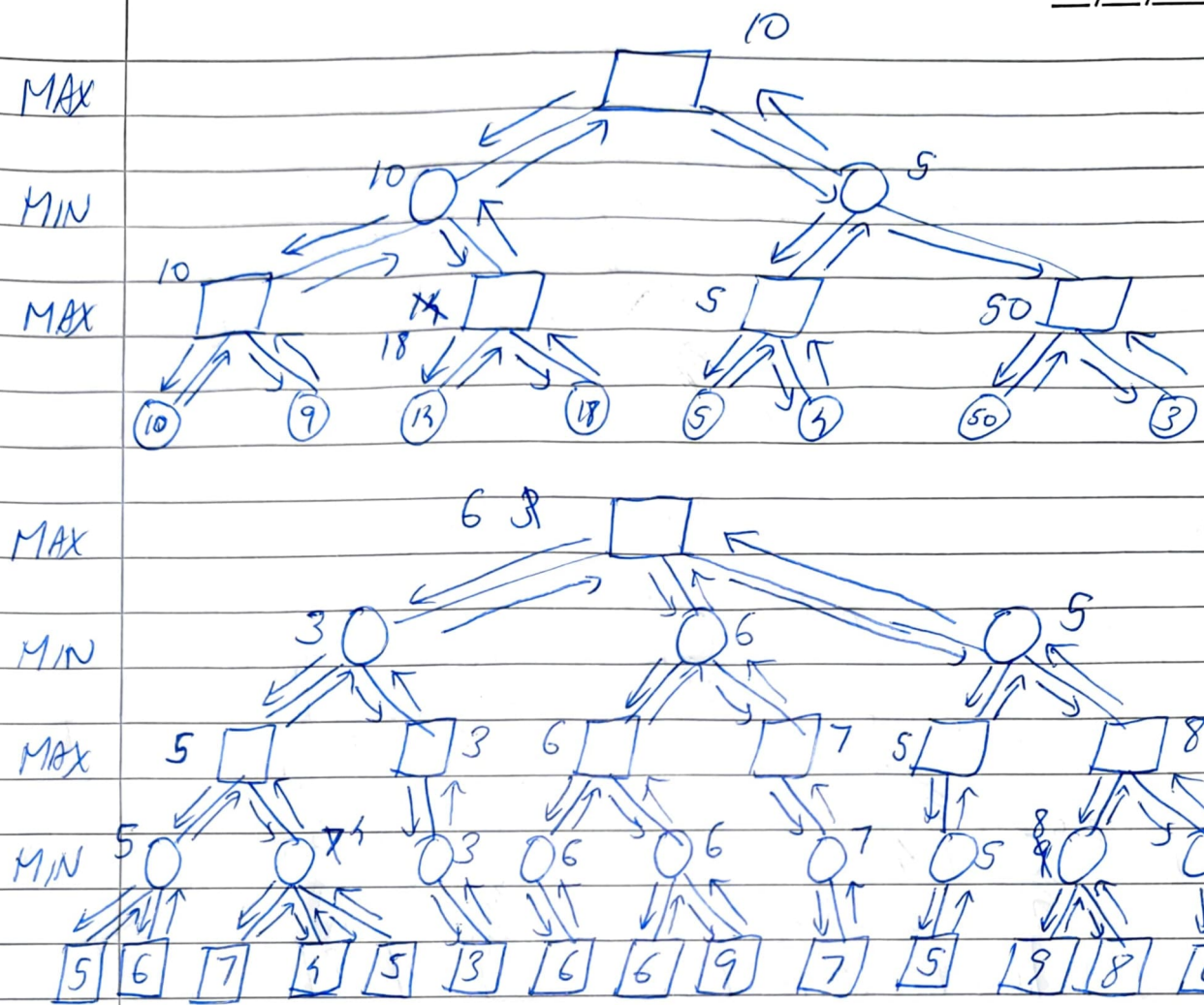
- α: Best value MAX can guarantee so far (lower bound)

  β: Best value MIN can guarantee so far (upper bound)

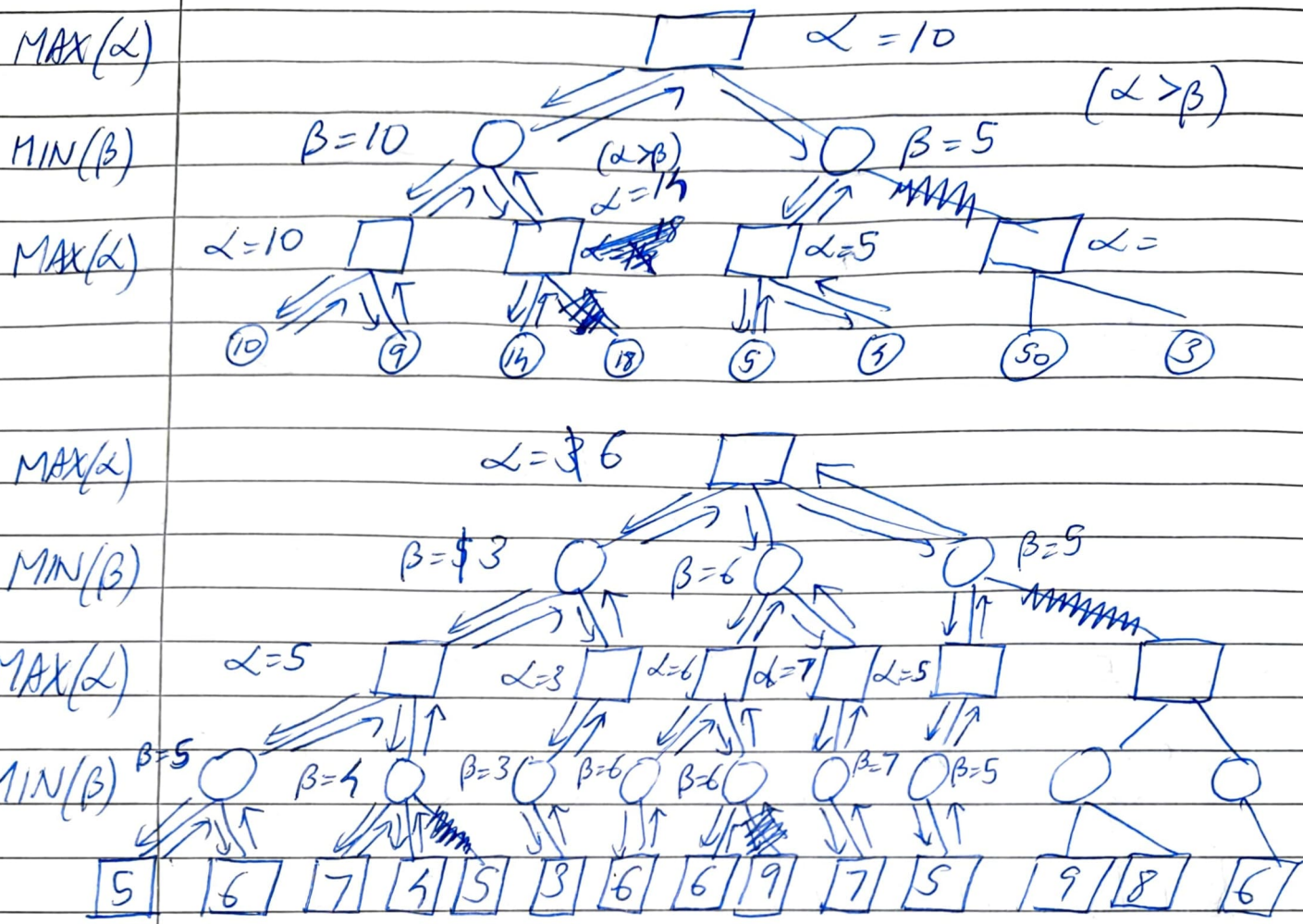
Note: Steps involved in Minimax Search Algorithm:

1. Generate the entire game tree to leaves
2. Apply utility (payoff) function to leaves
3. Use DFS to traverse from root → leaf
4. Back values from leaves towards the root
   (Max nodes computes maximum value from it's child values)
   (Min nodes computes minimum value from it's child values)
5. When value reach the root, optimal move is determined.

__/__/__



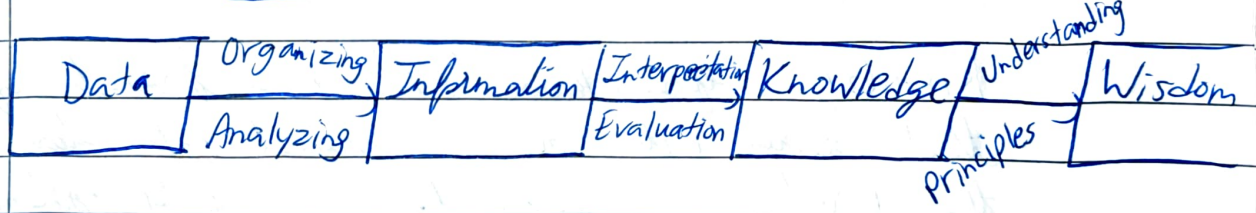
- Both Minimax and α-β cutoff/pruning give same path.
- Here MAX contains α and MIN contains β during calculation.
- If $\alpha \geq \beta$, where α belongs to left subtree of β nodes, then right subtree of α is pruned.
- Here we are making a decision whether to check next node for min/max value by comparing with parent nodes α/β value.



- As a result, time complexity = $O(b^{\frac{m}{2}})$

Knowledge Representation

- Knowledge is a progression that starts with data which is of limited utility
- By organizing or analyzing the data, we understand what the data means and this becomes information
- The interpretation or evaluation of information yields knowledge
- The understanding of principles embodied within the knowledge is wisdom



- Data is viewed as collection of disconnected facts

Ex: It is raining.

- Information emerges when relationships among facts are established and understood (who, what, where, when)

Ex: The temperature dropped 15° then it started raining

- Knowledge emerges when relationships among patterns are identified and understood. (how)

Ex: If humidity is very high, and temp drops substantially then atmosphere is unlikely to hold the moisture so it rains.

__/__/__

- Wisdom is pinnacle of understanding, uncovers principle of relationships that describe patterns. (why)

Ex Encompasses understanding all interactions that happen between raining, evaporation, air currents, temperature gradients changes and raining.

- Humans are best at understanding, reasoning and interpreting knowledge. They know things, which is knowledge, and as per the knowledge they perform various actions in the real world.

— Knowledge Representation (KR)

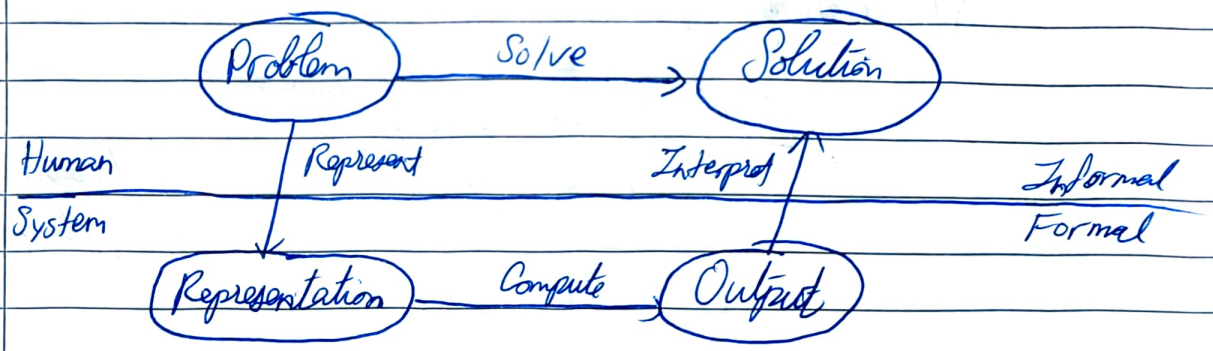
- Knowledge Representation (KR) is a branch of AI that deals with how to represent information about the world in a form that a computer system can use to think, reason and make decisions.

- KR + Reasoning focuses on how AI gets think and how that thinking leads to intelligent behaviours.

- KR stores and organizes real-world information in a way that a computer can understand, reason over and solve complex problem (medical diagnosis, NLU)

- Beyond saving data, KR also allows machine to learn from knowledge and experiences so that they can act intelligently similar to humans.

- Objects : Facts about entities in the world
- Events : Actions or occurrences happening in the world.
- Performance / Procedural Knowledge : Knowledge of how to do things (skills, procedures, methods)
- Meta Knowledge : Knowledge about what is known
- Facts : True statements about the real world that need to be represented.
- Knowledge Base (KB) : Central storage of knowledge for AI agents (logical sentences representing information)
- Knowledge : Awareness or understanding gained from facts, data and experiences.



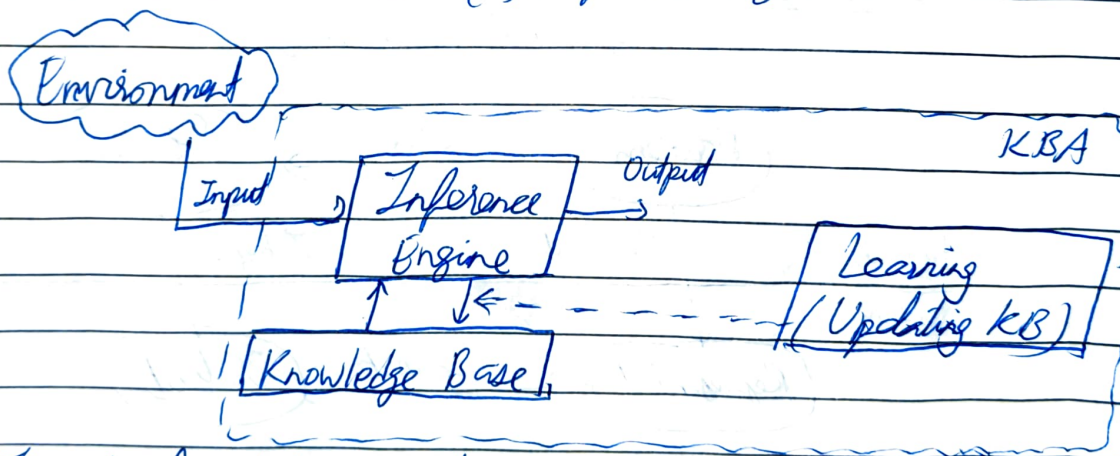
- Humans describe a problem in natural language ; after the system processes it, we get a solution in human understandable form.
- Problem is converted into a formal representation which includes logic statements, rules, semantic network, etc. This stage transforms human idea → machine-understandable knowledge.
- System performs a computation / reasoning on the formal representation, resulting in output generated by inference, rules or algorithms.

__/__/____

- Finally, the machine output is converted to human-readable solution. (informal level)

— Knowledge-Based Agents

- Agents that have the capability of maintaining an internal state of knowledge, reasoning over that knowledge, update their knowledge after observations and take actions.
- They represent the world with some formal representation and act intelligently.
- They are composed of: (i) Knowledge Base
  (ii) Inference system



- The KBA takes input from the environment by perception → takes it to the inference engine which also communicates with the KB to decide as per the knowledge in KB, learning element regularly updates the KB by learning new knowledge.
- Knowledge Base is a collection of logical sentences expressed in a KR language, storing facts about the world
- Inference means deriving new sentences from old, they

__/__/__

systems generates new facts about the world so the agent can update the KB (deduces new information by applying logical rules to the KB)

- An inference system mainly works on two rules:
(a) Forward Chaining (b) Backward Chaining.

☆ Logic Representation

- Way of representing knowledge using logic so that a machine can understand facts, reason about them and draw conclusions. (based on various conditions)
must ~~be~~ ~~have~~
- Here ~~has to be~~, no ambiguity in representation, unlike natural language.

- Logical representation require (i) Syntax (how to write the statements) and (ii) Semantics (what the statements mean)
It is of 4 types:

(a) Propositional Logic (PL) → Deals with simple true/false statement, no variables, no relations
(b) First-Order Predicate Logic (FOL) → More expressive
Uses variables, functions, relations
quantifiers
(c) Higher-Order Logic (HOL) → Even more expressive
(provides predicate of predicates)
(d) Fuzzy Logic → Deals with partial ~~logic~~ truth 0–1
not strict true/false
Used for uncertain / vague knowledge.

Propositional Logic (PL)

- Simplest form of logic where all statements are declarative (True / False), aka Boolean Logic (0/1)
- Propositions can either be True / False but not both. They consist of an object, relation or function and logical connectives. (logical operators) (does not care about meaning)

① ~~Rule for conjunction~~ : NEGATIVE operator ($\neg$)($\sim$)

A sentence such as $\neg P$ is called negation of P.

Ex: P = Today is Sunday
$\neg P$ = Today is not Sunday

P	$\neg P$
1	0
0	1

② AND operator ($\wedge$) (conjunction rule) (multiplication)

Ex: P = Rohan is smart
Q = Rohan is hardworking
$P \wedge Q$ = Rohan is smart and hardworking

P	Q	$P \wedge Q$
0	0	0
0	1	0
1	0	0
1	1	1

③ OR operator ($\vee$) (disjunction) (addition)

Ex: $P \vee Q$ = Rohan is smart or hardworking

P	Q	$P \vee Q$
0	0	0
0	1	1
1	0	1
1	1	1

④ CONDITIONAL operator ($\rightarrow$) (implies) (implication)
(if this then that)

	P	Q	$P \rightarrow Q$
Ex: If it is raining, then the street is wet ($P \rightarrow Q$)	0	0	1
	0	1	1
	1	0	0
	1	1	1

- Whenever P happens, Q must happen. (else rule broken)
  If P doesn't happen, rule doesn't care (True)

⑤ BICONDITIONAL operator ($\leftrightarrow$) (if and only if)

	P	Q	$P \leftrightarrow Q$
Ex: If I am breathing, then I am alive ($P \leftrightarrow Q$)	0	0	1
	0	1	0
	1	0	0
If I am alive, then I am breathing ($Q \leftrightarrow P$)	1	1	1

- Whenever P happens, Q must happen
  Whenever Q ~~see~~ happens, P must happen.
  (if same-same, True, if different, False)

- $P \wedge Q = Q \wedge P$ (Commutative)
  $P \vee Q = Q \vee P$
  $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$ (Associative)
  $(P \vee Q) \vee R = P \vee (Q \vee R)$
  $P \wedge T = P$ (Identity) $\quad \neg(\neg P) = P$ (double negation)
  $P \vee T = T$
  $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$ (Distributive)
  $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$
  $\neg(P \wedge Q) = (\neg P) \vee (\neg Q)$ (De Morgan's Law)
  $\neg(P \vee Q) = (\neg P) \wedge (\neg Q)$

- In AZ, propositional logic is the relation between truth value of one statement to the truth value of another.
- Even though $A \vee B \equiv B \vee A$ don't make sense in natural language, it does make sense in propositional logic.

Ex: $\neg(P \wedge Q) = (P \rightarrow \neg Q)$

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg Q$	$P \rightarrow \neg Q$
0	0	0	1	1	1
0	1	0	1	0	1
1	0	0	1	1	1
1	1	1	0	0	0

Ex: $(\neg P \vee \neg Q \vee R) \equiv (Q \vee R) \equiv (P \rightarrow R)$

P	Q	R	$\neg P$	$\neg Q$	$\neg P \vee \neg Q \vee R$	$P \rightarrow R$	$Q \vee R$
0	0	0	1	1	0	1	0
0	0	1	1	1	1	1	1
0	1	0	1	0	1	1	1
0	1	1	1	0	1	1	1
1	0	0	0	1	1	0	0
1	0	1	0	1	1	1	1
1	1	0	0	0	1	0	1
1	1	1	0	0	1	1	1

PTO →

__/__/__

Exo- $(P \vee Q) \vee \neg(P \vee (Q \wedge R)) = 1$

P	Q	R	Q∧R	P∨Q	P∨(Q∧R)	(P∨Q)∨¬( )
0	0	0	0	0	0	1
0	0	1	0	0	0	1
0	1	0	0	1	0	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	0	1	1	1
1	1	1	1	1	1	1

Exo- $(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$

P	Q	R	P→Q	P→R	Q→R	☺	P→(Q→R)	☺
0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1

Exo- $((P \Leftrightarrow (Q \to R)) \Leftrightarrow (P \Leftrightarrow Q)) \to R$

P	Q	R	Q→R	P⇔(Q→R)	P⇔Q	( )→R	((P⇔(Q→R))⇔
0	0	0	1	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	1	0	1	1
0	1	1	1	0	0	1	0
1	0	0	1	1	0	1	1
1	0	1	1	1	0	1	1
1	1	0	0	0	1	0	1
1	1	1	1	1	1	1	1

- However propositional logic has limited expressive power, we cannot describe sentences in terms of their properties or logical relationships.

- We cannot represent relations such as ALL, SOME or NONE with propositional logic.

— Inference Rules

- New sentence can be created by logically following the set of sentences of knowledge base.

Inference Rules	Premises (KB)	Conclusion
Modus Ponens	$X, X \to Y$	Y
Substitution / Chain Rule	$X \to Y, Y \to Z$	$X \Rightarrow Z$
AND Introduction	X, Y	$X \wedge Y$
Transposition (Contrapositive)	$X \to Y$	$\sim Y \to \sim X$
Resolution	$X \vee Y, \neg Y \vee Z$	$X \vee Z$

(implied by complementary literals)

~~Interpret~~

— Production Rules

- Consist of (condition, action) pairs that mean "if condition then action"

- It is composed of 3 parts: (a) a set of production rules (b) working memory (c) the recognize-act-cycle.

Ex: IF (at bus stop AND bus arrives) THEN action (get into the bus)

__/__/__

- Advantages: (a) Expressed in natural language.
  (b) Highly modular, can easily add, remove, modify an individual rule

- Limitations: (a) This system does not exhibit any learning capabilities, does not store the result of the problem for future uses.
  (b) During execution of program, many rules may be active hence rule-based production systems are inefficient.

- A few limitations / issues in Knowledge Representation are:

(i) Attributes (properties / features used to describe an object)

Different systems, datasets or contexts use different names for similar attributes such as isa, instance, type, class, category, etc. (what type of thing it is)

Eg: Dog isa Mammal
Mammal isa Animal
Snoopy instance Dog

(if AI does not understand that isa and instance mean the same, this breaks the inheritance chain)

(ii) Relationships (how two entities are connected)

Relationships may be ambiguous, unclear, directionally confusing, etc. leading to misinterpretation.

Ex: banned (John, NYC)
AI system might not know if John is banned from NYC or NYC is banned from John or "banned" is a typo for "band" (music group) (these must be explicitly defined)

__/__/____

(iii) Granularity (how detailed the knowledge representation should be)

If knowledge is too fine-grained, it becomes too complex, too many rules, hard to infer.
If too coarse, AI becomes dumb and misses details

Ex — Tom feeds a dog → feeds (tom, dog)
Tom gives a bone to the dog → gives (tom, bone, dog)

Since "feed" and "give" seem to be similar, should we use the same predicate or different ones like a coarse-grained one: transfers (tom, dog, food)
transfers (tom, dog, bone)
(same kind of action
V.S. more specific feed/give)

— First-Order Predicate Logic (FOL)

- Extension of propositional logic that is able to sufficiently express natural language statements representation in a concise way (can express relationship between objects)
(can develop info about objects easily)

- FOL not only assumes that the world contains facts like propositional logic (T/F) but also assumes constant terms (nouns), variables (pronouns), relationships (isBrother()), (functions)

- Atomic sentences are the basic form of FOL sentences formed by predicate symbol followed by sequence of terms in parenthesis.

Ex — Ravi and Ajay are brothers ⇒ Brothers (Ravi, Ajay)
(verb) (descriptor) (property)

__/__/__

- Complex sentences are made by combining atomic sentences using connectives

- FOL statements comprise of (a) Subject (main part of statement) and (b) Predicate (relation, binds atoms together)

Ex: x is an integer
Subject → Predicate

- A quantifier is a language element which specifies the quantity of specimen in the universe of discourse.

- Symbols that permit to determine or identify the range or scope of a variable in a logical expression. They are of two types:

(a) Universal Quantifier (∀, for all): (for every, for each)

Statement within it's range is true for everything or every instance of a particular thing

Ex: ∀x man(x) → drink(x, coffee) (All men drink coffee)

(b) Existential Quantifier (∃x, there exists, for some, for atleast one x)

- Statement within it's scope is true for atleast one instance of something

Ex: ∃x boys(x) ∧ intelligent(x) (Some boys are intelligent)

__/__/____

Ex: All birds fly
$\forall x \; bird(x) \rightarrow fly(x)$

- Every man respects his parent
$\forall x \; man(x) \rightarrow respects(x, parent)$

- Some boys play cricket
$\exists x \; boys(x) \wedge play(x, cricket)$

- Not all students like both Mathematics and Science
$\neg \forall x [student(x) \rightarrow like(x, Mathematics) \wedge like(x, Science)]$

- Only one student failed in Math
$\exists x [(Student(x) \wedge Failed(x, Math)) \wedge$
$\forall y (Student(y) \wedge y \neq x) \rightarrow \neg Failed(y, Math)]$

(There exists a student X who failed Math) +
(Every other student Y ($Y \neq X$) did not fail Math)

- Every student loves some student
$\forall x (Student(x) \rightarrow \exists y (Student(y) \wedge Loves(x, y)))$

- Every student loves some other student
$\forall x (Student(x) \rightarrow \exists y (Student(y) \wedge \neg(x = y) \wedge Loves(x, y)))$

- There is a student who is loved by every other student
$\exists x (Student(x) \wedge \forall y (Student(y) \wedge \neg(x = y) \rightarrow Loves(y, x)))$

- Bill takes either Analysis or Geometry (not both)
$Takes(Bill, Analysis) \iff \neg Takes(Bill, Geometry)$

- Bill takes Geometry ~~and~~ or Analysis (or both)
  Takes(Bill, Analysis) ∨ Takes(Bill, Geometry)

- No student loves Bill
  ¬ ∃x (Student(x) ∧ Loves(x, Bill))

- Bill has no sisters, atleast one sister.
  ¬ ∃x SisterOf(x, Bill) ∃x SisterOf(x, Bill)

- Bill has atmost one sister
  ∀x, y (SisterOf(x, Bill) ∧ SisterOf(y, Bill) → x = y)

- Bill has ~~atleast~~ exactly one sister.
  ∃x (SisterOf(x, Bill) ∧ ∀y (SisterOf(y, Bill) → x = y))

- Bill has atleast 2 sisters,
  ∃x, y (SisterOf(x, Bill) ∧ SisterOf(y, Bill) ∧ (x ≠ y))

PL	FOL
• Weak representation language	• Strong expressive language
• Represents whole sentences using single propositional symbols (P, Q, R) but cannot represent properties of individuals/relations between entities.	• Represents knowledge using ~~predict~~ predicates (properties or descriptors), constants, variables, functions, relations it is able to represent properties and relations between entities
• Meaning is context-independent, only deals with True/False statement	• Meaning depends on domain objects, quantifiers (context-dependent), deals with structured statements about entities and their relationships

__/__/____

→ Declarative Knowledge :

- Knowledge about facts, statements that are true.
- Static (does not change), easy to state, easy to read.
- Describes what is true, low cognitive adequacy. (better for machine beginners)
- Highly efficient, less modifiable.

Ex: The Sun rises in the East

→ Procedure Knowledge

- Knowledge about procedures, how to do something, how to perform an action, involves ~~procedures~~ methods / operations / processes
- Dynamic (involves steps) action-based knowledge
- Harder to express but captures action, requires good cognition matching (easier for humans)
- Less efficient but easy to modify / update steps

Ex: How to make tea.

☆ Inference Engine

- Applies logical rules to knowledge base to infer new information from known facts.

— Forward Chaining

- Form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in forward direction to extract more data (where premises are satisfied)

until goal is reached. (data-driven inference)

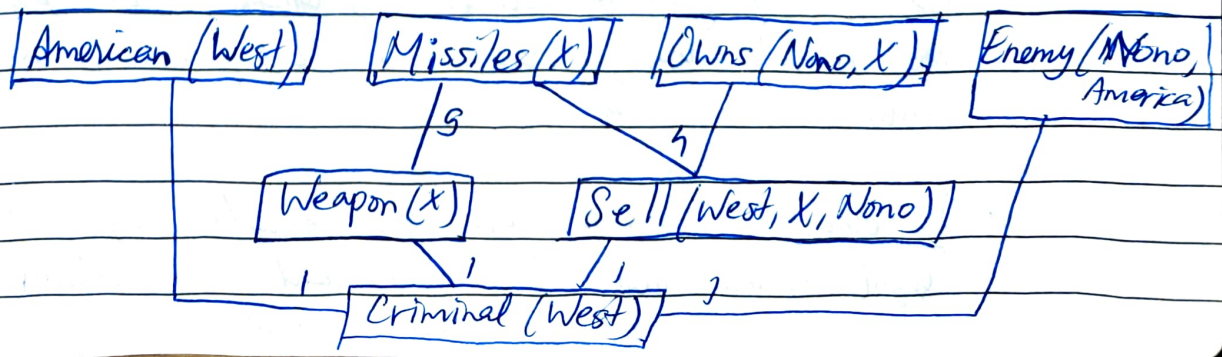
- Down-up approach, moves from bottom to top

Ex: Data: It is raining
Decision: We will take umbrella
We already know that it is raining, that is why ~~we~~ it decided to take umbrella.

Ex: We want to prove: Criminal(West)
Facts
① It is a crime for an American to sell weapons to a country that is an enemy of America
② Nono is an enemy of America
③ Nono has some missiles
④ All missiles owned by Nono were sold to Nono by West
⑤ Missiles are weapons
⑥ West is an American.

① $American(X) \wedge Weapon(Y) \wedge Sells(X, Y, Z) \wedge Enemy(Z, America) \rightarrow Criminal(X)$
② $Enemy(Nono, America)$
③ $Owns(Nono, X), Missile(X)$
④ $Missile(X) \wedge Owns(Nono, X) \rightarrow Sells(West, X, Nono)$
⑤ $Missile(X) \rightarrow Weapon(X)$
⑥ $American(West)$

Base Facts:

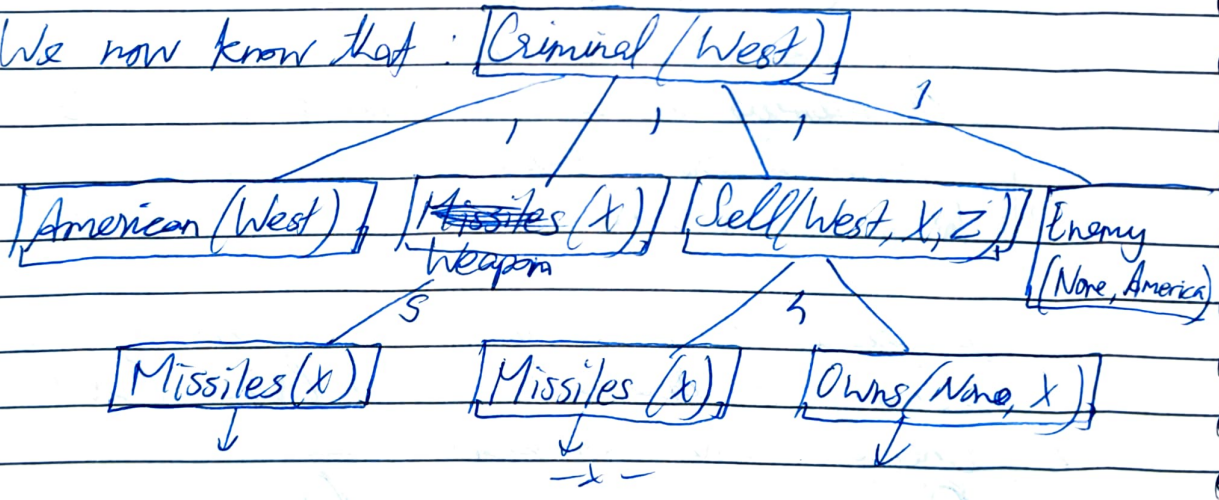


__/__/__

Backward Chaining

- Form of reasoning that starts with the goal and works backward, chaining through rules to find known facts that support the goal.
- Top-down approach, goal-driven approach (list of goals decide which rules are selected and used)
- Used in game theory, automated theorem proving tools, inference engine, etc.
- Uses DFS strategy for proof

Ex: We now know that: Criminal(West)



Formal Systems

- Machine-like setup for doing logic, consists of Axioms (S) (starting pieces, statements/facts you accept as given) and Inference Rules (L) (how to derive new sentences from old ones)
- Soundness: If you can prove something using the system (legal inference rules) it is guaranteed to be actually true.
  (You can prove the truth)

- Completeness: If something is true, the system is powerful enough to prove it. (The truth can be proven)

☆ Conversion to Clausal Form

- Since machines cannot reason directly with normal FOL formulas / statements, we have to convert it into clausal form (Conjunctive Normal Form) such that there are no quantifiers, no $\rightarrow$ or $\leftrightarrow$, all variables are universal, just a set of literals with appropriate AND or OR's.

- Resolution is an inference rule used by machines to prove things, but only works on clauses. (clean mechanical reasoning rule)

① Eliminate implications: $(P \rightarrow Q)$ becomes $(\neg P \vee Q)$
$P \leftrightarrow Q$ becomes $(\neg P \vee Q) \wedge (\neg Q \vee P)$

Ex $\forall x [(P(x) \rightarrow \exists y\, Q(x,y)) \wedge (\forall z\, R(y,z) \rightarrow S(x))]$

$\forall x [(\neg P(x) \vee \exists y\, Q(x,y)) \wedge (\neg \forall z\, R(y,z) \vee S(x))]$

② Move negations inwards (towards literals P, Q, R)

$\forall x [(\neg P(x) \vee \exists y\, Q(x,y)) \wedge (\exists z\, \neg R(y,z) \vee S(x))]$

Note De Morgan's law applied: $\neg \forall z A \rightarrow \exists z (\neg A)$
$\neg \exists z A \rightarrow \forall z (\neg A)$

③ Standardize variables apart to avoid collisions if two quantifiers use same variable name.
$(\forall x (P(x)) \wedge \exists x (Q(x)))$ → rename to y

④ Skolemization (remove all $\exists$ existential ~~vars~~ and replace that variable with a Skolem function of the universal variable(s) it depends on)

$$\forall x[(\neg P(x) \vee \exists y\, Q(x,y)) \wedge (\exists z(\neg R(y,z) \vee S(x,z)))]$$

$$\forall x[(\neg P(x) \vee Q(x, f(x))) \wedge (\neg R(f(x), g(x)) \vee S(x,z))]$$

Note: Notice here, $\exists y$ is surrounded/covered by $\forall x$ outside, while replacing, we used a Skolem function based on that dependant variable ($f(x)$, $g(x,y)$...)

⑤ Drop universal quantifiers (remove $\forall$ since they are implicitly $\forall$ universal)

$$(\neg P(x) \vee Q(x, f(x))) \wedge (\neg R(f(x), g(x)) \vee S(x,z))$$

⑥ Ensure CNF form (AND of ORs)

Since we already have conjunction of 2 disjunctions, no distribution needed

EXPERT SYSTEMS

- A computer program that contains expert knowledge about a particular problem, often in the form of if-then rules, that is able to solve problems at a level equivalent to or greater than human experts.

- Objective is to transfer expertise from human experts to a computer system and then to other humans (non-experts)

(general AI)

- While knowledge-based agents store knowledge about the world and use reasoning to choose actions based on percepts (robots), expert systems are domain-specific programs that use expert knowledge and inference rules to give advice / diagnosis (interactive) (no awareness of world)

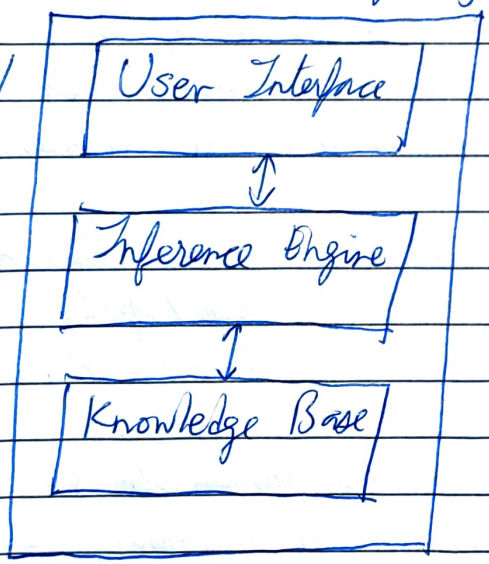
- Every expert system is knowledge-based but not every knowledge-based system is an expert system.
(Expert → reason, Agents → reason + act in the world)

- Some behaviours exhibited by human experts include recognizing and formulating the problem, solving problems quickly and properly, explaining the solution, learning from experience, restructure knowledge, break rules, determine relevance.

Category	Problems Addressed
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observations. Type of interpretation

__/__/____

Category	Problems Addressed
Planning	Develop plans to achieve goals
Design	Configuring objects under constraints
Monitoring	Comparing observations to plans
Debugging	Prescribing remedies for malfunctions (treatment)
Repair	Administer a prescribed remedy
Instruction	Diagnosing, debugging and correcting student performance
Control	Interpreting, predicting, ~~repairing~~ repairing and monitoring system behaviour



(Ask Questions/ Get Answers from user)

(Facility for the user to interact with expert system)

(reasoning (thinking), make logical deductions based on knowledge in KB)

(contains domain knowledge (if-then rules))

- Expert systems also contain:

(a) Working Memory (temporal memory where current facts are stored, useful during reasoning, gets updated as inference engine derives new facts)

(b) Knowledge Acquisition Facility (helps import knowledge from human experts, sometimes automatic, sometimes manually entered rules.) (MOLE)

__/__/__

(c) Explanation Facility (explains why system gave that answer, shows which rules fired, to increase user trust)

- Knowledge representation deals with formal modelling of expert knowledge in computer program. Must support acquiring new knowledge, retrieving knowledge, reasoning with knowledge.

- KR schemas include Production Rules (condition action pairs) frames and semantic nets.

- Inference Mechanisms examine a knowledge base to derive conclusions, answer questions or solve problems.
Types:

(a) Theorem Provers / Logic Programming (Prolog)
(b) Production Systems (rule-based)
(c) Frame Systems / Semantic Networks (structured object-like knowledge)
(d) Description Logic Systems (~~used in ontologies~~)

- In rule-based system, a rule fires if all IF-conditions are TRUE

→ Forward Chaining (Data-Driven)

- Starts from facts → applies rules → derive new facts (Working memory)

1. Match (find rules whose IF matches the working memory
2. Conflict Resolution (choose rule (specificity, recency, avoiding
3. Act (fire rule, update working memory) loops)
4. Repeat

__/__/__

- Used for monitoring, control systems and systems where data arrives continuously.

→ Backward Chaining (Goal-Driven)

- Starts with goal → looks for rules that can conclude it → create subgoals (goal stack)

1. Start with goal
2. Find rules whose THEN matches the goal
3. Push their IF conditions as new goals
4. Continue until all goals satisfied / goal cannot be proven.

- Used for diagnosis systems, expert medical systems, query answering.

- Advantages: → Useful when expertise is limited / expensive
  → Increase productivity and quality of decision-making
  → train new employees
  → enable remote knowledge transfer
  → Preserve expert knowledge.

- Limitations: → expertise is hard to extract from humans
  → expensive to design and maintain
  → knowledge engineers are rare
  → works well in narrow domains
  → cannot learn automatically (unless integrated with ML)
  → users may not trust machine decisions

Ex: PROLOG (logic programming language), EMYCIN (expert shell)
MYCIN (diagnoses infectious diseases, prescribe antimicrobial therapy)
DENDRAL (rule-based, analyzes molecular structure)
PROSPECTOR (provides advice on mineral exploration)
MOLE

Wumpus World

- Wumpus World is a cave of 4×4 = 16 rooms connected with passageways along the edges (4-neighbours). We have a knowledge-based agent (A) who will go forward in this world.

- The cave has a beast called a Wumpus who eats anyone who enters the room. It can be shot by the agent, but the agent has only one arrow.

- There are pits which are bottomless, if the agent falls in Pits, he will be stuck there forever.

- In one room, there is a possibility of finding a heap of gold.

- The goal of the agent is to find the gold and climb out of the cave w/o falling into pits / being eaten by the Wumpus.

- Rooms adjacent to the Wumpus room are smelly, it would have some stench. (4-adjacent)

- Rooms adjacent to PITS have a breeze
- There is glitter in the room with gold.
- Once the Wumpus is killed by the agent as if the agent is facing to it, Wumpus will emit a horribly loud scream that can be heard anywhere in the cave.

(PEAS)

— Performance Measures

(Game Ends)

+1000 if agents comes out of the cave with gold
-1000 if agent taken by Wumpus / falls in a pit
-1 for every action
-10 for using an arrow

— Environment

- 4 x 4 grid of rooms
- Agents initially in room [1,1] facing towards right
- Location of gold and Wumpus chosen randomly expect room [1,1], 20% probability of square being a pit except room [1,1]

— Actuators

- ←, →, ↑, Grab, Release, Shoot

— Sensors

- Agent will perceive the stench if he is in a room 4-adjacent to Wumpus.
- Agent will perceive the breeze if 4-adjacent to PIT

__/__/__

- Agent will perceive glitter if in room where gold is present
- Agent will perceive the dump if he walks into a wall.
- When Wumpus is shot, Agent can perceive loud scream

Wumpus World Properties

- Partially Observable (agent can only perceive adjacent rooms)
- Deterministic (result and outcome already known)
- Sequential (order is important)
- Static (Wumpus & Pit are not moving)
- Discrete
- One agent

Exploring the Wumpus World

① Initially in room [1,1], we already know room is safe (OK) for agent (A), no breeze/stench hence adjacent squares [2,1] and [3,1] are OK.

② Agent moves to [2,1], perceives a breeze (B), pit can be in [3,1] or [2,2] → put symbols (P?) ([2,1] = V)

③ Agent goes back to [1,1] and moves to [1,2], perceives a stench, Wumpus can either be in [1,3] or [2,2] mostly [1,3] (agent did not detect any stench there before) Which means that in the current state [2,2] is safe (no pit, no ~~stench~~ Wumpus) → OK ([1,2] → V)

④ Agent moves to [2,2], no stench, no breeze → OK then agent moves to [2,3] where it perceives glitter (gold)

Goal achieved and agent returns to exit of cave.

__/__/____

HEURISTIC SEARCH

☆ Generate & Test

1. Generate candidate solutions from space of possible solutions (search space)
2. Test each candidate with a predicate (function that checks whether the candidate satisfies the problem constraints.)
3. Stop when you find a valid candidate or exhaust the space.

- Exhaustive: Generate all possible candidates and test each one (super slow but guaranteed solution)
- Heuristic: Uses heuristics to avoid generating candidates that are unlikely to work (smart)
- Plan: Make list of candidates using some strategy then apply generate-and-test only on the list.

☆ Hill Climbing

- Generate-and-test + direction to move
  Always move to a state that looks higher (better)
- Uses a heuristic (evaluation) function to estimate how close a state is to the goal.
- Local search method, only looks at neighbouring states

— Simple Hill Climbing

(1) Evaluate initial state.
(2) Loop until goal found or no operators left:
- Pick and apply new operator (generate new state)
- Evaluate new state; if goal → stop
  if better than current state →
  make it the new current state
  else ignore it.

• Move only if new state is strictly better.

— Steepest-Ascent Hill Climbing

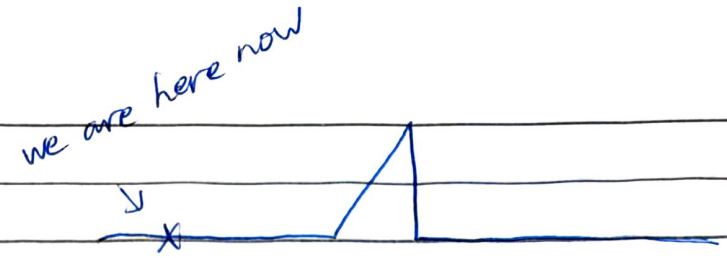
(1) Evaluate initial state
(2) Loop until goal found / no change occurs in full cycle:
- Look at all successor states
- Pick the best among them (highest heuristic value)
- If best successor is better than current → move to it
→ Else → stop (local maxima reached)

• Instead of choosing the first better move, choose the best possible move from all neighbours

— Limitations of Hill Climbing

(a) Local Maximum (peak that is higher than neighbours but not the highest peak (global maxima))
(Algorithm gets stuck since no adjacent state is better)

(b) Plateau (flat region, all neighbours have the same evaluation, no direction to move, stuck)



(c) Ridge (Slope exists but requires a sequence of moves each of which looks worse)
(Not aligned with the moves HC is allowed to make (↑ → ↓ ←), cannot detect uphill diagonal path (↗))

Ex: Blocks World

Start (0)			Goal (h)		(Local)
A	-1		D	+1	
D	+1		C	+1	
C	+1		B	+1	
B	-1		A	+1	

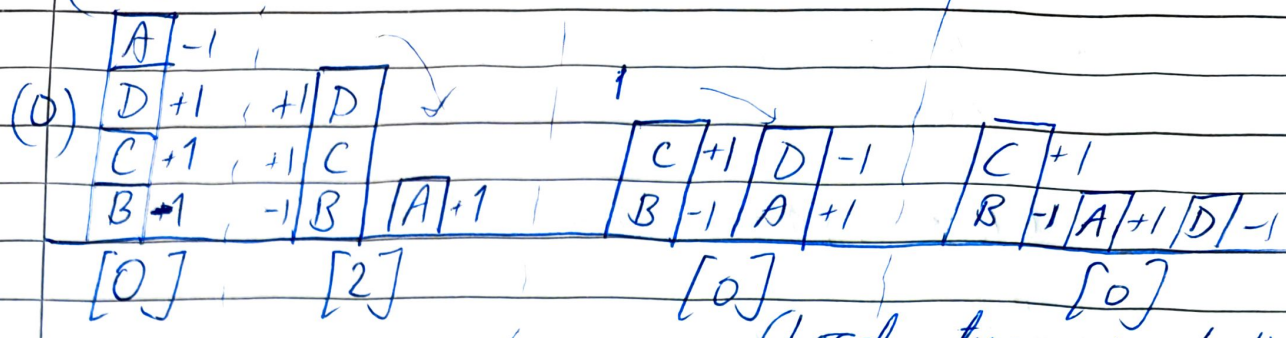
- To move from start state to goal state; we will use the following approaches:

(a) Local heuristic function (Considers immediate consequences of an action to decide next step)

(b) Global heuristic function (Considers a broader view, taking into account, future impact or overall structure of problem to guide the search)

(a) (+1) for each block resting on the block it is supposed to be resting on
(-1) for each block resting on the wrong block / thing (with reference to goal state configuration)
(Local)

(Start)



(Local optimum reached)
(maxima)

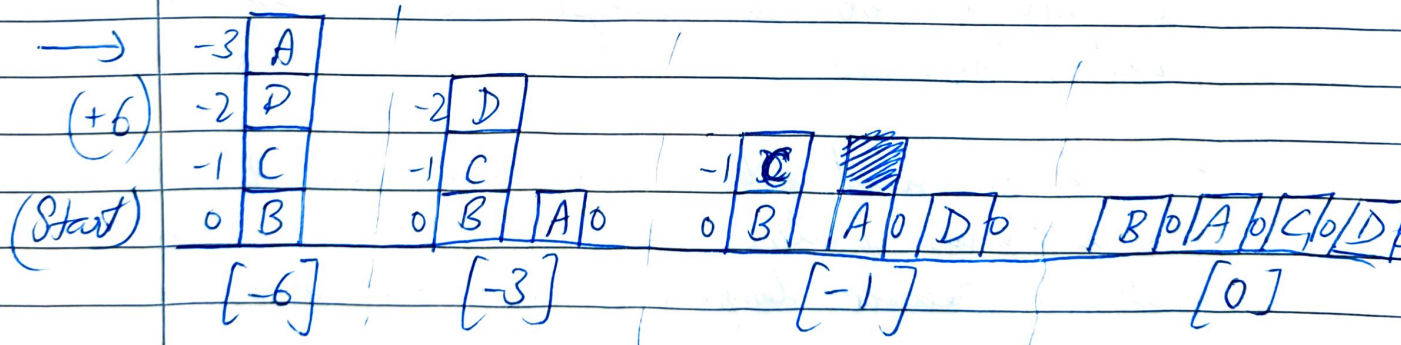
(b) (Global)
For each block that has the correct support structure (+1) to every block in support structure.
For each block that has wrong support structure (-1) to every block in support structure.
(no. of blocks correctly placed below that block (base = 0)

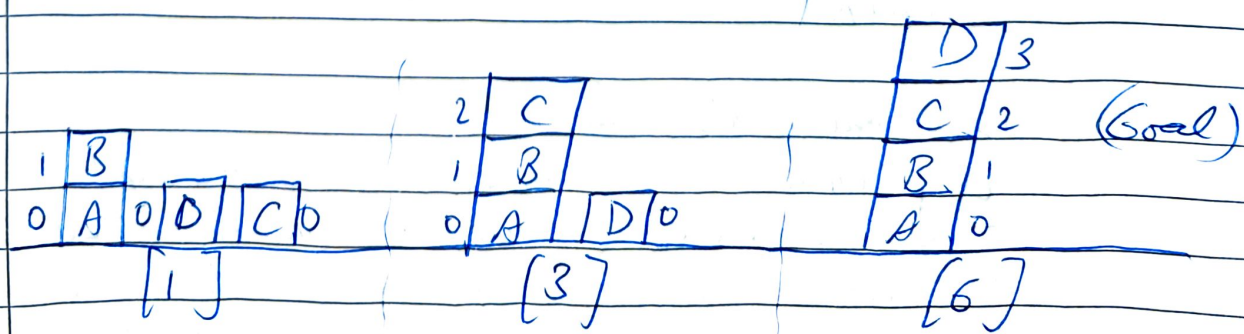
Start (-6)

A	-3
D	-2
C	-1
B	0

D	3
C	2
B	1
A	0

Goal (+6)





__/__/__

- Order of application of operators can make a big differena.
- Global information might be encoded in heuristic functions
- Local information sees only small details, treating each block individually and only looking at their closest neighbours
- Global information gives better understanding of whole structure (looks at entire tower below that block)

★ Simulated Annealing

- Hill Climbing + Controlled randomness
- Instead of only focusing on moving upwards (to better states), simulated annealing allows you to move downwards too, to escape local maxima, plateaus and ridges.
- Early on, start with a high temperature → more randomness (allow some bad moves), then slowly reduce temperature (less randomness) and finally end in a stable state close to global optimum

1. Evaluate initial state
2. Loop until goal is reached / no operators left
   - Set temperature T using an annealing schedule
   - Select a move (operator) and generate a new state.
   - Evaluate new state: if better → accept it

__/__/__

If worse → accept it with a probability.

$$P = \left[e^{-\frac{\Delta E}{kT}} \right]$$

where ΔE → how worse new state is (old - new)

T → temperature

k → constant

- If $\Delta E < 0$ → new state is better
- If $\Delta E > 0$ → new state is worse.

(3) If new state is worse ($\Delta E > 0$), compute the probability then generate a random number between 0 and 1.

(r)

If $r < p$ → accept the worse move

If $r > p$ → reject the worse move

☆ Constraint Satisfaction

- A problem-solving method where variables are assigned values from domains such that all constraints are satisfied.
- Here, CSP is defined by variables, domains (possible values for each variable) and constraints (rules restricting which value combinations are allowed)

Ex: Sudoku, crypt arithmetic puzzles.

- Instead of generating random states, this algorithm eliminates many impossible choices early, restricts search to only valid possibilities and explores the state space intelligently.

(1) Constraint Propagation: Deduce everything possible using constraints (relationships, early contradictions, remove impossible values) (Reduce state space)

__/__/__

(2) Search with Backtracking: Make the right guesses and solve for constraints. If contradiction, backtrack and try another guess.

Ex: Crypt-Arithmetic Puzzle (SEND + MORE = MONEY)

Each letter = different digit
Carry bits satisfy arithmetic relationships
M must be 1 (since MONEY has 5 digits), etc.
Every deduction reduces the domain of possible values until the solution becomes inevitable.

↘

9567 + 1085 = 10652
SEND + MORE = MONEY

- There are two types of rules:

(i) Propagation rules (logical deductions made from existing constraints) (to reduce search space)

Ex: If X=2 and
X+Y=9 → Y=3

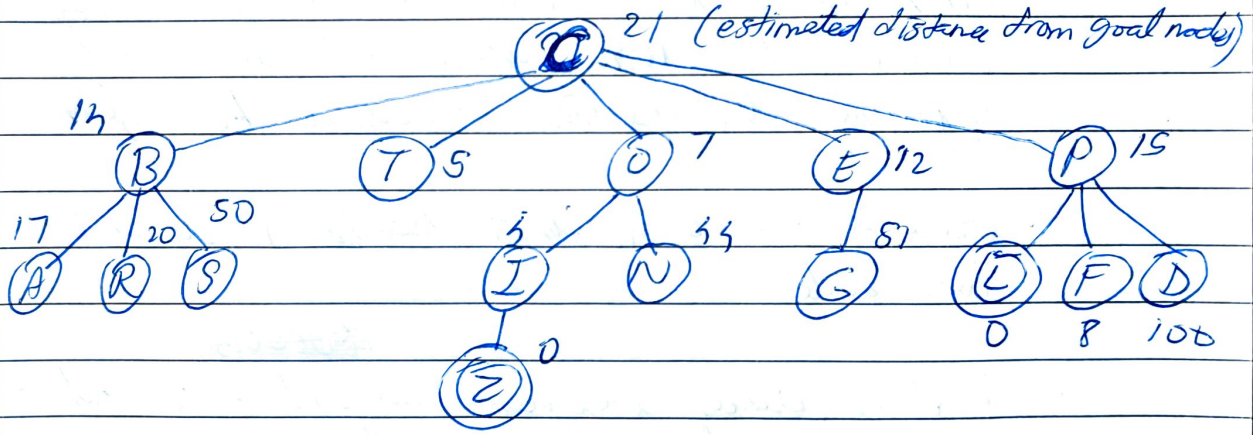
(ii) Guessing rules (when propagation cannot finish the puzzle)
(if contradiction → backtrack)

- Thus reducing substantially the amount of search.

★ Best First Search

- Combines advantages of both BFS and DFS into a single method
- At each step of BFS search process, we select the most promising of the nodes (based on heuristic function) from the priority queue, and then expand the chosen node by using rules to generate it's successors.
- OPEN nodes (generated, not yet examined) (priority queue)
  CLOSED nodes (already examined)

Ex:



OPEN (priority queue)	CLOSED
{T, O, E, B, P}	{C}
{O, E, B, P}	C, T
{I, E, B, P, N}	C, T, O
{Z, E, B, P, N}	C, T, O, I
{E, B, P, N}	C, T, O, I, Z

Goal node has been found
Final path is C → O → I → Z

__/__/____

★ Mean-Ends Analysis

- Many search strategies ~~do~~ traverse either in forward or backward direction, but a mix of these two is usually appropriate to solve a complex and large problem.
- Such a mixed search algorithm allows us to solve a major part of the problem first and then go back to ~~solve the~~ solve the small problems which arise while combining major parts of the problems.
- MEA process is centered on finding the difference between the current state and goal state and applying the operators to reduce the difference
- Operators → a action that transforms current state into new state.

(~~SUCCESS~~) ≠

① Find differences between current state and goal state ~~and goal state~~. If no difference → return (exit)

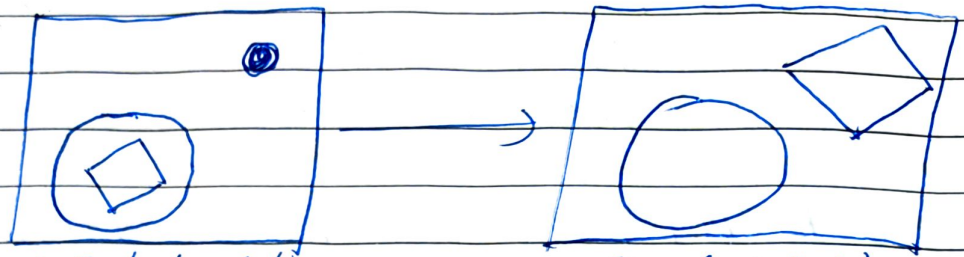
② Reduce difference by doing the following steps until success / failure occurs:
- Select a new operator O applicable for the current difference (precondition, effects) that reduces difference
- Try to apply operator O to current state. By making a description of two states:
(a) O-START (state where preconditions are satisfied)
(b) O-RESULT (resultant state if O applied on O-START)

③ Solve FIRST-PART (MEA (CURRENT, O-START)) and LAST PART (MEA (O-RESULT, GOAL)).

__/__/____

(3) If both parts are successful, then SUCCESS else try another operator.

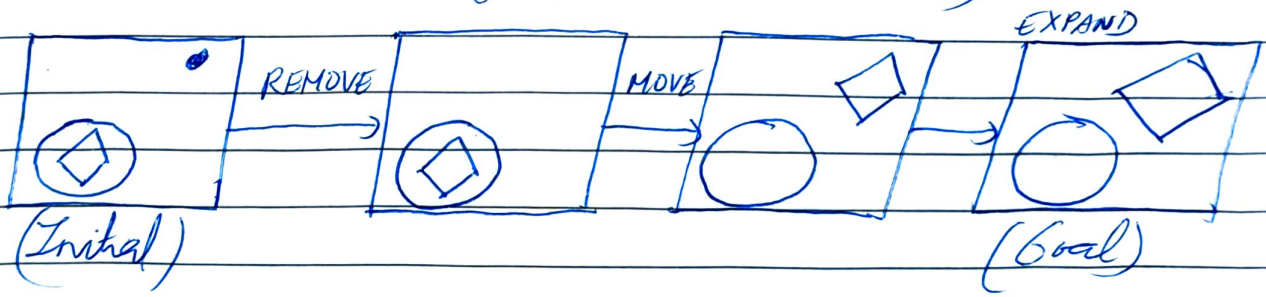
Ex:-



(Initial state) (Goal State)

Operators ⇒ MOVE (move diamond outside circle)
DELETE (delete black circle)
EXPAND (expand / increase size of diamond)

First difference → no black dot in goal state
(Remove it)
Next difference → diamond is outside circle
(Move the diamond)
Next difference → size of ~~square~~ diamond is different
(expand diamond size)



(Initial) (Goal)

Here, say CURRENT = diamond inside the circle
GOAL = diamond outside the circle (top-right)
Difference = (diamond not in goal location)
Most important difference (goal) → Diamond must be at goal position.

To reduce this difference,

__/__/__

Operator = MOVE (diamond, current_location, goal_location)

O-START (state where operator's precondition are satisfied)
MOVE needs diamond to be movable and
goal location must be free.

O-RESULT (state produced by applying MOVE)

Here O-START = current state (since precondition are already true, no subproblems)
O-RESULT = goal state.

FIRST-PART → transform CURRENT into O-START
LAST-PART → apply operator, O-RESULT = goal state