AI_Module3

Topics:

1. Informed Search Strategies:
    a. **Greedy best-first search,**
    b. **A*search,**
    c. **Heuristic functions**.
2. Logical Agents:
    a. **Knowledge–based agents,**
    b. **The Wumpus world,**
    c. **Logic,**
    d. **Propositional logic, Reasoning patterns in Propositional Logic**

3. 1 Informed Search Strategies

Informed Search: Informed search is a search strategy that utilizes problem-specific knowledge, to find solutions more efficiently. Informed search methods make use of heuristics and evaluation functions to guide the search towards more promising paths.

Heuristic Function(h(n)): It is a heuristic function that provides an estimate of the cost from the current node to the goal node. This heuristic is admissible if it never overestimates the true cost to reach the goal. In other words, **h(n)** is always less than or equal to the actual cost.

Actual cost function(g(n)): It is Cost of the path from the **start node to node n**. It represents the actual cost incurred to reach the current node from the initial node. For the initial node (start node), **g(n)** is usually 0.

Evaluation Function (f(n)): The evaluation function, denoted as **f(n),** is the total estimated cost of the cheapest path from the start node to the goal node that passes through node n. It is the sum of g(n) and h(n): **f(n) = g(n) + h(n).**

f(n) represents the priority of a node. Nodes with lower f(n) values are explored first, making the algorithm prioritize paths that are likely to be more efficient.

3.1.a Greedy Best First Search Algorithm

Greedy best first search tries to expand the node that is closest to the goal to lead to a solution quickly. It evaluates the nodes by using just the heuristic function i.e. for Greedy best first search, f(n) = h(n).

Let's explore the application of this method to route-finding challenges in Romania for the map given in figure 3.2.

We will employ the straight-line distance heuristic, denoted as h_{SLD}. Specifically, for our destination in **Bucharest**, we require knowledge of the straight-line distances to Bucharest, as illustrated in Figure 3.22.

As an illustration, consider h_{SLD}(In(Arad)) = 366. It's important to note that the values of h_{SLD} cannot be derived directly from the problem description. Furthermore, understanding that h_{SLD} correlates with actual road distances and serves as a valuable heuristic requires a certain level of experience.

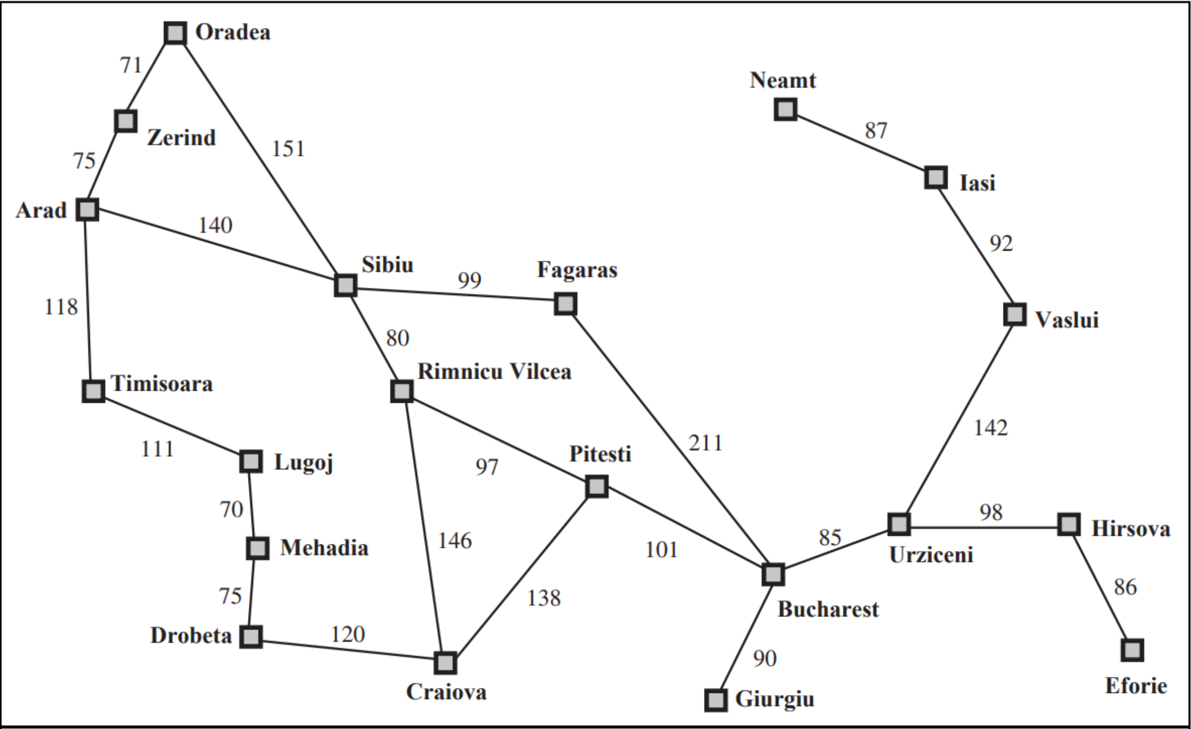
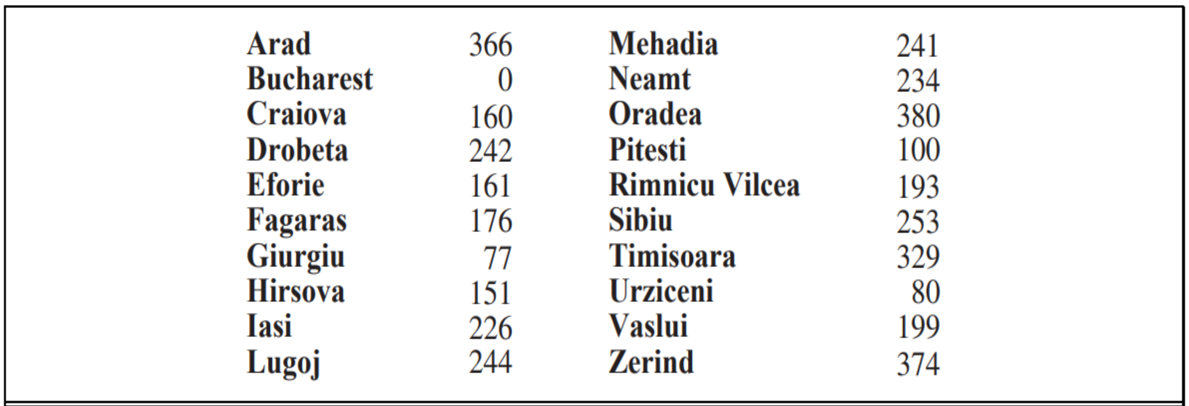


Figure 3.2 A simplified road map of part of Romania.



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

Following Figure shows the progress of a greedy best-first search using h_{SLD} to find a path from Arad to Bucharest.

The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras because it is closest. Fagaras in turn generates Bucharest, which is the goal.

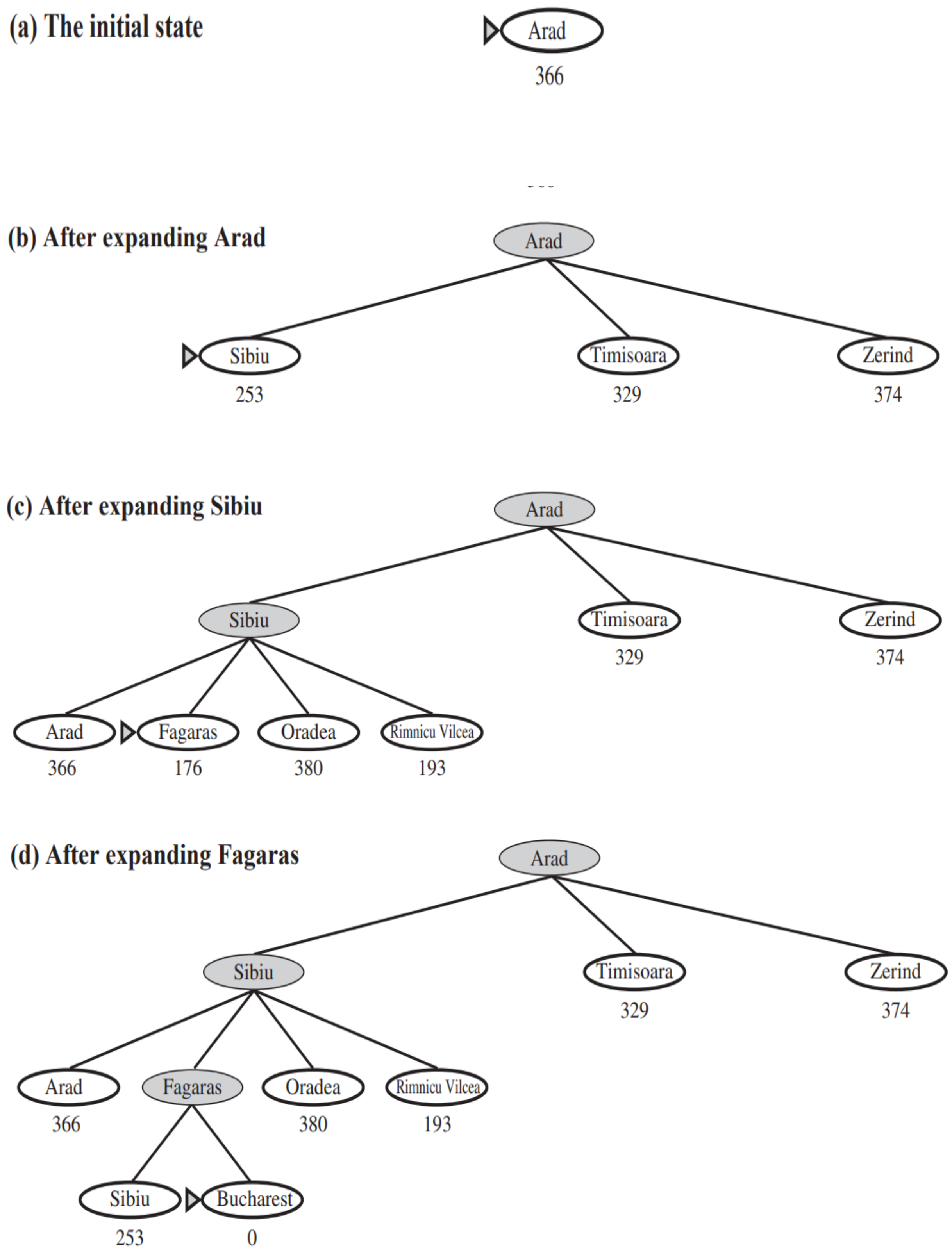
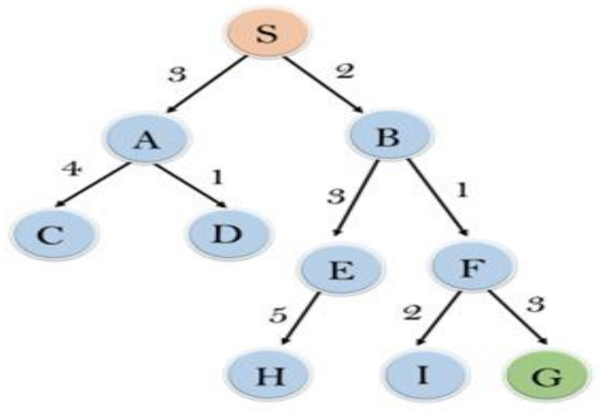


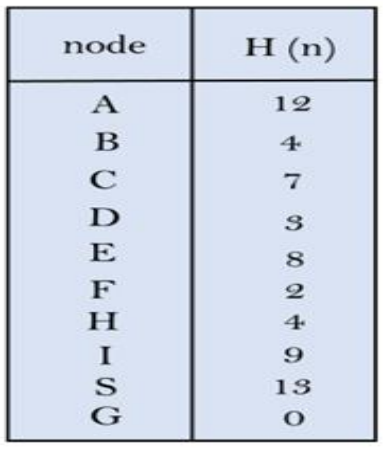
Figure: Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD} . Nodes are labelled with their h-values.

Best first search algorithm:

Step 1: Place the starting node into the OPEN list.
Step 2: If the OPEN list is empty, Stop and return failure.
Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
Step 4: Expand the node n, and generate the successors of node n.
Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
Step 7: Return to Step 2.

Example: Consider the tree and heuristic values given below:

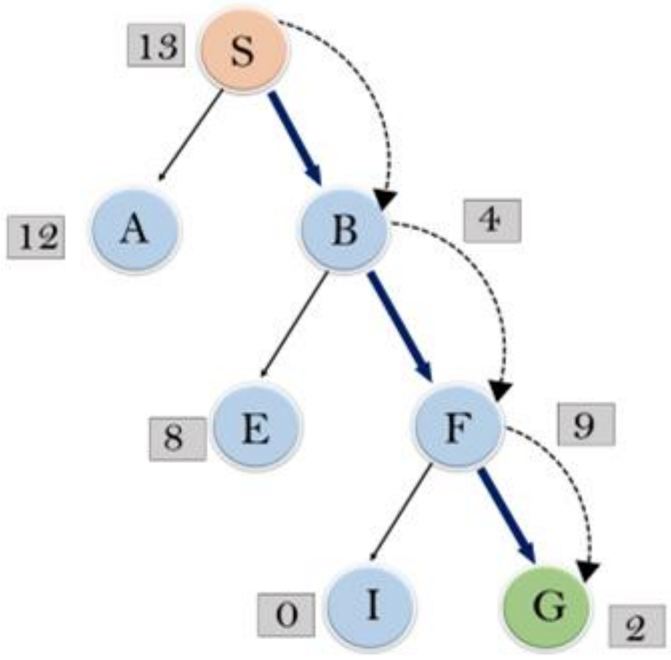




node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.

Step	OPEN List	CLOSED List	Details
Initialization	[A, B]	[S]	
Iteration1: Expand B	[A]	[S,B]	h(B)<h(A)
Iteration2:	[E,F,A]	[S,B]	
Expand F	[E,A]	[S,B,F]	H(F)<H(E),H(A)
Iteration3:	[I,G,E,A]	[S,B,F]	
Visit Goal G	[I,E,A]	[S,B,F,G]	H(G)< H(E),H(A),H(I)



Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is O(b^m).

Space Complexity: The worst case space complexity of Greedy best first search is O(b^m). Where, m is the maximum depth of the search space.

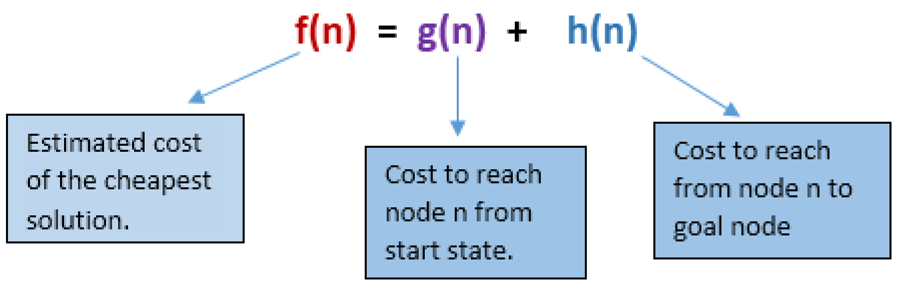
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

3.1.b A* Search Algorithm

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster.

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

A* search Algorithm:

Step1: Place the starting node in the OPEN list.
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
Step 6: Return to Step 2.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example1: Let's explore the application of this method to route-finding challenges in Romania for the map given in figure 3.2 .

We will employ the straight-line distance heuristic, denoted as h_{SLD}. Specifically, for our destination in Bucharest, we require knowledge of the straight-line distances to Bucharest, as illustrated in Figure 3.22.

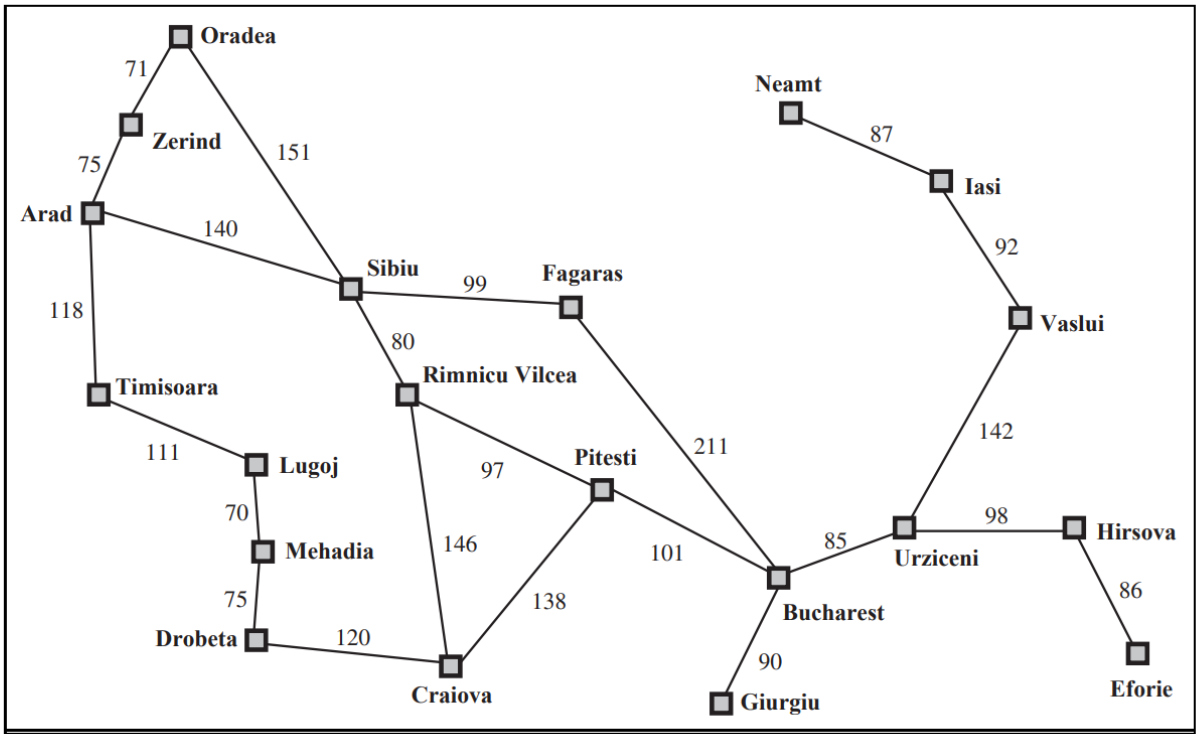
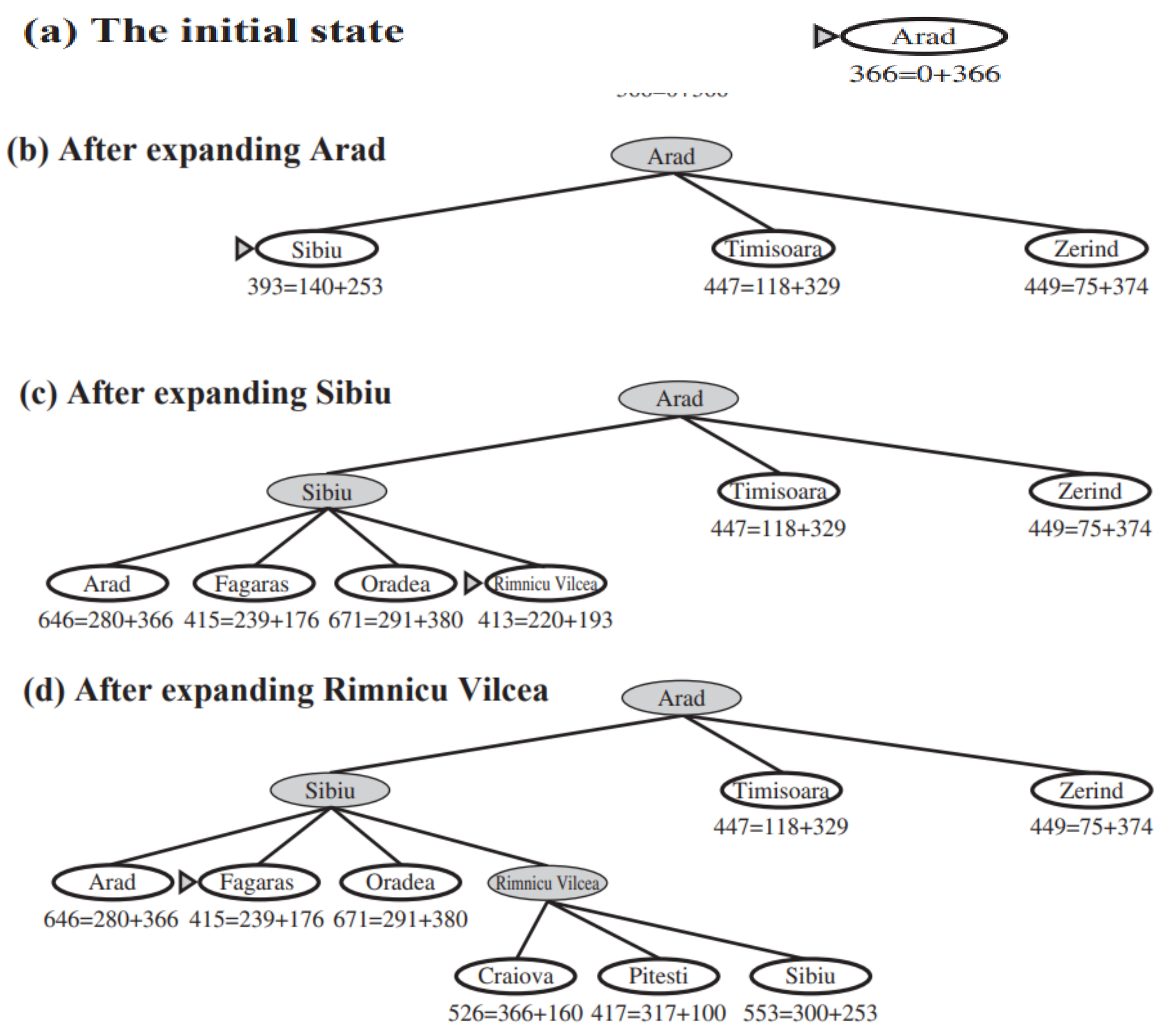


Figure 3.2 A simplified road map of part of Romania.

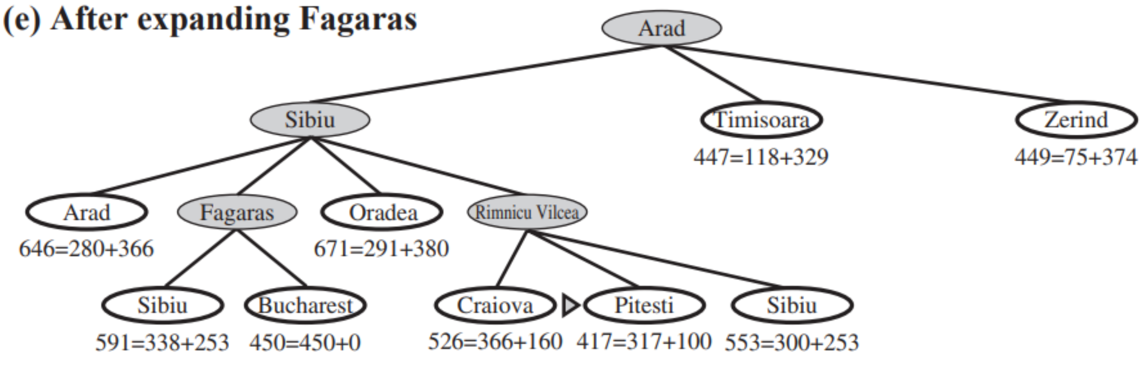
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

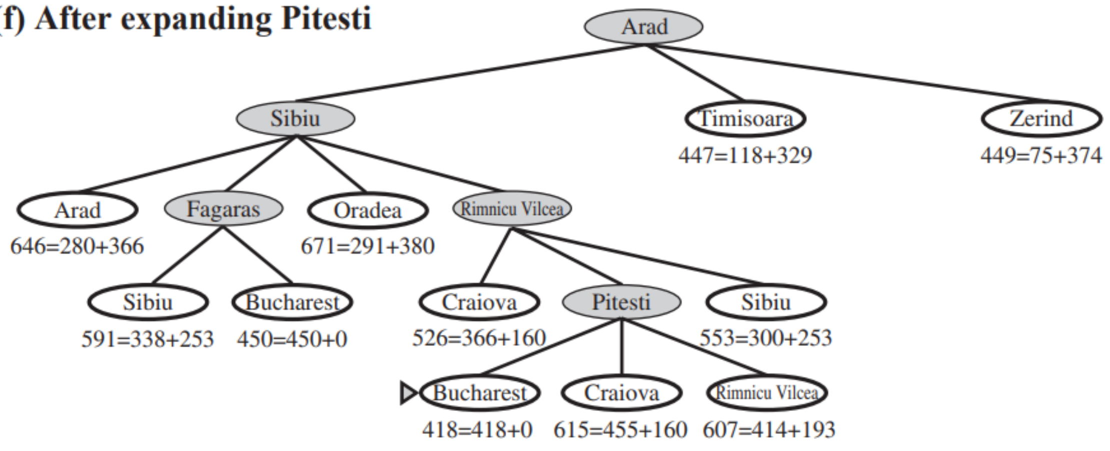
Figure below illustrates the Stages in **an A∗ search for Bucharest**. Nodes are labelled with f = g + h. The h values are the straight-line distances to Bucharest



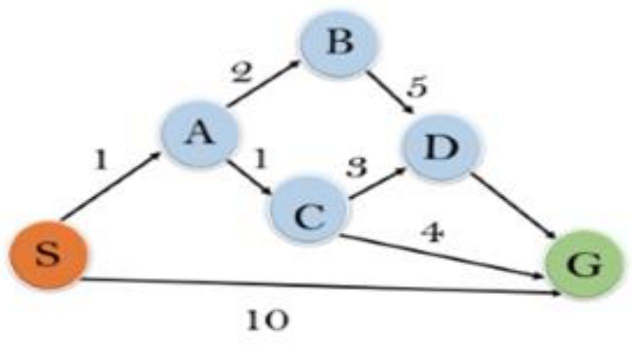
(e) After expanding Fagaras

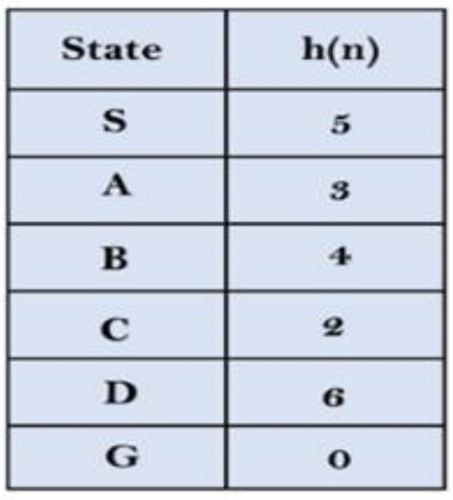


(f) After expanding Pitesti



Example 2: In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state. Here we will use OPEN and CLOSED list**.**





State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:

To apply the A* Search algorithm to the given graph, we will use the following steps:

Step 1: Define the Components

- **g(n)**: The cost from the start node S to the current node n.
- **h(n)**: The heuristic cost (as provided in the table) from the current node n to the goal G.
- **f(n) = g(n) + h(n)**: The total estimated cost of the path through node n.

Step 2: Initialize Lists

- **Open List**: Contains nodes to be evaluated. Initially, it contains only the start node S.
- **Closed List**: Contains nodes that have been evaluated.

We will begin from S, the starting node.

Step 3: Stepwise Solution

Initial State:

- **Open List**: [S]
- **Closed List**: [ ]
- **f(S) = g(S) + h(S) = 0 + 5 = 5**

Step3 : Iteration1

Select the node from the Open List with the lowest f(n) (i.e., S).
Expand S. Its neighbors are A and G.

- g(A)=1, h(A)=3 → f(A)=1+3=4
- g(G)=10,h(G)=0 → f(G)=10+0=10

Add A and G to the Open List and move S to the Closed List.

- **Open List**: [A(4),G(10)]
- **Closed List**: [S]

Step3 : Iteration2

Select A (smallest f(n)).
Expand A. Its neighbors are B and C.

- g(B)=1+2=3, h(B)=4 → f(B)=3+4=7
- g(C)=1+1=2, h(C)=2 → f(C)=2+2=4

Add B and C to the Open List, and move A to the Closed List.

- **Open List**: [C(4), B(7)]
- **Closed List**: [S, A]

Step3 : Iteration3

Select node with the lowest f(n): C(smallest f(n)=4).
Expand C. Its neighbours are D and G.
g(D)=5, h(D)=6, so f(D)=g(D)+h(D)=5+6=11.
g(G)=6, h(G)=0, so f(G)=g(G)+h(G)=6+0=6.
Add G and D to the Open List. Move C to the Closed List.
Open List: [G(6),B(7),D(11)]

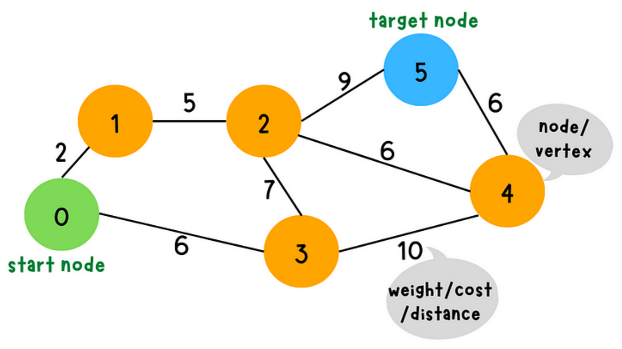
Closed List: [S,A,C]

Step3 : Iteration4
Select node with the lowest f(n): G (**smallest f(n)=6**).
G is the goal node, so the algorithm terminates here.

Final Solution
The path found is S→A→C→G.
Total Cost: The total cost is the path S→A→C→G which is 1+1+4=6.

Exercise: Solve the following using A* Algorithm

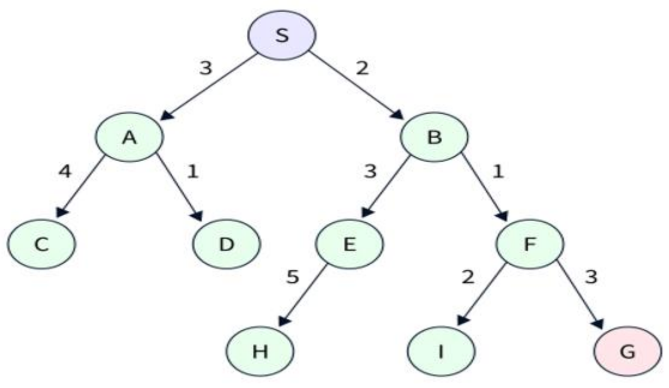
1.



h(n)

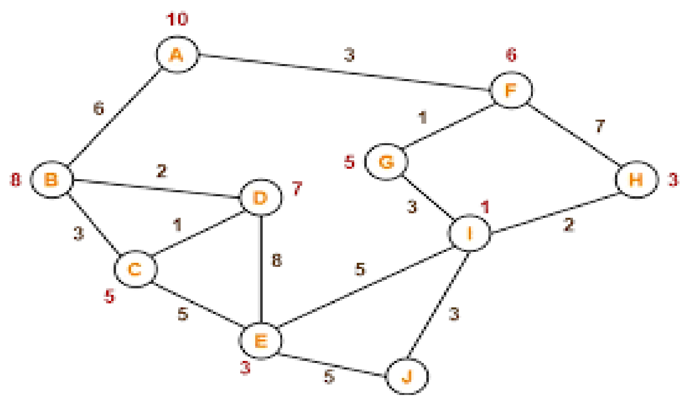
node	Heuristic distance to target node
0	20
1	16
2	6
3	10
4	4
5	0

2.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

3.



3.1.b.1 Points to remember:

A* Algorithm and First-Occurrence Path: The A* algorithm is designed to return the first path it finds from the start node to the goal node. Once a valid path is discovered, A* terminates its search, making it more efficient in scenarios where finding a single optimal solution is sufficient.

Quality of Heuristic: The effectiveness of the A* algorithm is significantly influenced by the quality of the heuristic function **h(n))**. A well-designed heuristic provides a good estimate of the remaining cost from a given node to the goal, guiding **A*** to explore more promising paths and enhancing its efficiency.

Node Expansion Condition: A* algorithm expands nodes based on the evaluation function **$f(n)=g(n)+h(n)$,** where:
$g(n)$ is the cost of the path from the start node to node n,
$h(n)$ is the heuristic estimate of the cost from node n to the goal node.
The algorithm expands nodes that satisfy the condition $f(n)\leq M$, where M is a specified threshold or maximum cost. This condition ensures that A* explores nodes within a predefined cost limit, allowing for efficient pathfinding without exhaustively searching the entire space.

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

3.1.b.2 Conditions for optimality: Admissibility and consistency:

A* search algorithm is optimal if it follows below two conditions:

Admissible:

The first condition requires for optimality is that **h(n)** should be an admissible heuristic for **A* tree** search. An admissible heuristic is one that never overestimates the cost to reach the goal. Because g(n) is the actual cost to reach n along the current path, and f(n) = g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n. If the heuristic function is admissible, then A* tree search will always find the least cost path.

Consistency (or sometimes monotonicity):

Second required condition is consistency for only **A* graph**-search. A heuristic h(n) is consistent if, for every node n and every successor n ′ of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n ′ plus the estimated cost of reaching the goal from n ′ : $h(n) \le c(n, a, n') + h(n')$.
This is a form of the general **triangle inequality**, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides. Here, the triangle is formed by n, n ′ , and the goal **Gn** closest to n.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is $O(b^d)$, where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is $\mathbf{O(b^d)}$

3.1.c Heuristics Functions

Heuristic Functions **h(n)** guide search algorithms by estimating the cost or distance to a goal state from the current state (n).

Consider the 8-puzzle game. The object of the 8 puzzle is to slide the title horizontally or vertically into the empty space until the configuration matches the goal configuration.

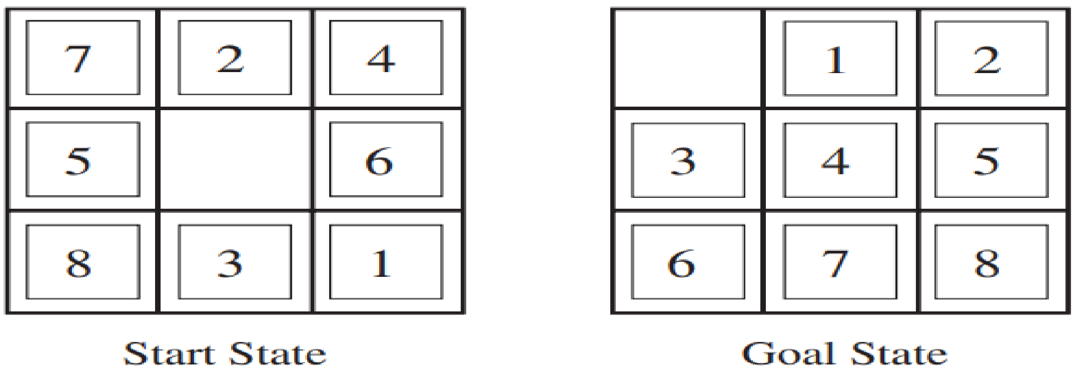


Figure illustrates the A typical instance of the 8-puzzle. The solution is 26 steps long.

The average solution cost for a randomly generated 8-puzzle instance is about 22 steps. The branching factor is about 3. (When the empty tile is in the middle, four moves are possible; when it is in a corner, two; and when it is along an edge, three.) This means that an exhaustive tree search to depth 22 would look at about $3^{22} \approx 3.1 \times 10^{10}$ states.

The two commonly used candidates for 8 puzzles are as follows:

h1 = the number of misplaced tiles. For Figure, all of the eight tiles are out of position, so the start state would have h1 = 8. h1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once

h2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the city block distance or Manhattan distance. h2 is also admissible because all any move can do is move one tile one step closer to the goal. Tiles 1 to 8 in the start state give a Manhattan distance of h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 .

As expected, neither of these overestimates the true solution cost, which is 26. The performance of heuristic search algorithms depends on the quality of the heuristic function. One can sometimes construct good heuristics by relaxing the problem definition, by storing precomputed solution costs for subproblems in a pattern database, or by learning from experience with the problem class.

1. The effect of heuristic accuracy on performance:

Experimentally it is determined that h_2 is better than h_1. That is for any node n, $h_2(n) \geq h_1(n)$. This implies that h_2 dominate h_1. Domination translates directly into efficiency. A^* using h_2 will never expand more nodes than A^* using h_1.

2. Generating admissible heuristics from relaxed problems:

A problem with fewer restrictions on the actions is called a relaxed problem. The state-space graph of the relaxed problem is a super graph of the original state space because the removal of restrictions creates added edges in the graph. Because the relaxed problem adds edges to the state space, any optimal solution in the original problem is, by definition, also a solution in the relaxed problem; but the relaxed problem may have better solutions if the added edges provide short cuts. Hence, the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

For example, if the 8-puzzle actions are described as

- A tile can move from square **A** to square **B** if
- **A** is horizontally or vertically adjacent to **B** and **B** is blank,

we can generate three relaxed problems by removing one or both of the conditions:

a) A tile can move from square A to square B if A is adjacent to B.
b) A tile can move from square A to square B if B is blank.
c) A tile can move from square A to square B.

If a collection of admissible heuristics h1 . . . hm is available for a problem and none of them dominates any of the others, which should we choose? As it turns out, we need not make a choice. We can have the best of all worlds, by defining h(n) = max{h1(n), . . . , hm(n)}

3.Generating admissible heuristics from subproblems: Pattern databases:

Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem. For example, Figure below shows a subproblem of the 8-puzzle instance.

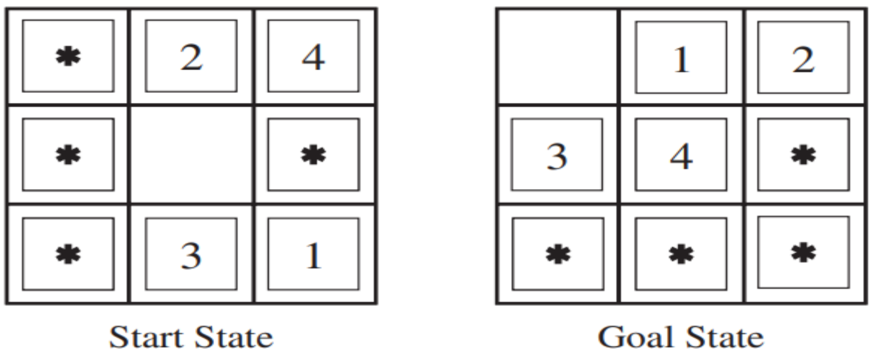


Fig: A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

The subproblem involves getting tiles 1, 2, 3, 4 into their correct positions. Clearly, the cost of the optimal solution of this subproblem is a lower bound on the cost of the complete problem. It turns out to be more accurate than Manhattan distance in some case.
The idea behind pattern databases is to store these exact solution costs for every possible subproblem instance—in our example, every possible configuration of the four tiles and the blank. (The locations of the other four tiles are irrelevant for the purposes of solving the subproblem, but moves of those tiles do count toward the cost.) Then we compute an admissible heuristic h_{DB} for each complete state encountered during a search simply by looking up the corresponding subproblem configuration in the database.

4 Learning heuristics from experience:

A heuristic function, denoted as h(n), aims to approximate the solution cost starting from the state represented by node **n**. One of the strategies is to learning from practical experiences. In this context, "experience" refers to solving numerous instances of problems like 8-puzzles.
In each optimal solution to an 8-puzzle, valuable examples emerge, comprising a state along the solution path and the actual cost of reaching the solution from that particular point.
Employing these examples, a learning algorithm can be employed to generate a heuristic function, **h(n),** with the potential to predict solution costs for other states encountered during the search process.

Inductive learning methods are most effective when provided with relevant features of a state for predicting its value, rather than relying solely on the raw state description.

For instance, a feature like "**number of misplaced tiles**" (**$x_1(n)$**) can be useful in predicting the distance of a state from the goal in an 8-puzzle. By gathering statistics from randomly generated 8-puzzle configurations and their actual solution costs, one can use these features to predict h(n).

Multiple features, such as **$x_2(n)$** representing the "number of pairs of adjacent tiles that are not adjacent in the goal state," can be combined using a linear combination approach:

$$h(n) = c_1x_1(n) + c_2x_2(n).$$

The constants (**c1** and **c2**) are adjusted to achieve the best fit with the actual data on solution costs. It is expected that both **c_1** and **c_2** are positive, as misplaced tiles and incorrect adjacent pairs make the problem more challenging. While this heuristic satisfies the condition **$h(n) = 0$** for goal states, it may not necessarily be both admissible and consistent.

3.2.a Logical Agents

What are Logic Agents?

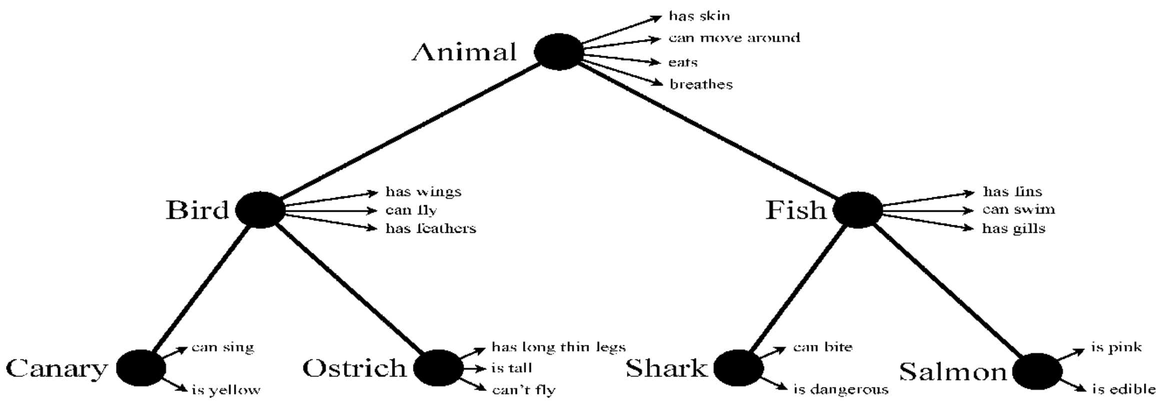
Stuart Russell and Peter Norvig, in their influential textbook "**Artificial Intelligence: A Modern Approach**," describe logical agents as those that operate based on knowledge representation and logical inference.
According to their **framework**, an agent perceives its environment through sensors, maintains an internal state (knowledge base), and acts upon the environment through effectors. **Logical agents** specifically use logical reasoning to make decisions.

What is Knowledge Representation?
Knowledge representation in **artificial intelligence (AI)** refers to the process of creating a **structured and formalized representation of information** in a way that can be used by a computer system to **reason, make decisions, or perform tasks.** The goal is to model knowledge in a manner that facilitates effective **problem-solving, learning, and communication within an AI system.**

Example 1 : Semantic Networks

Semantic networks are a graphical representation of knowledge that **uses nodes to represent concepts and arcs (edges)** to represent relationships between these concepts.
Each node in the network represents an **entity or concept**, and the **arcs depict the relationships between** them.
This form of knowledge representation is often used to model **hierarchies, associations, and dependencies**.

Knowledge Representation using Semantic Networks



Example 2: Knowledge Representation using Propositional logic

In propositional logic, knowledge is represented using propositions, which are statements that can be either **true or false**. Logical operators such as **AND, OR, and NOT** are used to combine propositions.
Consider the following knowledge about a weather prediction system:
P: It is raining.
Q: The sky is cloudy.
R: The weather forecast predicts rain.

Now, we can represent some logical relationships:

- If the sky is cloudy **(Q)**, and the weather forecast predicts rain **(R)**, then we can infer that it might be raining **(P)**. This relationship can be represented as: $(Q \wedge R) \rightarrow P$.
- If it is not raining (NOT P), then the weather forecast predicting rain **(R)** must be false. This relationship can be represented as: $\neg P \rightarrow \neg R$.

What is Logical Reasoning?
Logical reasoning is a cognitive process of **making inferences or drawing conclusions** based on **logical principles, rules, and relationships.** It involves analyzing information and using **valid deductive or inductive arguments** to reach a sound or reasonable conclusion. Logical reasoning is an essential aspect **of problem-solving, decision-making, and critical thinking**.

Example: Syllogism
A syllogism is a form of deductive reasoning where a conclusion is drawn from **two given or assumed propositions** (premises).

- **Premise 1**: All humans are mortal.
- **Premise 2:** Socrates is a human.
- **Conclusion**: **Therefore, Socrates is mortal.**

Knowledge based Agents
A knowledge-based agent is a type of intelligent agent that makes decisions and takes actions based on **knowledge** it possesses. A knowledge-based agent is characterized by its ability to represent and manipulate knowledge in a structured way, allowing it to reason, make decisions, and take actions based on the information stored in its knowledge base.

Key Components of Knowledge base:

1. **Knowledge Base (KB) :** The central component of a knowledge-based agent is its **knowledge base**. The **knowledge base** is a collection of sentences expressed in a knowledge representation language. Each sentence represents an assertion about the world. The knowledge base is where the agent stores information that it uses to **make decisions and take actions**. Sometimes we dignify a sentence with the name axiom, when the sentence is taken as given without being derived from other sentences.

2. **Knowledge Representation Language:** The sentences in the knowledge base are expressed in a language called a knowledge representation language. This language allows the agent to formally represent information about the world in a way that the agent can understand and manipulate.

3. **Axioms:** Some sentences in the knowledge base may be dignified with the name "axiom," especially when they are taken as given without being derived from other sentences. Axioms are fundamental statements that serve as foundational knowledge for the agent.

4. **TELL Operation:** There is a mechanism for adding new sentences to the knowledge base. This operation is referred to as TELL. It allows the agent to incorporate new information into its knowledge base.

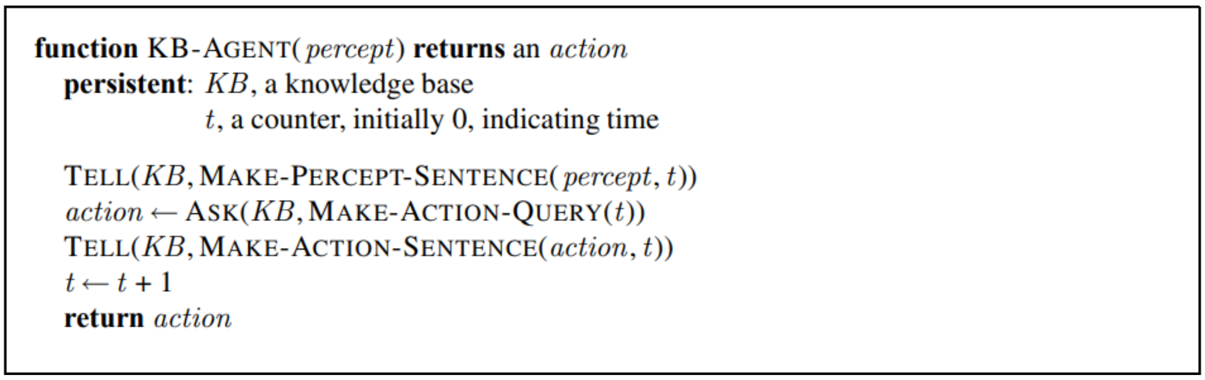
5. **ASK Operation:** The agent needs a way to query the knowledge base to retrieve information. The standard operation for querying is referred to as ASK. It allows the agent to ask questions about what is known.

6. **Inference :** Both TELL and ASK operations may involve inference, which is the process of deriving new sentences from existing ones. Inference must adhere to the requirement that answers derived from the knowledge base follow logically from the information previously TELLed to the knowledge base.

7. **Background Knowledge:** The knowledge base may initially contain some background knowledge. This knowledge provides a foundational understanding of the environment in which the agent operates.

8. **Agent Program:** The knowledge-based agent program outlines the overall structure of the agent. It takes a percept (input) and returns an action as output. The agent program incorporates the knowledge base and other components to facilitate decision-making and action-taking.

   Agent Program: Figure below illustrates a generic knowledge-based agent. Given a **percept**, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action



```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Agents' **knowledge and goals is more important:** At the knowledge level, we only need to specify the

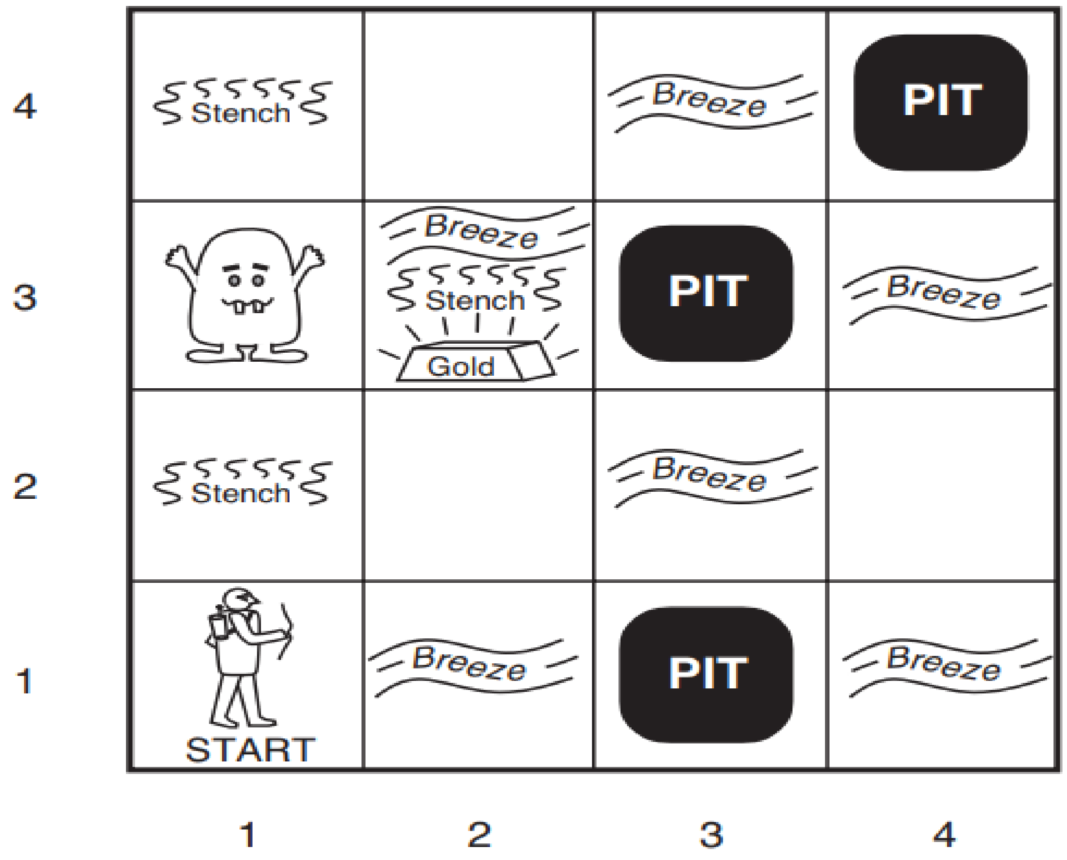
- agent's knowledge and
- goals to determine its behavior.

For instance, consider an **automated taxi** with the goal of transporting a passenger from **San Francisco to Marin County**. If the taxi knows that the **Golden Gate Bridge is the sole link** between these locations, we can expect it to cross the bridge, understanding that it aligns with its goal.

Importantly, this analysis remains independent of the taxi's implementation details.
Whether its geographical knowledge is represented through ***linked lists, pixel maps, or if it reasons using symbolic strings stored in registers or through neural network signal propagation***, the behavior is determined solely by its **knowledge and goals.**

3.2.b The Wumpus World

The **wumpus** world is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the terrible **wumpus**, a beast that eats anyone who enters its room. The **wumpus** can be shot by an agent, but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the **wumpus**, which is too big to fall in). The only mitigating feature of this bleak environment is the possibility of finding a heap of gold.
A typical wumpus world is illustrated in the figure below. The agent is in the bottom left corner, facing right.



PEAS description of Wumpus World

- **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.

- **Environment**: A 4 × 4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.
- **Actuators:** The agent can **move Forward, TurnLeft by 90◦ , or TurnRight by 90◦** . The agent dies a miserable death if it enters a square containing a **pit or a live** wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and **bumps** into a wall, then the agent does not move. The action **Grab** can be used to pick up the gold if it is in the same square as the agent. The action **Shoot** can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the **wumpus or hits a wall**. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action **Climb** can be used to climb out of the cave, but only from square [1,1]
- **Sensors:** The agent has five sensors, each of which gives a single bit of information:
  1. In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.
  2. In the squares directly adjacent to a pit, the agent will perceive a Breeze.
  3. In the square where the gold is, the agent will perceive a Glitter.
  4. When an agent walks into a wall, it will perceive a Bump.
  5. When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

5 Percept Symbols : [Stench, Breeze, Glitter, Bump, Scream].
The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a **stench** and a **breeze**, but no **glitter**, **bump**, or **scream**, the agent program will get **[Stench, Breeze, None, None, None].**

The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [None, None, None, None, None]. (b) After one move, with percept [None, Breeze, None, None, None]

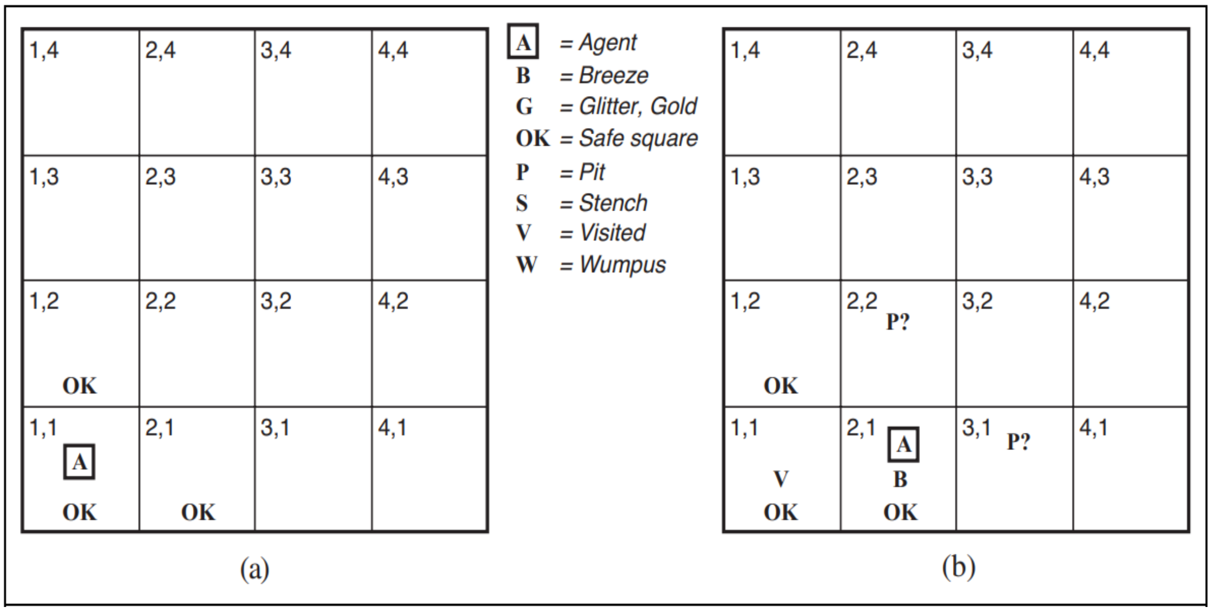


Fig 7.3

Two later stages in the progress of the agent. **(a)** After the third move, with percept [Stench, None, None, None, None]. **(b)** After the fifth move, with percept [Stench, Breeze, Glitter , None, None].

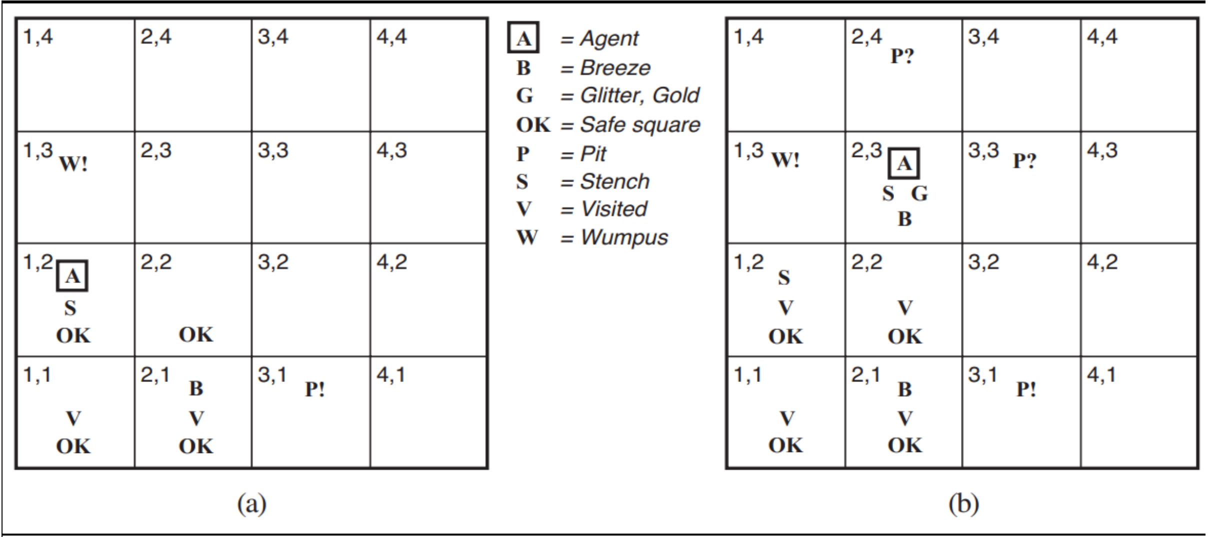


Fig 7.4

- The agent's initial knowledge base contains the rules of the environment, as described previously; in particular, it knows that it is in [1,1] and that [1,1] is a safe square; we denote that with an "A" and "OK," respectively, in square [1,1].
- The first percept is [None, None, None, None, None], from which the agent can conclude that its neighboring squares, [1,2] and [2,1], are free of dangers—they are OK. Figure 7.3(a) shows the agent's state of knowledge at this point.

- A cautious agent will move only into a square that it knows to be OK. Let us suppose the agent decides to move forward to [2,1]. The agent perceives a breeze (denoted by "B") in [2,1], so there must be a pit in a neighboring square. The pit cannot be in [1,1], by the rules of the game, so there must be a pit in [2,2] or [3,1] or both. The notation "P?" in Figure 7.3(b) indicates a possible pit in those squares. At this point, there is only one known square that is OK and that has not yet been visited. So the prudent agent will turn around, go back to [1,1], and then proceed to [1,2].
- The agent perceives a stench in [1,2], resulting in the state of knowledge shown in Figure 7.4(a). The stench in [1,2] means that there must be a wumpus nearby. But the wumpus cannot be in [1,1], by the rules of the game, and it cannot be in [2,2] (or the agent would have detected a stench when it was in [2,1]). Therefore, the agent can infer that the wumpus is in [1,3]. The notation W! indicates this inference. Moreover, the lack of a breeze in [1,2] implies that there is no pit in [2,2]. Yet the agent has already inferred that there must be a pit in either [2,2] or [3,1], so this means it must be in [3,1]. This is a fairly difficult inference, because it combines knowledge gained at different times in different places and relies on the lack of a percept to make one crucial step.
- The agent has now proved to itself that there is neither a pit nor a wumpus in [2,2], so it is OK to move there. We do not show the agent's state of knowledge at [2,2]; we just assume that the agent turns and moves to [2,3], giving us Figure 7.4(b). In [2,3], the agent detects a glitter, so it should grab the gold and then return home.

3.2.c Logic:

In AI Logic is a fundamental component of logical representation and reasoning. It enables machines to understand and represent data and knowledge in a reasoning way. Logical reasoning is a process of inferring a conclusion based on observations or data. It is concerned with the principles of reasoning and how conclusions can be drawn from given premises. Logic provides the theoretical foundation for reasoning.

Types of Logic:

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0…1
Fuzzy logic	degree of truth	degree of belief 0…1

Logics are formal languages for representing information such that conclusions can be drawn. Syntax defines the sentences in the language. Semantics define the meaning of sentences i.e. define truth of a sentence in a world.

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
$x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

Syntax : Syntax refers to the structure and rules governing the formation of sentences or expressions in a language or representation system.Syntax is the set of rules that dictate which sentences are well-formed in the representation language. For example, "x + y = 4" adheres to the syntax, while "x4y+ =" does not.

Semantics: Semantics deals with the meaning of sentences or expressions in a language. It specifies the truth or falsehood of sentences in relation to possible worlds or situations. **Semantics** defines the truth of each sentence in relation to possible worlds. For instance, the sentence "x + y = 4" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.

Model: A model is a mathematical abstraction that represents a possible world in the context of logic. It is used to fix the truth or falsehood of sentences based on specific assignments of values to variables. **Models** serve as precise mathematical abstractions of **possible worlds**. They fix the truth or falsehood of sentences based on assignments of real numbers to variables. The term "**model**" is used interchangeably with "**possible world**." Informally, we may think of a possible world as, for example, having x men and y women sitting at a table playing bridge, and the sentence x + y = 4 is true when there are four people in total. Formally, the possible models are just all possible assignments of real numbers to the variables x and y.

Satisfaction: Satisfaction is a relationship between a model and a sentence, indicating that the model makes the sentence true. If a sentence is true in a particular model, we say that the model satisfies the sentence.
In the given context, if a sentence α is true in a model m, it is said that m satisfies α. Alternatively, m is considered a model of α. The notation M(α) is used to represent the set of all models that satisfy the sentence α.

Model or Possible World in logic

The semantics defines the truth of each sentence with respect to each possible world. In standard logics, every sentence must be either **true or false** in each possible world—there is no "in between." When we need to be precise, we use the term model in place of "**possible world**." If a sentence **α** is true in model m, we say that m satisfies α or sometimes m is a model of **α**. We use the notation M(α) to mean the set of all models of α.

Logical entailment between sentences: A sentence follows logically from another sentence. In mathematical notation, we write α |= β. Entailment in logic refers to a relationship between propositions where the truth of one proposition necessarily guarantees the truth of another. If proposition A entails proposition B, it means that whenever A is true, B must also be true. In formal logic, this relationship is often represented as A |= B.

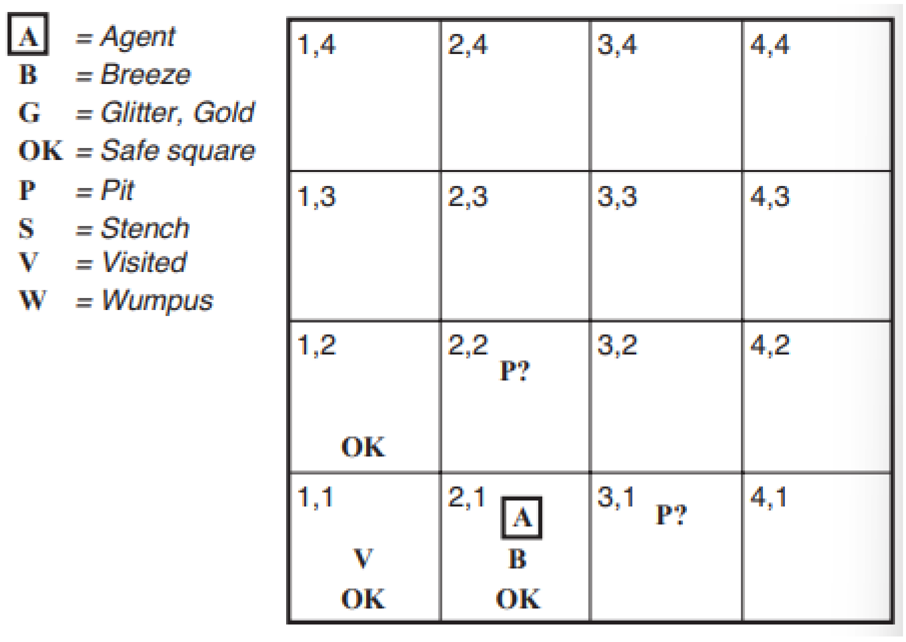
Formal Definition: The formal definition of entailment is this:

α |= β if and only if, in every model in which **α is true, β** is also true.
Using the notation just introduced, we can write
α |= β if and only if M(α) ⊆ M(β) .

E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"

Example: The relation of entailment is familiar from **arithmetic**; the idea that the **sentence x = 0** entails the sentence **xy = 0**. Obviously, in any model where **x is zero**, it is the case that **xy is zero** (**regardless of the value of y**)

Wumpus World Example: Consider the situation in Figure below: the agent has detected nothing in [1,1] and a breeze in [2,1].



These precepts, combined with the agent's knowledge of the rules of the wumpus world, constitute the KB. The agent is interested (among other things) in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits. Each of the three squares might or might not contain a pit, so (for the purposes of this example) there are 2^ 3 =8 possible models. These eight models are shown in Figure 7.5 below:

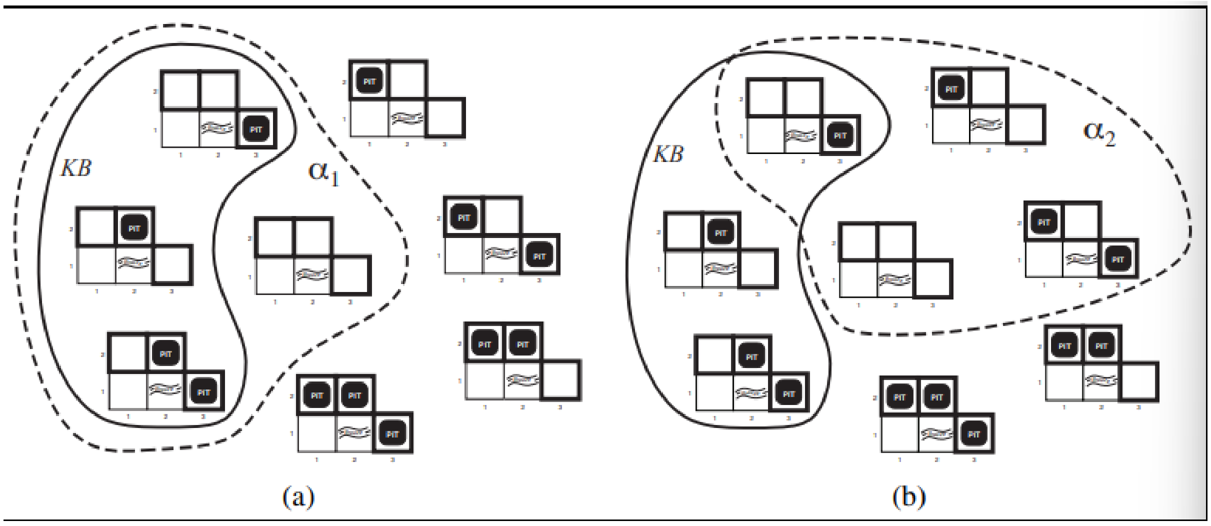


Fig7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α1 (no pit in [1,2]). (b) Dotted line shows models of α2 (no pit in [2,2]).

The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences. The KB is false in models that contradict what the agent knows— for example, the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1]. There are in fact just three models in which the KB is true, and these are shown surrounded by a solid line in Figure 7.5. Now let us consider two possible conclusions:
α1 = “There is no pit in [1,2].”
α2 = “There is no pit in [2,2].”
We have surrounded the models of α1 and α2 with dotted lines in Figures 7.5(a) and 7.5(b), respectively. By inspection, we see the following: in every model in which KB is true, α1 is also true.
Hence, KB |= α1: there is no pit in [1,2]. We can also see that in some models in which KB is true, α2 is false. Hence,

$KB \not\models \alpha_2$:

the agent cannot conclude that there is no pit in [2,2]. (Nor can it conclude that there is a pit in [2,2].)

Logical Inference and Model Checking:
Logical inference is the process of deriving new sentences (conclusions) from existing knowledge or premises. Model checking is an example of a logical inference algorithm where all possible models are enumerated to check if a conclusion holds in all models where the premises are true.

Example: In the wumpus-world example, logical inference is used to determine conclusions about the presence of pits in adjacent squares based on percepts and knowledge.

Model Checking:
Model checking is an inference algorithm that checks if a conclusion holds in all models where the premises are true. In the Wumpus-world example, model checking is applied to determine if certain conclusions (e.g., "There is no pit in [1,2]") hold in all possible models consistent with the agent's knowledge (KB). The inference algorithm illustrated in Figure 7.5 is called model checking, because it enumerates all possible models to check that α is true in all models in which KB is true, that is, that $M(KB) \subseteq M(\alpha)$.

NOTE: To comprehend the concepts of entailment and inference, consider envisioning the set of all consequences derived from a knowledge base (KB) as a haystack, and α as a needle within it. Entailment corresponds to the presence of the needle in the haystack, while inference is akin to the act of discovering it. This distinction is formalized through notation: if an inference algorithm denoted by 'i' can deduce α from KB, we express it as:

$$KB \vdash_i \alpha$$

This notation is read as "α is derived from KB by i" or "i derives α from KB."

Sound or Truth Preserving:

Definition: An inference algorithm is sound or truth preserving if it only derives sentences that are actually entailed by the premises.
Importance: A sound inference procedure ensures that conclusions derived from premises are always true in the real world.
Example: Model checking is considered a sound procedure, as it derives conclusions that hold in all models where the premises are true.

Completeness:

Definition: An inference algorithm is complete if it can derive any sentence that is entailed by the premises.

Importance: Completeness ensures that the inference procedure covers all possible entailed sentences.
Consideration: Completeness becomes crucial for knowledge bases with infinite consequences.
Example: For finite knowledge bases, a systematic examination can decide whether a sentence is entailed, ensuring completeness. However, in infinite knowledge bases, completeness is still achievable with suitable inference procedures.

Correspondence between world and representation

We have outlined a reasoning process whose conclusions are ensured to be accurate in any conceivable scenario where the initial premises hold true. Specifically, if the knowledge base (KB) accurately reflects the real-world state, then any statement α derived from KB through a sound inference procedure is also valid in the real world. Thus, although the inference process operates on "syntax" — involving internal physical configurations like bits in registers or patterns of electrical signals in brains — the process mirrors the real-world dynamics. It demonstrates how certain aspects of the real world are affirmed due to the presence of other aspects in the real world. This relationship between the world and its representation is depicted in Figure 7.6.

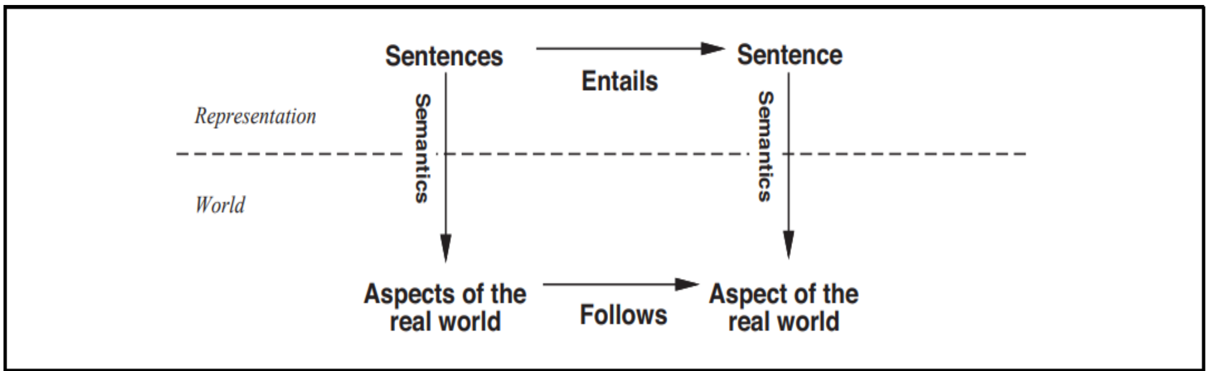


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

Grounding:

Definition: Grounding refers to the connection between logical reasoning processes and the real environment in which an agent exists.
Explanation: It addresses how we establish that the knowledge base (KB) is true in the real world.
Example: In the Wumpus-world example, the agent's sensors create a connection by producing suitable sentences based on perceptual information. The truth of percept sentences is defined by the sensing and sentence construction processes. General rules, derived through learning, contribute to the knowledge base, although learning is fallible.

Overall, the discussion highlights the key concepts of entailment, logical inference, model checking, soundness, completeness, and grounding within the context of reasoning and knowledge representation.

3.2.d Propositional Logic

- Propositional logic, also known as **sentential logic or propositional calculus**, is a branch of formal logic that deals with the logical relationships between **propositions** (statements or sentences) without considering the internal structure of the propositions.
- In propositional logic, **propositions are considered as atomic units**, and logical operations are applied to these propositions to form more complex statements.

Some key elements and concepts in propositional logic:

1. **Propositions:** These are statements that can be either true or false. Propositions are represented by variables, typically denoted by letters (p, q, r, etc.).
2. **Logical Connectives:** These are symbols that combine propositions to form more complex statements. The main logical connectives in propositional logic include:
    1. **Conjunction (∧):** Represents "and." The compound proposition "p ∧ q" is true only when both p and q are true.
    2. **Disjunction (∨):** Represents "or." The compound proposition "p ∨ q" is true when at least one of p or q is true.
    3. **Negation (¬):** Represents "not." The compound proposition "¬p" is true when p is false.
    4. **Implication (→):** Represents "if...then." The compound proposition "p → q" is false only when p is true and q is false.
    5. **Biconditional (↔):** Represents "if and only if." The compound proposition "p ↔ q" is true when p and q have the same truth value.
3. **Truth Tables:** Truth tables are used to systematically list all possible truth values for a compound proposition based on the truth values of its constituent propositions. Truth tables help determine the truth conditions for complex statements.
4. **Logical Equivalence:** Two propositions are logically equivalent if they have the same truth values for all possible combinations of truth values of their component propositions.

Syntax in Propositional Logic

In propositional logic, the syntax dictates the permissible sentences.

Atomic Sentences: Atomic sentences are comprised of a single proposition symbol, each symbol representing a proposition that can be either true or false. Symbol names, starting with an uppercase letter and potentially containing other letters or subscripts (e.g., P, Q, R, $W_{1,3}$, North), are arbitrary but often chosen for mnemonic value.
For instance, $W_{1,3}$ might stand for the proposition that the wumpus is in [1,3]. Notably, symbols like W, 1, and 3 are not meaningful constituents of the atomic symbol. Two proposition symbols, "**True**" (always true) and "**False**" (always false), have fixed meanings.

Complex Sentences: Construction of complex sentences involves simpler ones through the use of parentheses and logical connectives. **Five common** logical connectives include:

Negation ¬ (not): A sentence like $\neg W_{1,3}$ is termed the negation of $W_{1,3}$. A literal is either a positive literal (atomic sentence) or a negated atomic sentence (negative literal).

Conjunction ∧ (and): A sentence with ∧ as its main connective, e.g., $W_{1,3} \land P_{3,1}$, is a conjunction; its parts are the conjuncts.

Disjunction ∨ (or): A sentence using ∨, like $(W_{1,3} \land P_{3,1}) \lor W_{2,2}$, is a disjunction of the disjuncts $(W_{1,3} \land P_{3,1})$ and $W_{2,2}$.

Implication ⇒ (implies): A sentence such as $(W_{1,3} \land P_{3,1}) \Rightarrow \neg W_{2,2}$ is termed an implication (or conditional). Its premise or antecedent is $(W_{1,3} \land P_{3,1})$, and its conclusion or consequent is $\neg W_{2,2}$. Implications are also known as rules or if–then statements. The implication symbol may also be represented as ⊃ or → in other texts.

Biconditional ⇔ (if and only if): The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a biconditional. Some texts represent this as ≡.

This syntax allows the formulation of propositions and their logical relationships using a set of well-defined connectives.
Figure below gives a formal grammar, BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
 & & \\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Semantics in Propositional Logic

Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- In propositional logic, a model simply **fixes the truth value—true or false—for every proposition symbol**.
- For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is
- $m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}$

Rule for Atomic Sentences

- **True** is true in every model and **False** is false in every model.

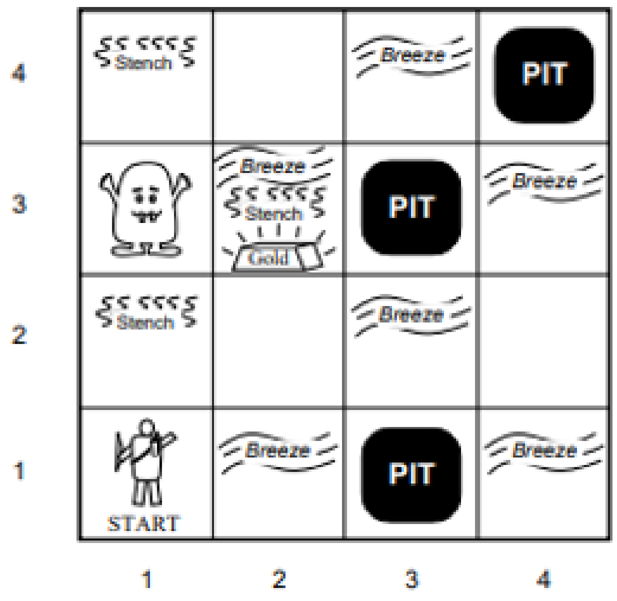
<u>For complex sentences</u>**, we have five rules, which hold for any subsentences P and Q in any model m (here "iff" means "if and only if"):**

- **¬P** is true iff P is false in m.
- **P ∧ Q** is true iff both **P and Q** are true in m.
- **P ∨ Q** is true iff either **P or Q** is true in m.
- **P ⇒ Q** is true unless **P is true and Q** is false in m.
- **P ⇔ Q** is true iff P and Q are both true or both false in m.

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \lor Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \lor Q$ column to see the result: *true*.

A simple knowledge base : Wumpus World



Now that we have defined the semantics for propositional logic, we can construct a knowledge base for the Wumpus world as follows :

Symbols for each [x, y] location:

- Px,y is true if there is a pit in [x, y].
- Wx,y is true if there is a wumpus in [x, y], dead or alive.
- Bx,y is true if the agent perceives a breeze in [x, y].
- Sx,y is true if the agent perceives a stench in [x, y].

Sentences

- There is no pit in [1,1]:
  $\mathbf{R_1 : \neg P_{1,1}}$.
- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:
  $\mathbf{R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})}$.
  $\mathbf{R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})}$.
- The preceding sentences are true in all Wumpus worlds:
  $\mathbf{R_4 : \neg B_{1,1}}$
  $\mathbf{R_5 : B_{2,1}}$.

A simple inference Procedure

Our goal now is to decide whether KB |= α for some sentence α. For example, is $\neg P_{1,2}$ entailed by our KB? Our first algorithm for inference is a **model-checking approach** that is a direct implementation of the definition of entailment: enumerate the models, and check that α is true in every model in which KB is true. Models are assignments of true or false to every proposition symbol. Returning to our wumpus-world example, the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$. With seven symbols, there are 2^7 = 128 possible models; in three of these, KB is true (Figure 7.9).

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Figure 7.9 A truth table constructed for the knowledge base given in the text. *KB* is true if R_1 through R_5 are true, which occurs in just 3 of the 128 rows (the ones underlined in the right-hand column). In all 3 rows, $P_{1,2}$ is false, so there is no pit in [1,2]. On the other hand, there might (or might not) be a pit in [2,2].

In those three models, $\neg P_{1,2}$ is true, hence there is no pit in [1,2]. On the other hand, $P_{2,2}$ is true in two of the three models and false in one, so we cannot yet tell whether there is a pit in [2,2]. Figure 7.9 reproduces in a more precise form the reasoning illustrated in Figure 7.5.

A general algorithm for deciding entailment in propositional logic is shown in Figure 7.10. The algorithm is sound because it implements directly the definition of entailment, and complete because it works for any KB and α and always terminates—there are only finitely many models to examine. If KB and α contain n symbols in all, then there are 2^n models. Thus, the time complexity of the algorithm is $O(2^n)$. (The space complexity is only O(n) because the enumeration is depth-first). Every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input.

```
function TT-ENTAILS?($KB, \alpha$) returns $true$ or $false$
  inputs: $KB$, the knowledge base, a sentence in propositional logic
          $\alpha$, the query, a sentence in propositional logic

  $symbols \leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
  return TT-CHECK-ALL($KB, \alpha, symbols, \{\,\}$)
--------------------------------------------------------------------------------
function TT-CHECK-ALL($KB, \alpha, symbols, model$) returns $true$ or $false$
  if EMPTY?($symbols$) then
     if PL-TRUE?($KB, model$) then return PL-TRUE?($\alpha, model$)
     else return $true$ // when KB is false, always return true
  else do
     $P \leftarrow$ FIRST($symbols$)
     $rest \leftarrow$ REST($symbols$)
     return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
            and
            TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword "**and**" is used here as a logical operation on its two arguments, returning *true* or *false*.

## End of the Module 3 ##