COMP4431 Artificial Intelligence

Knowledge based Agent and Expert System

Raymond Pang
Department of Computing
The Hong Kong Polytechnic University

Agenda

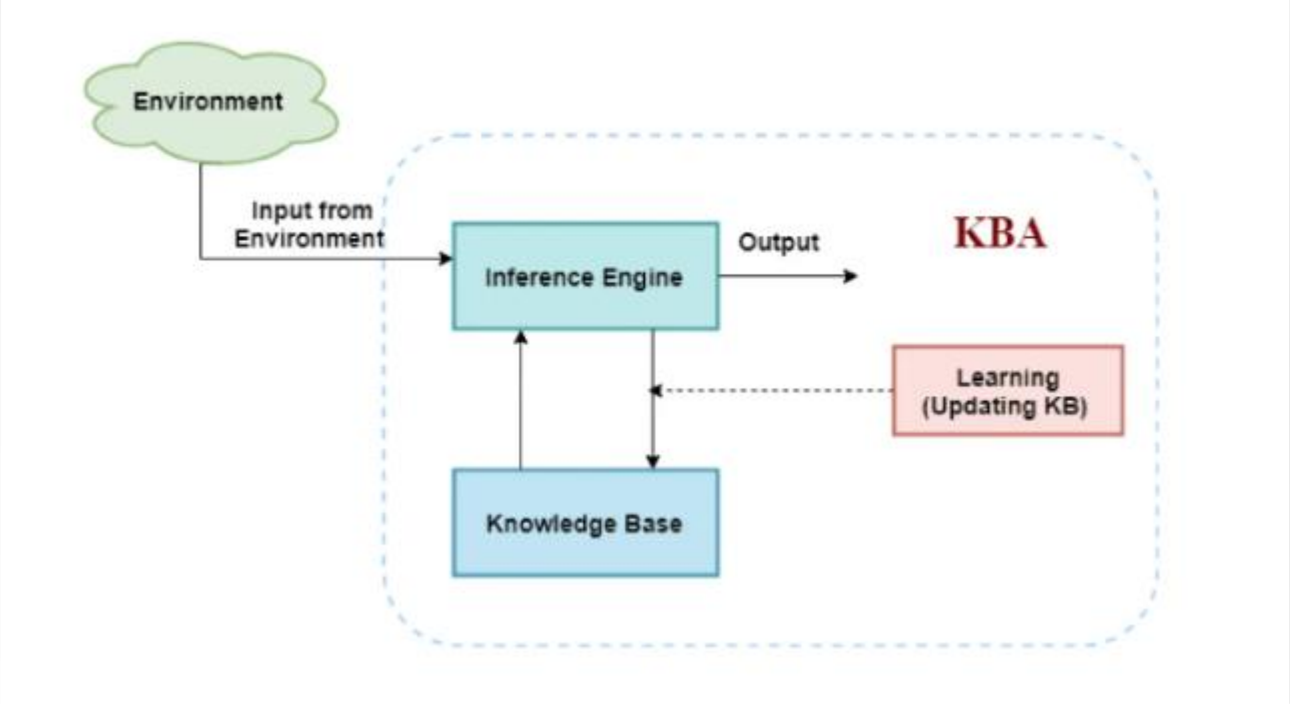
- Knowledge based agent (KBA)
  - Knowledge base and representation
- Inference Engine
  - Forward Chaining
  - Backward Chaining
- Game
  - MiniMax Algorithm
  - Alpha-beta Pruning

Knowledge-based Agent

- An Intelligent agent needs knowledge about the world for taking decisions and reasoning to act efficiently
- Knowledge-based agents (KBA) are those agents who have the capability of
  - maintaining an internal state of knowledge,
  - reason over that knowledge,
  - update their knowledge after observations and,
  - take action.
- These agents can represent the world with some formal representation and act intelligently

Knowledge-based Agent

- KBAs are composed of 2 main parts
  - ❑Knowledge-base
  - ❑Inference Engine



Knowledge-based Agent

- Knowledge base: a central component/repository of all knowledge in the KBA
  - A collection of sentences (here ‘ sentence’ is a technical term and it is not identical to sentence in natural language like English)
  - These sentences are expressed in so-called a knowledge representation language
- In practice, the knowledge bases are built based on human/expert experience
  - It can also be learned by the agent itself, but it will be based on other more complex techniques like machine learning or reinforcement learning, etc.

Inference Rules

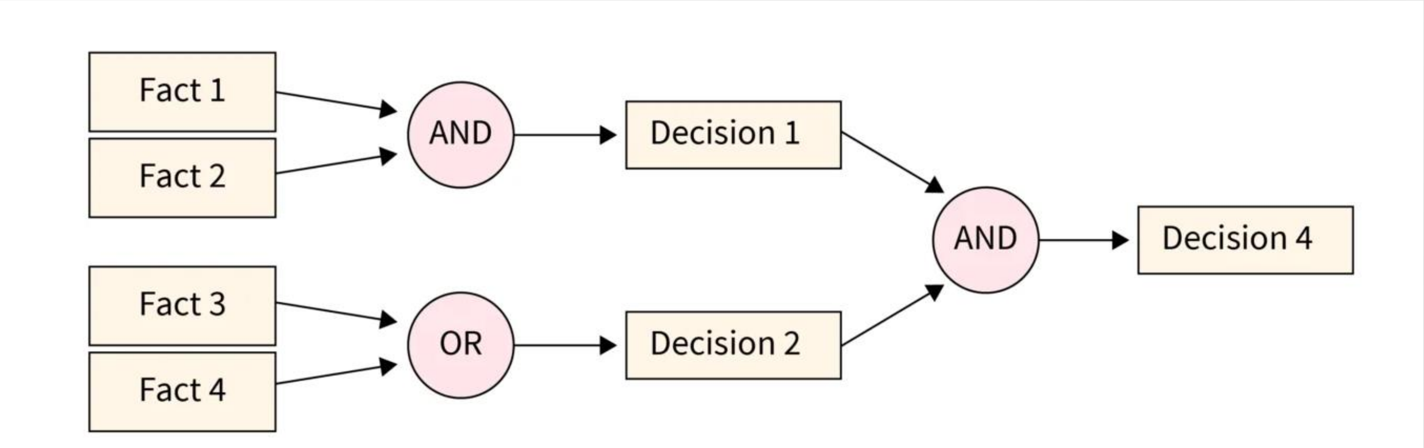
- One common way of knowledge base representation
  - Sometimes referring as Production Rules
- A production rule consists of (condition, decision) pairs which mean “if condition then decision”
  - Here “decision” can also be the condition for another rule, not necessary to be a real action to perform
- We said a rule is fired if its condition is fulfilled

Inference Rules

- A KB can have many rules, e.g.
  - IF(at bus stop AND bus arrives) THEN decision (get into the bus)
  - IF(on the bus AND paid AND empty seat) THEN decision (sit down)
  - IF (on bus AND unpaid) THEN decision (pay charges)
  - IF (on the bus AND sit down) THEN decision (play mobile phone)
  - IF (bus arrives at destination) THEN decision (get down from the bus)
- As one rule is fired, it may trigger another rule to be fired. There can be a chain reaction (referred as **inference chain**)!

Forward Chaining

- Forward chaining is a method of reasoning in the Inference Engine
- Inference rules are applied to existing data to extract additional data until an endpoint (goal) is achieved.
- This approach is often used in expert systems for tasks such as troubleshooting and diagnostics.



Forward Chaining Steps

- The system is given one or more facts (from the environment or user input)
- Then the rules are searched in the KB for each fact. Rules that fulfil the conditions are selected (i.e. IF part)
- Now each rule is able to produce new conditions from the conclusion of the invoked one.
- The added conditions are processed again by repeating step 2. The process will end if there is no new conditions exist.

Example of Forward Chaining: A simple weather forecast system

- The system starts with KB :
  - Rule 1

    IF the ambient temperature is above 90°F

    THEN the weather is hot
  - Rule 2

    IF the relative humidity is greater than 65%

    THEN the atmosphere is humid
  - Rule 3

    IF the weather is hot and the atmosphere is humid

    THEN thunderstorms are likely to develop

Example of Forward Chaining: A simple weather forecast system

From:

- Fact 1 (Input)

  the ambient temperature is 92°F

- Rule 1

  IF the ambient temperature is above 90°F

  THEN the weather is hot

- The inference engine can deduce:

  the weather is hot

Example of Forward Chaining: A simple weather forecast system

From:

- Fact 2 (Input)

  the weather is hot
- Fact 3 (Input)

  the atmosphere is humid
- Rule 3

  IF the weather is hot and the atmosphere is humid

  THEN thunderstorms are likely to develop

- The inference engine can deduce:

  thunderstorms are likely to develop

Properties of Forward Chaining

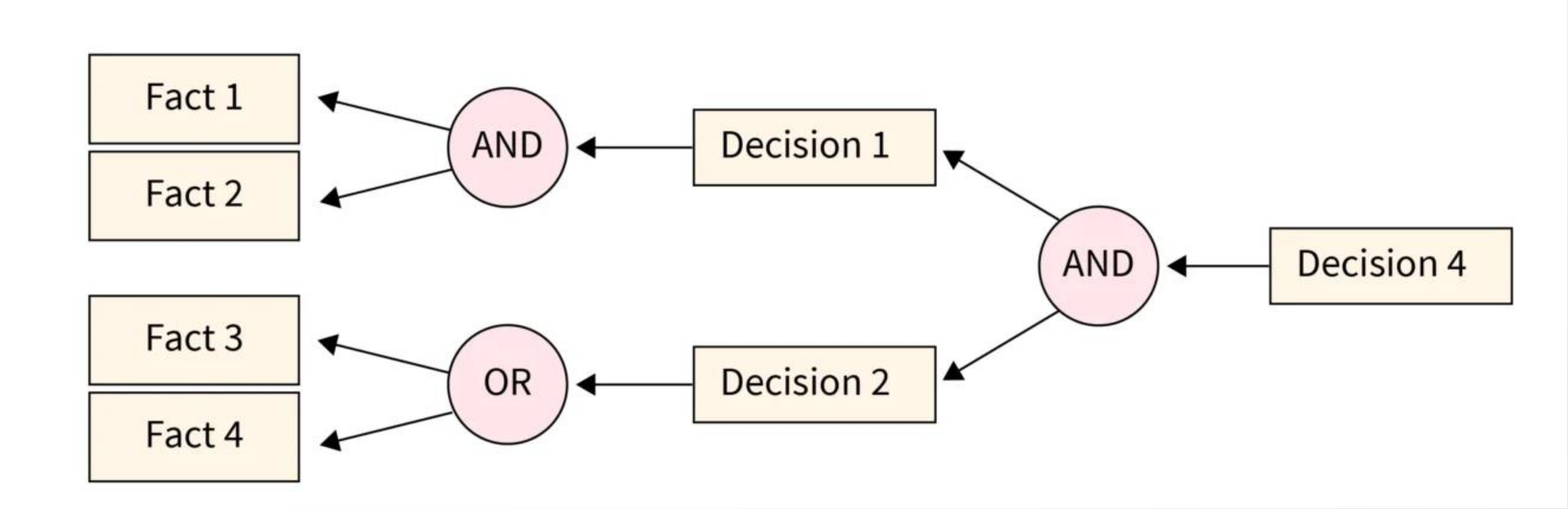
- **Data-Driven:** The reasoning starts from available data (facts) and works toward a goal.
- **Bottom-Up Approach:** It builds knowledge from facts, gradually moving towards conclusions.
- **Breadth-First Search Strategy:** The inference engine explores multiple rules simultaneously, applying them step by step.
- **Possibility of Irrelevant Rules:** Forward chaining may explore rules that do not contribute to the final solution, making it less efficient in some cases.

Ways of Reasoning

- Forward chaining / reasoning
  - Start with all the known data and progress toward the conclusion
- An alternative: we select a possible conclusion and try to prove its validity by looking for supporting evidence
  - It's like what a detective does!

Backward Chaining

- Backward chaining is a goal-driven reasoning strategy for inference.
- It starts with a goal and works backward to determine if the available facts support the goal.



Example of Backward Chaining

- Goal:
  - Does the patient have the flu?
- Rule:
  - IF the patient has a fever and sore throat, THEN they might have the flu.
- Sub-goals:
  - Verify if the patient has a fever.
  - Verify if the patient has a sore throat.

Backward Chaining

- **Goal-Driven:** Reasoning begins with a desired goal and searches for evidence to support it.
- **Top-Down Approach:** The system starts from the goal and works back to find relevant facts.
- **Depth-First Search Strategy:** The inference engine follows a path deeply before exploring other possibilities, prioritizing each goal or sub-goal in sequence.
- **Possibility of Infinite Loops:** If not handled properly, backward chaining may get stuck in loops while looking for evidence to support the goal.

A Simple Medical Diagnosis System

- Rule 1

IF (nasal congestion and viremia)

THEN influenza

- Rule 2

IF (runny nose)

THEN nassal congestion

- Rule 3

IF (body-aches> 100)

THEN achiness

- Rule 4

IF (temperature > 100)

THEN fever

- Rule 5

IF (headache)

THEN achiness

- Rule 6

IF (fever and achiness and cough)

THEN viremia

Forward Chaining Results

- For Input :
  - Runny nose
  - Temperature = 101.7
  - Headache
  - Cough
- Trace of rule execution

Execution Cycle	Premises	Selected Rule	Derived Fact
1	runny nose	Rule 2	nassal congestion
2	temperature	Rule 4	fever
3	headache	Rule 5	achiness
4	fever, achiness, cough	Rule 6	viremia
5	nassal congestion, viremia	Rule 1	influenza

Backward Chaining Results

- Initial goal : diagnosis influenza
- Trace of rule execution

Execution Cycle	Goal resolving	Rule fired	Goals created & accumulated	Premise satisfied
1	influenza	Rule 1	nassal congestion, viremia	nil
2	nassal congestion	Rule 2	runny nose, viremia	runny nose
3	viremia	Rule 6	fever, achiness, cough	cough
4	fever	Rule 4	Temperature > 100, achiness	Temperature > 100
5	achiness	Rule 3	Body-aches, achiness	nil
5	achiness	Rule 5	headache	headache
6	All goals are cleared and satisfied premises, so this hypothesis was correct, subject has influenza			

Game

Games

- Multiagent environment
  - In which each agent needs to consider the actions of other agents
  - And how they affect its own welfare
- Games
  - Views multiagent environment as a game, provided that the impact of each agent on the other is significant
- Games in AI
  - The state of a game is easy to represent
  - Agents are usually restricted to a small number of actions
  - Outcomes are defined by precise rules

Game Playing Strategy

- Maximize winning possibility assuming that opponent will try to minimize (Minimax Algorithm)
- Ignore the unwanted portion of the search tree (Alpha Beta Pruning)
- Evaluation(Utility) Function
  - A measure of winning possibility of the player

Formulate Game to Search Problem

- S_0
  - The initial state, which specifies how the game is set up at the start
- Player(*s*)
  - Defines which players has the move in a state
- Actions(*s*)
  - Defines a set of legal move in a state
- Result(*s, a*)
  - The transition model, which defines the result of a move
- Terminal-Test(*s*)
  - A terminal test, which is true when the game is over
- Utility(*s, p*)
  - A utility function, also called an objective function or payoff function
  - Defines the final numeric value for a game that ends in terminal state *s* for a player *p*

Tic-Tac-Toe

X	O	

e(p) = 6 - 5 = 1

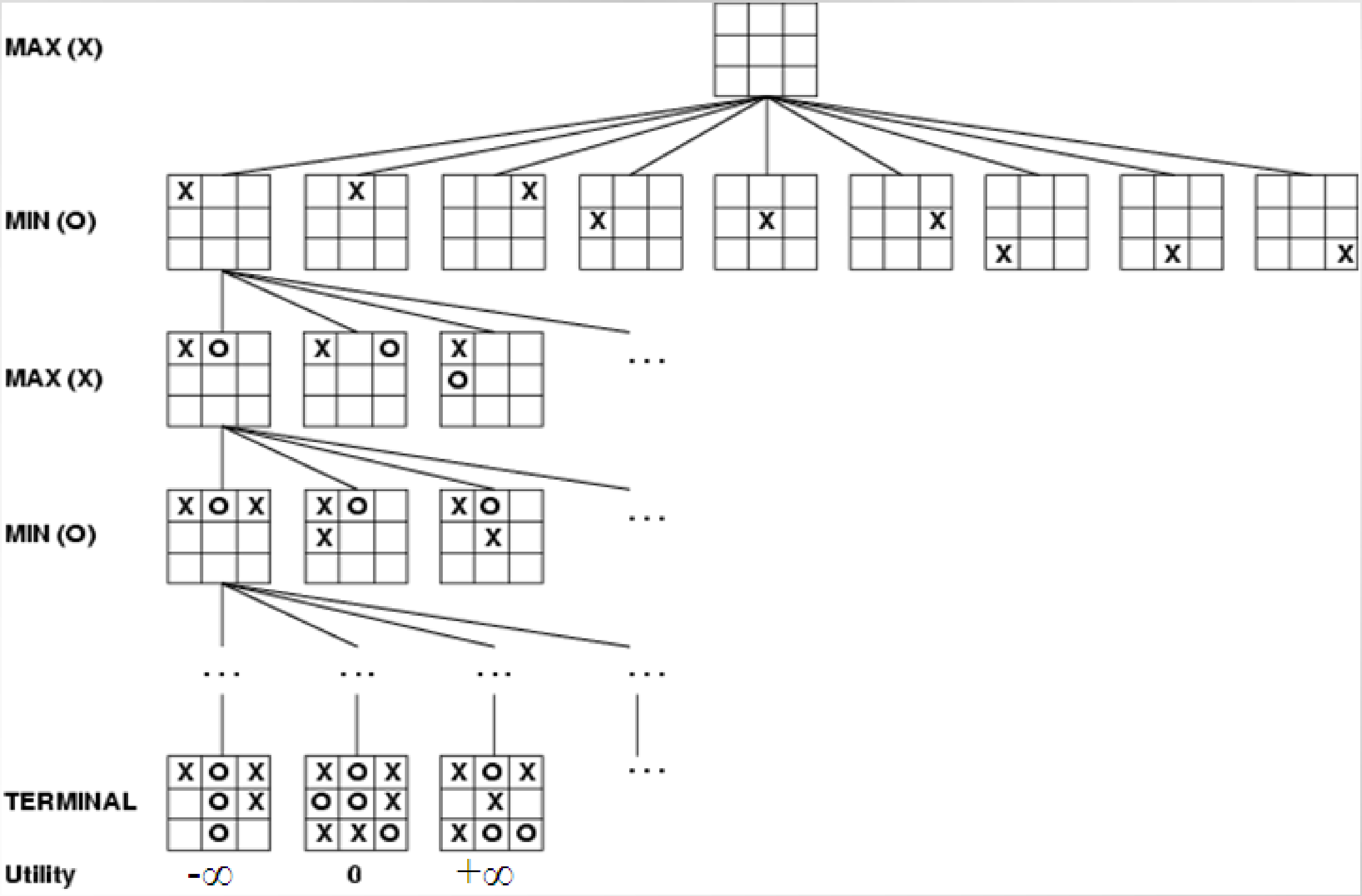
- **Initial State**: Board position of 3x3 matrix with 0 and X.
- **Actions:** Putting 0's or X's in vacant positions alternatively
- **Terminal test:** Which determines game is over
- **Utility function**:

  e(p) = (No. of complete rows, columns or diagonals are still open for player) – (No. of complete rows, columns or diagonals are still open for opponent)

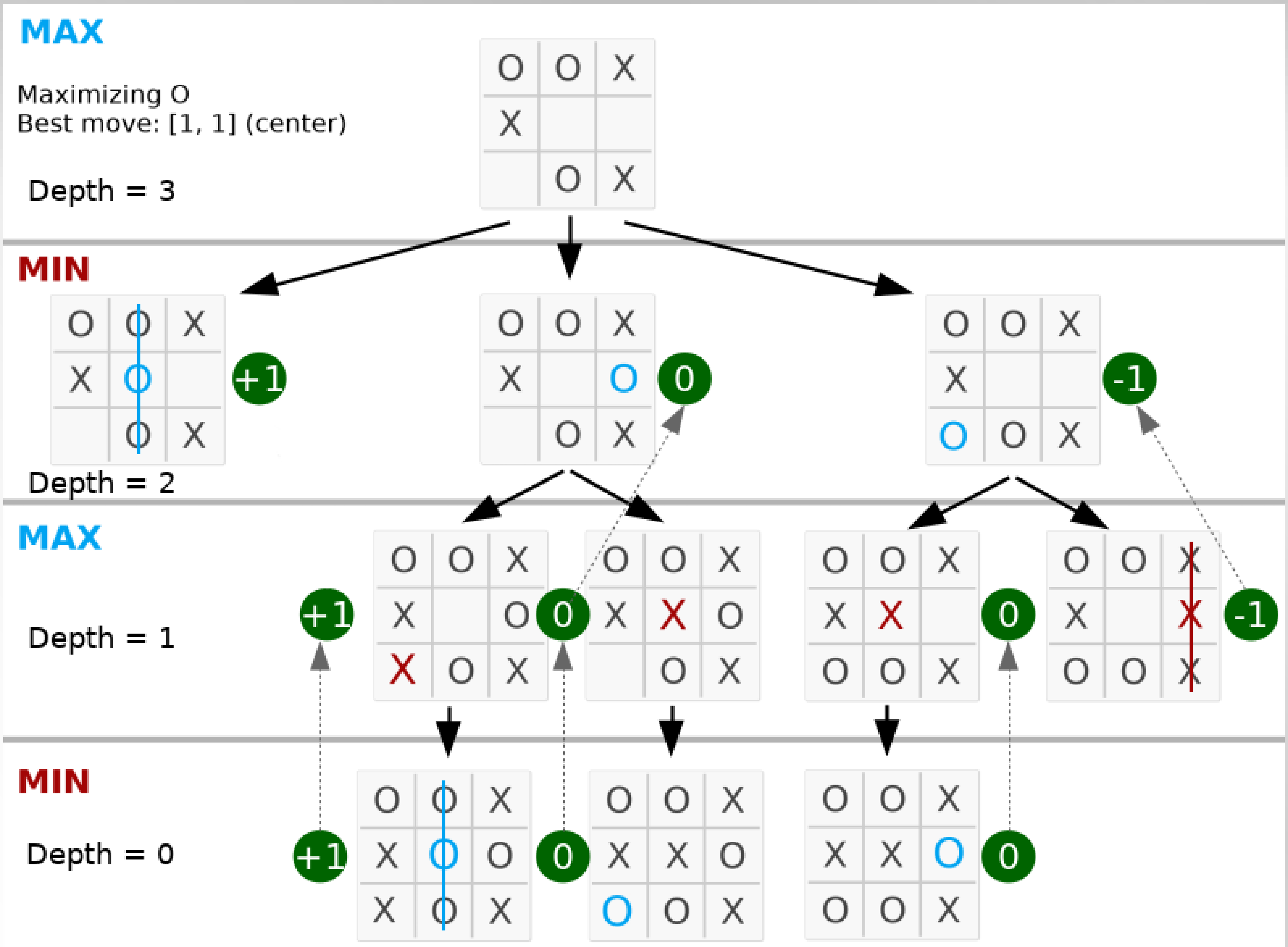
Minimax Algorithm

- Generate the game tree
- Apply the utility function to each terminal state to get its value
- Use these values to determine the utility of the nodes one level higher up in the search tree
  - From bottom to top
  - For a max level, select the maximum value of its successors
  - For a min level, select the minimum value of its successors
- From root node select the move which leads to highest value

Game tree for Tic-Tac-Toe



Courtesy : Artificial Intelligence and Soft Computing. Behavioural and Cognitive Modelling of the Human Brain



https://github.com/Cledersonbc/tic-tac-toe-minimax

Properties of Minimax

- Complete : Yes (if tree is finite)
- Time complexity : $O(b^d)$
- Space complexity : O(bd) (depth-first exploration)

Observation

- Minimax algorithm, presented above, requires expanding the entire state-space.
- Severe limitation, especially for problems with a large state-space.
- Some nodes in the search can be *proven to be irrelevant to the outcome* of the search

Alpha-Beta Strategy

- Maintain two bounds:

  Alpha (α): a lower bound on best that the player to move can achieve

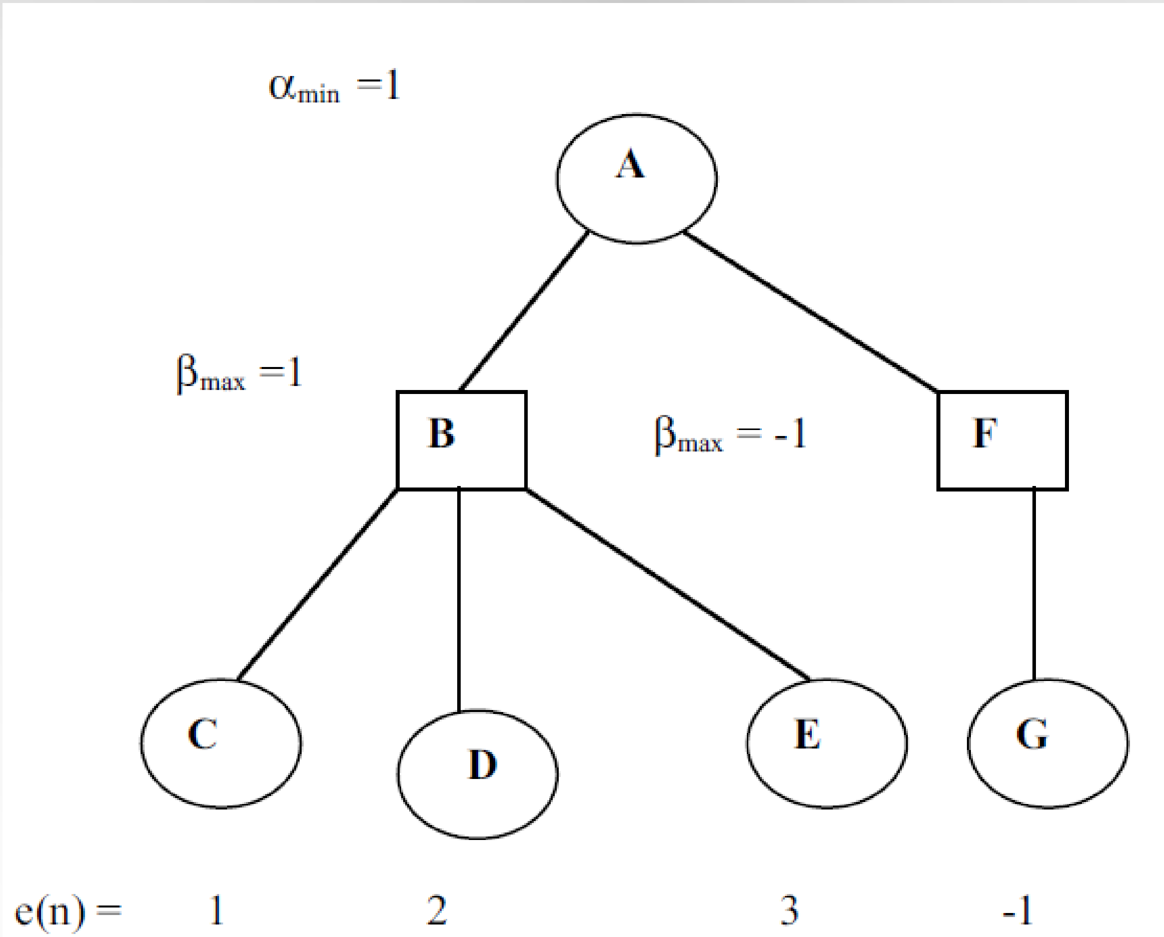
  Beta (β): an upper bound on what the opponent can achieve
- Search, maintaining α and β
- Whenever $\alpha \geq \beta_{higher}$, or $\beta \leq \alpha_{higher}$ further search at this node is irrelevant

How to Prune the Unnecessary Path

- **If** beta value of any MIN node below a MAX node is less than or equal to its alpha value, **then** prune the path below the MIN node.

- **If** alpha value of any MAX node below a MIN node exceeds the beta value of the MIN node, **then** prune the nodes below the MAX node.

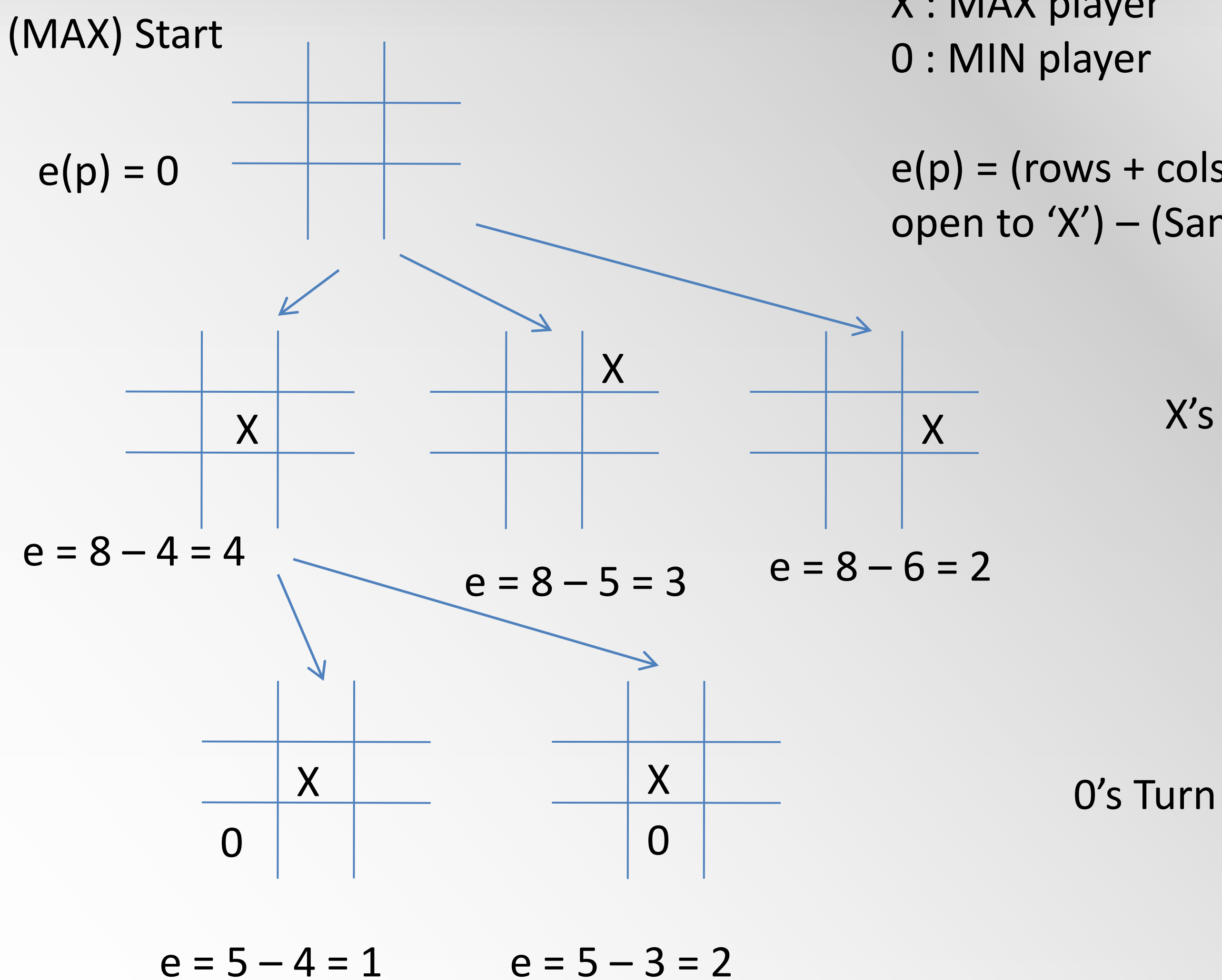
Example



Tic-Tac-Toe

X : MAX player
0 : MIN player

e(p) = (rows + cols + diagonals open to ‘X’) – (Same to ‘0’)



Alpha-Beta Search Algorithm

- If the MAXIMIZER nodes already possess αmin values, then their current **αmin value = Max (αmin value, α'min);** on the other hand, if the MINIMIZER nodes already possess βmax values, then their current

  βmax value = Min (βmax value, β'max).

- If the estimated βmax value of a MINIMIZER node N is less than the αmin value of its parent MAXIMIZER node N' then there is no need to search below the node MINIMIZER node N. Similarly, if the αmin value of a MAXIMIZER node N is more than the βmax value of its parent node N' then there is no need to search below node N.

Alpha-Beta Analysis

- Pruning does not affect the final result.
- Assume a fixed branching factor and a fixed depth
- Best case: $b^{d/2} + b^{(d/2)-1}$
- Approximate as $b^{d/2}$
- Impact ?

  Minmax: 10^9 = 1,000,000,000

  Alpha-beta: $10^5 + 10^4$ = 110,000
- But best-case analysis depends on choosing the best move first at cut nodes (not always possible)
- The worst case : No cut-offs, and Alpha-Beta degrades to Minmax

Summary

- Knowledge based agent (KBA)
  - Knowledge base and representation
- Inference Engine
  - Forward Chaining
  - Backward Chaining
- Game
  - MiniMax Algorithm
  - Alpha-beta Pruning