mean end analysis algo

nlp and exprest system

CS-208: Artificial Intelligence Lecture-01

What is Intelligence?

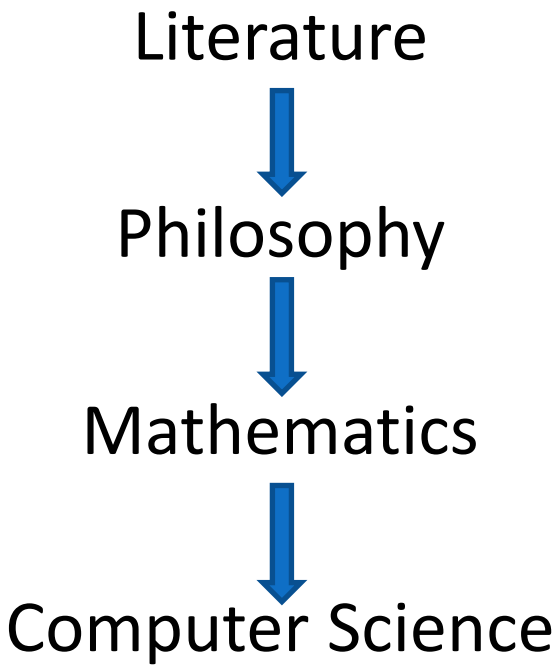
The study of Artificial Intelligence(AI) is about getting the computers to do things that seems to be intelligent. The purpose is that more intelligent computers can be more helpful to us.

What do you mean by the term intelligence?

- ✓ Is it the ability to reason?
- ✓ Is it the ability to acquire knowledge and apply knowledge?
- ✓ Is it the ability to perceive and manipulate the things in the physical world?

Obviously all of these are part of what is intelligence. It is a vague word that eludes from the abstract definition and it appears to be an amalgam of various information representation and information processing talents

Early History of Artificial Intelligence(AI)



AI in form of Literature

Though the intellectual and technological tools necessary to attempt AI (such mathematical logic, digital computer etc.) came into being during the last few decades, the idea of AI prevailed as early as 725 BC, A well known Greek myth has Aphrodite with life: a statue constructed by the Cypriot king Pygmalion. In the medieval legend the *Rabbi of Prague* creates the *first golem* or robot. The most recent AI literature is Mary Shelley's Frankenstein.

AI in form of Philosophy

AI jumped from literature to philosophy with the work of empiricists. They believed that human cognition is divided into two parts sensation(external stimuli) and conception (internal mental event) and given finite set of rules that associate conjunction of sensation with conception. This was the first step in the separation AI from mythology and mysticism.

AI in the form Mathematics

During twentieth century mathematician carried forward the work of empiricist and proposed a general method in which all truth would be reduced to a kind of computation. George Boole was among the first to ponder AI

AI has Become the Part of Computer Science

With the advent of digital computer, AI has become the branch of Computer Science. This term Artificial Intelligence was coined by John McCarthy in the year 1956. AI has matured over last 70 years with the advancement in digital technology.

Ultimate Goal AI

There is some formalism through which we can view the behaviour of the human brain as a form of computation.

Once we discover the precise nature of this computation, it can be implemented in the computer, thereby achieving the ultimate goal AI **(True Artificial Intelligence)**

HISTORY OF AI DURING THE DIGITAL ERA

The idea of Intelligent robots and artificial beings first appeared in the ancient Greek myths of Antiquity. Aristotle's development of the syllogism and it's use of deductive reasoning was a key moment in mankind's quest to understand its own intelligence. The roots of the artificial intelligence are long and deep. The following is a quick glance at the most important recent events in AI.

1943

- Warren McCullough and Walter Pitts publish" A Logical Calculus of Ideas Immanent in Nervous Activity." The paper proposed the first mathematic model for building a neural network.

1949

- In his book *The Organization of Behavior: A Neuropsychological Theory,* Donald Hebb proposes the theory that neural pathways are created from experiences and that connections between neurons become stronger the more frequently they're used. Hebbian learning continues to be an important model in AI.

1950

- Alan Turing publishes "Computing Machinery and Intelligence, proposing what is now known as the Turing Test, a method for determining if a machine is intelligent.
- Harvard undergraduates Marvin Minsky and Dean Edmonds build SNARC, the first neural network computer.
- Claude Shannon publishes the paper "Programming a Computer for Playing Chess."
- Isaac Asimov publishes the "Three Laws of Robotics.“

1952

- Arthur Samuel develops a self-learning program to play checkers.

1954

- The Georgetown-IBM machine translation experiment automatically translates 60 carefully selected Russian sentences into English.

1956

- The phrase artificial intelligence is coined at the "Dartmouth Summer Research Project on Artificial Intelligence." Led by John McCarthy, the conference, which defined the scope and goals of AI, is widely considered to be the birth of artificial intelligence as we know it today.
- Allen Newell and Herbert Simon demonstrate Logic Theorist (LT), the first reasoning program.

1958

- John McCarthy develops the AI programming language Lisp and publishes the paper "Programs with Common Sense." The paper proposed the hypothetical Advice Taker, a complete AI system with the ability to learn from experience as effectively as humans do.

1959

- Allen Newell, Herbert Simon and J.C. Shaw develop the General Problem Solver (GPS), a program designed to imitate human problem-solving.
- Herbert Gelernter develops the Geometry Theorem Prover program.
- Arthur Samuel coins the term machine learning while at IBM.
- John McCarthy and Marvin Minsky found the MIT Artificial Intelligence Project.

1963

- John McCarthy starts the AI Lab at Stanford.

1966

- The Automatic Language Processing Advisory Committee (ALPAC) report by the U.S. government details the lack of progress in machine translations research, a major Cold War initiative with the promise of automatic and instantaneous translation of Russian. The ALPAC report leads to the cancellation of all government-funded MT projects.

1969

- The first successful expert systems are developed in DENDRAL, a XX program, and MYCIN, designed to diagnose blood infections, are created at Stanford.

1972

- The logic programming language PROLOG is created.

1973

- The "Lighthill Report," detailing the disappointments in AI research, is released by the British government and leads to severe cuts in funding for artificial intelligence projects.

1974-80

- Frustration with the progress of AI development leads to major DARPA cutbacks in academic grants. Combined with the earlier ALPAC report and the previous year's "Lighthill Report," artificial intelligence funding dries up and research stalls. This period is known as the "First AI Winter."

1980

Digital Equipment Corporations develops R1 (also known as XCON), the first successful commercial expert system. Designed to configure orders for new computer systems, R1 kicks off an investment boom in expert systems that will last for much of the decade, effectively ending the first "AI Winter."

1982

- Japan's Ministry of International Trade and Industry launches the ambitious Fifth Generation Computer Systems project. The goal of FGCS is to develop supercomputer-like performance and a platform for AI development.

1983

- In response to Japan's FGCS, the U.S. government launches the Strategic Computing Initiative to provide DARPA funded research in advanced computing and artificial intelligence.

1985

- Companies are spending more than a billion dollars a year on expert systems and an entire industry known as the Lisp machine market springs up to support them. Companies like Symbolics and Lisp Machines Inc. build specialized computers to run on the AI programming language Lisp.

1987-93

- As computing technology improved, cheaper alternatives emerged and the Lisp machine market collapsed in 1987, ushering in the "Second AI Winter." During this period, expert systems proved too expensive to maintain and update, eventually falling out of favor.
- Japan terminates the FGCS project in 1992, citing failure in meeting the ambitious goals outlined a decade earlier.
- DARPA ends the Strategic Computing Initiative in 1993 after spending nearly $1 billion and falling far short of expectations.

1991

- U.S. forces deploy DART, an automated logistics planning and scheduling tool, during the Gulf War.

1997

- IBM's Deep Blue beats world chess champion Gary Kasparov

2005

- STANLEY, a self-driving car, wins the DARPA Grand Challenge.
- The U.S. military begins investing in autonomous robots like Boston Dynamic's "Big Dog" and iRobot's "PackBot."

2008

- Google makes breakthroughs in speech recognition and introduces the feature in its iPhone app.

2011

- IBM's Watson trounces the competition on *Jeopardy!.*
- **2012**
- Andrew Ng, founder of the Google Brain Deep Learning project, feeds a neural network using deep learning algorithms 10 million YouTube videos as a training set. The neural network learned to recognize a cat without being told what a cat is, ushering in breakthrough era for neural networks and deep learning funding.

2014

- Google makes first self-driving car to pass a state driving test.

2016

- Google Deep Mind's Alpha Go defeats world champion Go player Lee Sedol. The complexity of the ancient Chinese game was seen as a major hurdle to clear in AI.

Definition of AI (in Four Different Categories)

Thinking Humanly The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning (Bellman)	**Thinking Rationally** The study of the computations that make it possible to perceive, reason, and act (Winston)
Acting Humanly Study of how to make computers do things at which, at the moment, people are better (Rich and Knight)	**Acting Rationally** Computational Intelligence is the study of the design of intelligent agents." (Poole et al)

Human-centered Approach *: It involves observations and hypotheses about human behaviour* .
Rationalist Approach: *It involves a combination of mathematics and engineering*.

CS-208:Artificial Intelligence
L-02
Solving AI Problems

Distinguishing AI Problems

The problems for which can not be solved through precise models, methods and algorithms as well as involve uncertainty are classified as AI problems.

Two major approaches to solve AI problems:

I. One is using Hard-Computing Techniques which are classical AI problem solving methods based on symbolism and search with an aid of symbolic logic.

II. The other uses soft-computing techniques- which consist of five major components namely Fuzzy Logic, Neural Networks, Probabilistic Reasoning, Genetic Algorithm and Chaos Theory.

In this course we focus on the first approach for solving AI problems

How to build a system that solve AI Problems

This involves the following four important tasks:

1. **Define the Problem Precisely**: This definition must include the precise specification of what the initial situation will be as well as what final situations constitute the acceptable solution to the problem.
2. **Analyze the Problem** : A few very important features can have immense impact on the appropriateness of the various possible techniques for solving the problem.
3. I**solate and Represent the task knowledge** that is necessary to solve the problem
4. **Choose the best problem solving technique(s)** and apply it(them) to the particular problem

An Example of Build a Program That Could Play Chess

Before start writing a program that can play chess, we need to specify the following:

1. ***The starting position of the chess board***
2. ***The rules that defines the legal moves***
3. ***The board positions that represent a win for one side or other***

Goal: The program not only play a legal game of chess, but also winning the game if possible

Formal Definition

(Representing Chess Playing Problem from programming point of view)

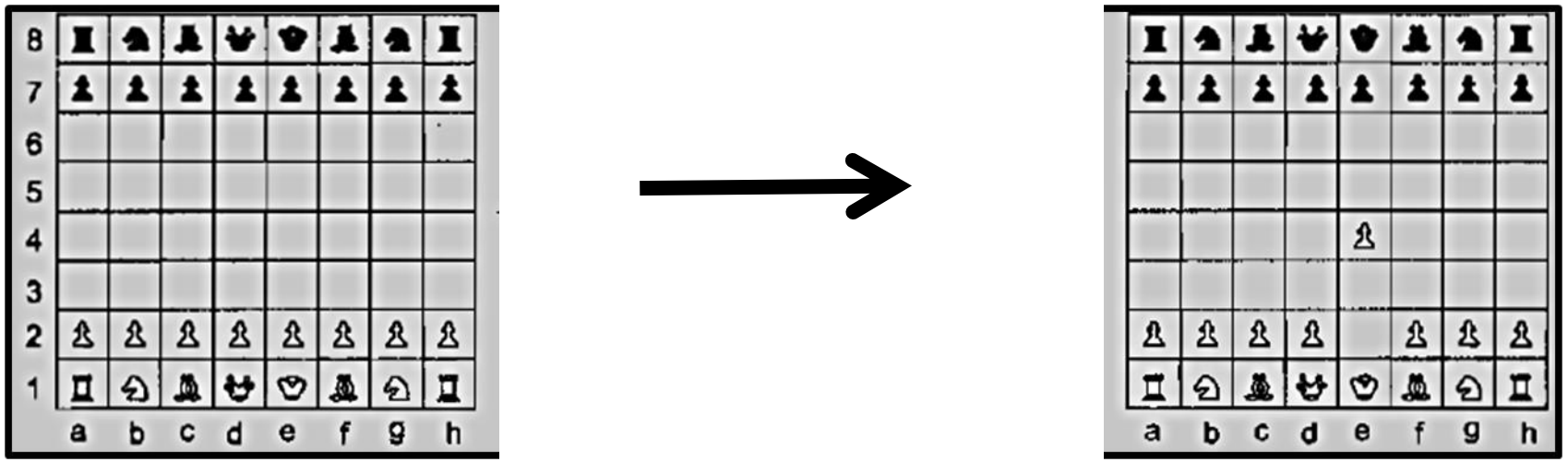
1. **The Starting Position:** it can be described by 8x8 array where each position contains a symbol at appropriate place same as in the official chess opening board position.
2. **The Goal Position**: Any board position that the opponent can not make any legal move or his/her king is under attack.
3. **Legal moves:** These provide the ways of getting from the starting state to goal state

Describes the starting position and the goal position can be done very easily, but the real difficulty lies in how to describe the rules for legal moves.

One Way of Describing the Legal Moves

This left side serves as a pattern to be matched against the current board position

This right sides reflects the change in the board after the application of the legal moves.



Difficulties in This Legal Move Representation

If we write rules in this manner, there will be 10^{120} rules equal to the number of possible board positions. This creates the following implementation difficulties

I. *Nobody can supply the complete set of rules*

- It takes too much time (more than life span of our earth)

- It can not done without mistake or repetition

II. *No program can handle this huge number of rules*

- It is not possible for searching applicable rules for the current board position

- It is not possible to Store 10^{120} rules in computer memory

What is State Space?

Background Knowledge

Definition of a State: *A situation in the world described by some facts is a state.*

A state can be *instantaneous* or it can *last for some time (non-instantaneous)*.

The state changes(transitions) can be either *discrete* or *continuous*.

In physics: *A physical state is a set of measurable physical parameters that is fully describing some part of the universe*.

In computer science: *A process state is a set of values that are currently stored in registers, buffers and stack. So process state is a snap shot of a process in time.*

In Artificial Intelligence: *A problem state* includes everything important about a situation in one bundle (that is everything necessary to reason about it). It usually means a list of facts.. There may be lot of facts and we often include only those facts that are necessary for reason about in solving the AI problems.

Two tricks to simplify the state description:

1. *Record only facts relevant to the very specific problem.*
2. *Compressing the facts.*

State Space Search as Solving the AI Problem

- We can view the solving of an AI problem as *moving around the state space that begins with a starting state and ends with one of the goal state*.
- The set of all possible states of a problem constitute *problem universe*.
- Now the problem of playing chess can be defined as a problem of moving around in a state space, where each state corresponds to a legal position of the chess board.
- chess can be played by starting at an starting/initial state, using a set of legal rules to move from one state to another and attempt to end up in one of the goal/final state.

Structure State Space Representation corresponds to the Structure of Problem Solving

- State space representation allows the **formal** definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operation

- State space representation also permits us to define the process of solving a particular problem as a combination of known techniques and the search.

An Illustrative Example

Water Jug Problem: There are two water jugs, one is 4-liter capacity and the other is 3-liter capacity. Neither has any measuring marks on it. There is a water pump that can be used to fill the jugs with water. How can we get 2 liters of water in the 4-liter jug?

- How to make a good representation of a state for this problem?
- How to describe a set of state transition rules?

Representation of the (Water Jug) Problem State

The problem state can be represented as an ordered pairs of integers **(*a*, *b*)** that represent the quantities of water in 4-liter and 3-liter jugs respectively.

Where,

a represents the quantity of water in 4-liter jug and takes one of the values 0, 1, 2, 3, 4.

b represents the quantity of water in 3-liter jug and takes one of the values 0, 1, 2, 3.

Starting(or Initial) State can be represented by (0,0).

Goal(or End) State is represented by (2,*x*).

State Transition Rules or Production Rules

Before describing the rules/operators, we need to make few assumptions which are not explicitly mention in the informal problem statement. The unstated assumptions for the water jug problem are:

a. *We can fill water in a jug using the pump.*

b. *We can pour water out of a jug onto the ground.*

c. *We can transfer water from one jug to another.*

d. *There no other measuring device available.*

- ✓ Left side of the rule represent the current state.
- ✓ Right side of the rule represent the state that reflect the changes in the current state by application of the rule.
- ✓ Group of similar branches into categories called operators or production rules. It is important to note that the pre-condition of the rule has to be satisfied by the current state.

Production Rules for Water Jug Problem

Filling the water jug up to its full capacity

1. (a, b) → (4, b) Fill 4-liter jug

   If a<4 (Pre-Condition)

2. (a, b) → (a, 3) Fill 3-liter jug

   If b<3

Emptying the water jug completely

3. (a, b) → (0, b) Empty the 4-liter jug on the ground

   If a>0

4. (a, b) → (a, 0) Empty the 3-liter jug on the ground

   If b>0

Transferring the water from one jug to another

5. (a, b) $\rightarrow$ (4, b-(4-a)) Pour water from 3 liter jug into 4 liter jug until it becomes full.
   If (a+b) ≥ 4 and b>0

6. (a, b) $\rightarrow$ (a-(3-b), 3) Pour water from 4 liter jug into 3 liter jug until it becomes full.
   If (a+b) ≥ 3 and a>0

7. (a, b) $\rightarrow$ (a+b, 0) Pour all the water from 3 liter jug into 4 liter jug.
   If (a+b) ≤ 4 and b>0

8. (a, b) $\rightarrow$ (0, a+b) Pour all the water from 4 liter jug into 3 liter jug.
   If (a+b) ≤3 and a>0

Pour some quantity of water(d) out of the water jug

9. (a, b) → (a-d, b) Pour some quantity of water(d) out of 4-liter jug
   If a>0

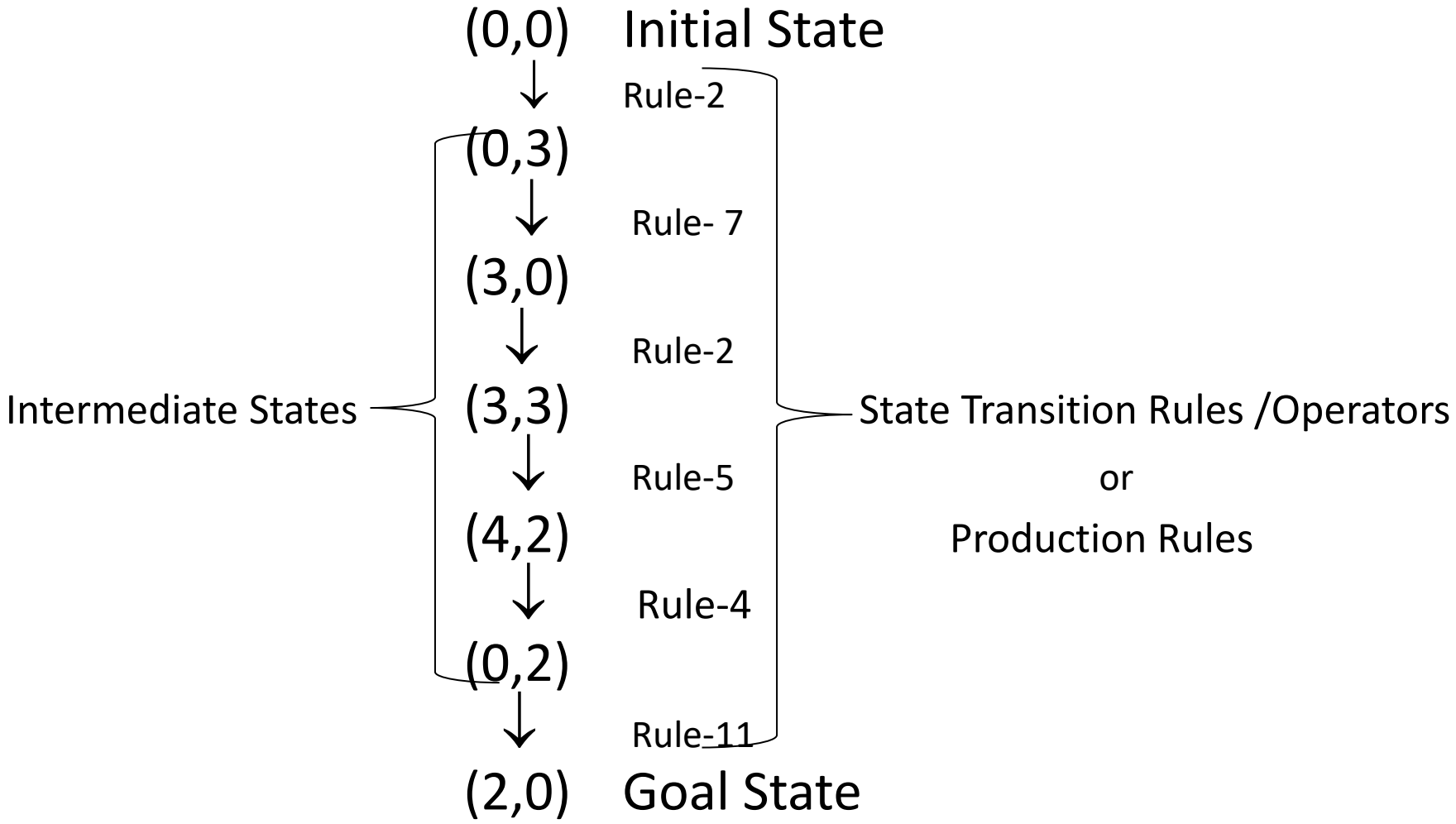
10. (a, b) → (a, b-d) *Pour some quantity of water(d) out of 3-liter jug*
    If b>0

Special Case Rules That Represent the Problem Solving Knowledge

11. (0, 2) → (2, 0) Transfer 2 liter of water from the 3-liter jug into the 4-liter jug

12. (a, 2) → (0, 2) Empty the 4-liter jug

One of the solutions to Water Jug Problem



Translating Informal Description of an AI Problem into Formal Description

1. Define the Problem State Space-the problem state contains al possible configuration of relevant objects
2. Specify one or more states within state space that describe possible situation from which the problem solving process may start. These states are called initial states
3. Specify one or more stats that would be acceptable as a solution to the problem. These states are called goal state.
4. Specify a set of rules that describe the action available
   - What are all the unstated assumptions that are hidden in the informal problem description.
   - How general should the rules be made?
   - How much work required to solve the problem should be pre-computed and represented in the rules.

Production System

A production system consists of

1. **A Set of Rules** *each consisting of left side that determines the applicability of the rule and the right side that describe the operation to be performed, if the rule is applied.*
2. **One or more Knowledge/Databases** *that contain whatever information is appropriate for the particular task.*
3. **Control Strategy** *that specifies the order in which the rules will compared to the databases and way of resolving conflicts that arise several rules match at once*.
4. **A Rule Applier**.

****

- ***Heuristic*** **is a rule of thumb, a piece of information that can provide useful guidance to a problem solver, but which is not guaranteed to be applicable or beneficial in all situations.**
- ***Heuristic Search*** **is a search using heuristic knowledge to focus the exploration of the state space on more promising path.**

****

CS-208: Artificial Intelligence
Lectures-03
AI Problem Characteristics

Seven AI Problem Characteristics

CHARACTERISTIC-I: Is the problem decomposable or not?

Decomposable means that breaking the given problem into smaller(Independent) sub-problems, each of which can be solved by using small collection of specific rules.

Advantage: Large problem can be solved easily using this problem decomposition technique

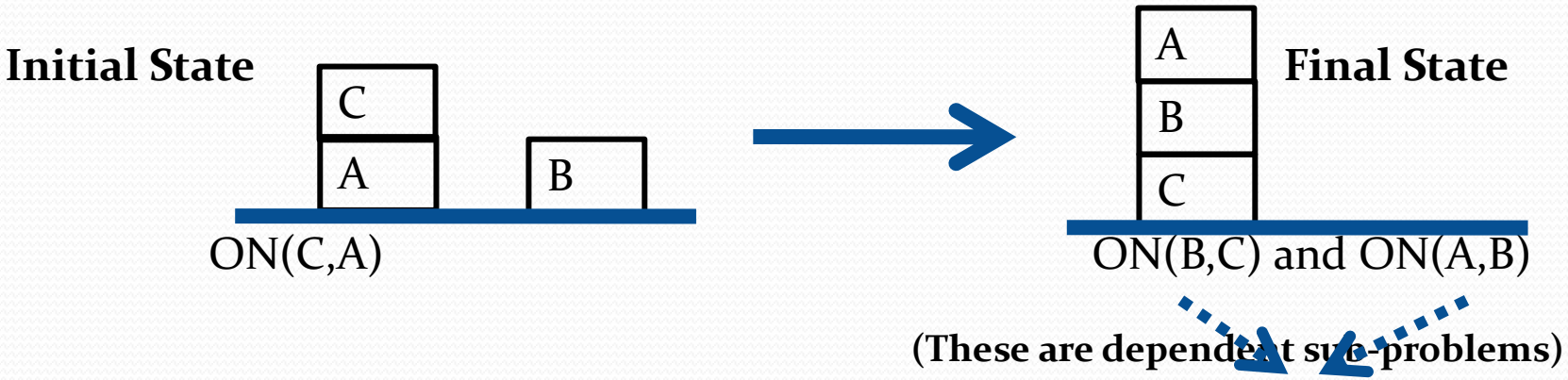
Problem-1: Integration (Example of Decomposable Problem)

$$\int (x^2 + 3x)dx = \int x^2 dx + \int 3x dx + c = \frac{x^3}{3} + \frac{3x^2}{2} + c$$

(These are independent sub-problems)

Conclusion: ***Here the sub-problem are independent and can be solved separately in parallel. So this problem is decomposable***

- **Problem-2 : Block World (Example of Non-decomposable Problem)**



Rules of the Game:

1. CLEAR(*x*) → ON (*x*, Table) *Pick the Block X and place it on the table*

   if block x has nothing on it

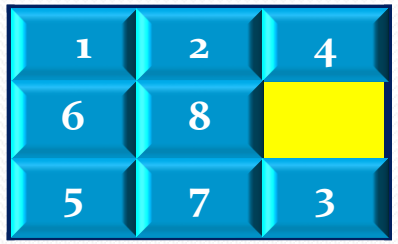
2. CLEAR(*x*) and CLEAR(*y*) → ON(*x*, *y*) *Place Block X on Block Y*

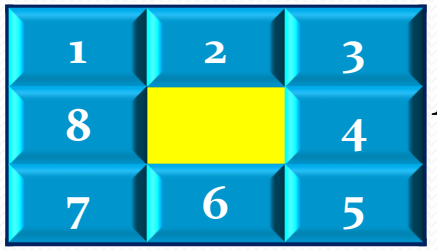
Conclusion: ***Here the sub-problem are dependent and can be solved in sequence. So this problem is not decomposable***

CHARACTERISTIC-II: Can Solution Steps be Ignored or Undone?

Consider the following three types of problems:

1. Mathematical Theorem Proving.
2. The 8-Puzzle Problem: There is square tray in which eight small square tiles are placed. Each tile has distinct number (1 to 8) on it. The ninth small square tile space is uncovered creating a blank space. A tile that is adjacent to the blank space can be slide into the place. A game consist of a starting position and a specified goal position. The Goal is to transform the starting position into goal position by sliding the tiles around'

Initial State 

→ 

Final State

3. Playing Chess .

These problems are belong to three different classes, namely:

1. ***Ignorable Class***: in which solution steps can be ignored. Theorem proving is an example of ignorable class problem. This class of problems can be using a simple control structure that never backtrack(easy to implement).
2. ***Recoverable Class***: in which solution steps can be undone. The 8-puzzle is an example of recoverable class problem. This class of problems can be solved by a slightly more complicated control strategy that does sometimes make mistakes and backtracking will be necessary to recover from the mistakes using stack.
3. ***Irrecoverable Class***: in which solution steps can be neither ignored nor undone. Playing Chess is an example of irrecoverable class problem. This class of problems can be solved by a system that expends a great deal of effort in making each decision (since every decision must be final) and planning process

CHARACTERISTIC-III: Is the Problem Universe Predictable?

In 8-puzzle problem, we know exactly what will happen every time we will make a move. Here, it is possible for us to plan an entire sequence of moves to get into goal state and confident about the resulting state.

Advantage: We can use planning to avoid having to undo an unwanted move during actual moves. Still it be necessary to use backtrack wrong moves during the planning process.

In card games like bridge, this type of planning is not be possible. Because it is not possible to know where are all the cards or what other player will do them during their turns.

Types of Problems

1. Certain Outcome: Example 8-puzzle problem, where the problem universe is predictable. This type of problems can be solved by planning *without feedback from the environment*.

2. Uncertain Outcome: Example card games, where problem universe is not predictable. This type of problems may be solved by using probability and plan revision according to *the feedback from the environment* during the game is required.

CHARACTERISTIC-IV: Is good solution Absolute or Relative?

Problem-1 (Here *Good Solution is Absolute*)

Consider the following facts on a database:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeian died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years (*in the context of human*).
7. It is now 2020 A.D.

Find an asnswer to the question: *Is Marcus alive now?*

8. Marcus is mortal ← from 1 and 4
9. Marcus age is 1980 ←from 3 and 7
10. Marcus is dead now ← from 6, 8 and 9

One way reasoning to get the answer

11. All Pompeian are dead now ←from 7 and 5.
12. Marcus is dead now ←from 11 and 2

Alternative way of reasoning to get the answer

Here there are two reasoning paths that will lead to the answer and we are interested in any one of the answer to question irrespective of which path was being followed.

If we followed one path that successfully lead to the answer then there is no reason to go back and explore other paths that might also lead to the solution

Problem-2: The travelling Salesperson (*Here Good solution is Relative*)

Problem Statement: To find shortest route in visiting each city exactly once from the given list of cities with starting and ending with one of the given city.

Solution: Find distances for all possible routes (*finding the all possible candidate solution*) and select the route with shortest distance (*selecting the best candidate solution*).

Any Path Problem (good solution is absolute)	Best Path Problem (good solution is relative)
Easier to solve	Harder to solve
It can be solved in a reasonable amount of time using heuristics that suggest good path to explore	Much more exhaustive search will be needed.

CHARACTERISTIC-V: Is the Solution a State or Path?

Problem-1: Consider the interpretation for the sentence: "*The bank president ate a dish of pasta salad with the fork*".

Consider following components of this sentence (these components have more than one interpretaions and one of them is appropriate):

- ***Bank*** → *one meaning is Financial Institution and the other is a side(bank) of a river.*
- **Dish** → *one meaning is* a *food (prepared in a particular way) and the other is concave pot.*
- **Pasta salad** → *one meaning is salad containing pasta and the other can be like* ***dog-food*** *does not contain dog.*
- **With the fork** → *one meaning modifies the verb eat and other modification structures like with vegetable and with his/her friends.*

Because of interaction among the interpretations of the constituents of this sentence, some search may be required to find the final interpretation for the sentence. So solution to the problem is the sequence of operations that produce the final state.

Problem-2: Water Jug Problem.

Here it is sufficient to report that the problem is solved and the final state is (2,0).

CHARACTERISTIC-VI: What is the Role of Knowledge?

Knowledge Required for playing Chess(confined to chess game):

1. Rules for making the legal moves
2. Control strategy mechanism
3. Good strategy and tactics to constrain the search and speedup execution.

Knowledge Required for Understanding News Story / matter (very vast) :

Requires lot of previous knowledge about persons, events, situations, etc. to understand /follow the news.

CHARACTERISTIC-VII: Does the Task Requires Interaction with a Person?

Based on this characteristic the AI problems can be divided into two categories namely: *Solitary* and *Conversational*.

Solitary: in which the computer is given a problem description and produces an answer with no intermediate communication and/or no demand for an explanation of the reasoning process.

***Conversational*:** in which there is intermediate communications between a person and the computer, "**either** *to provide addition assistance to the computer* **or** *to provide additional information to the user* **or** *both*".

CS-208:Artificial Intelligence
Lectures-04
Search preliminaries

Search Methods

What is a Search Process? Every Search Process can be viewed as a traversal through a directed graph in which each node represents a problem state and each directed edge represents a relationship between the states it connects.

What is the Aim of a Search Process? The search process must find a path through the directed graph starting at an initial state and ending in one or more final state.

Five Important Issues that arise in all searching Techniques

I. The direction in which to conduct the search

II. Topology of the Search Process

III. How each node in the search process will be represented?

IV. Selecting the Applicable Rules.

V. Using a heuristic functions to guide the search.

I. The direction in which to conduct the search

Forward versus backward reasoning: The main objective of a search process is to discover a path through a problem space from an initial state to a goal state.

There are two direction in which a search process could proceed

i. Forward from the start state or
ii. Backward from the goal state

Depending on the topology of the search space, it may be significantly more efficient to conduct the search in one direction rather than the other.

Factors that influence the direction of the search.

a. Are more possible start states or goal state?

It is desirable to move from smaller set of states to a larger set of states.

b. In which direction the branching factor is higher?

It is better to proceed in the direction of lower branching factor.

c. Will the program be asked to justify its reasoning process to a user?

It is important to proceed in the direction that corresponds more closely with the way that user will think.

Bi-directional Search strategy: simultaneous search forward from the start state and backward from the goal state until two paths meet somewhere in-between.

Disadvantage: The bi-directional search may become ineffective when the two searches may pass each other resulting in more work than it would have taken for one of them.

II. Topology of the Search Process

A simple way to implement any search strategy is to perform as a tree traversal:

Problem Tree: Each node of the tree is expanded by the production rules to generate a set of successor nodes and each of which inturn be expanded. This will continue until a node that represents a solution is found.

Problem Graph: A Tree search procedure can be converted to a graph search procedure by modifying the following actions performed at each time when a node is generated:

a) Examine the set of nodes that have been created so far to check if the new successor node is already exists or not.

b) If it does not, simply add the successor to the graph just as in case of a problem tree

c) If it does already exist then do the following two things

i. Set the node that is being expanded to point the already existing node that is same as the new successor. Simply throw away the new successor.

ii. If the tracking of the best path is need to be recorded then check whether new path is better or worse than the old one

- If the new path is worse then do nothing
- If the new path is better then record the new path as correct path to get the node and propagate the corresponding changes down through its successors as necessary. This will create a cycle (a cycle is a path in which a given node appears more than once)

III. How each node in the search process will be represented?

Here three questions that are need to be answered :

1. How can individual objects and facts be represented?
2. How can the representations of objects and facts be combined to form the representation of a complete problem state?
3. How can the sequence of problem state that arise in a search process be represented efficiently?

The first two are the problem of knowledge representation and the third one is particularly important in the context of a search processes.

IV. Selecting the Applicable Rules

A clever search involves choosing the ones that more likely lead to a solution from among the rules that can be applied at a particular point.

Matching: Extracting rules that can be applied from the collection of rules requires some kind matching between the current state and preconditions of the rules.

Indexing: One way to select applicable rules is to search through all the rules and comparing each one's preconditions to the current problem state. This causes two problems:

i. Scanning all the rules at every step of the search process would be hopelessly inefficient. (solution: *Easy way is to use current state an index into the set of rules and select matching ones immediately*)

ii. If there is no ways immediately obvious whether or not a rule's preconditions are satisfied by the particular problem state.

V. Using a heuristic functions to guide the search

A heuristic was defined as a techniques that aids in the discovery of solution to a problem in a most profitable direction even though there is no guarantee that it will never lead in the wrong direction.

There are two major ways in which a domain specific heuristic information can be incorporated in to a rule based search procedure:

- In the rules themselves.
- As a heuristic function that takes a given problem states as an input and determine how promising they are, by providing a value.

CS-208:Artificial Intelligence
Lectures-05
Uninformed Searching Methods

Depth First Searching Technique(DFS)

Principle: Pursue a single branch of the tree until it yields to a solution or until some *pre-specified depth* has been reached, only then go back and explore other branches.

In this search the other alternatives at the same level are ignored completely as long as there is a hope of reaching the goal node using the original choice.

In some problem the DFS through a tree would slip past the level at which goal node appears and waste incredible energy in exhaustively exploring parts of the lower down. For this type of problems the DFS is the worst possible approach.

DFS Algorithm

Step-1: Form single element Stack consist of the root node-representing starting state.

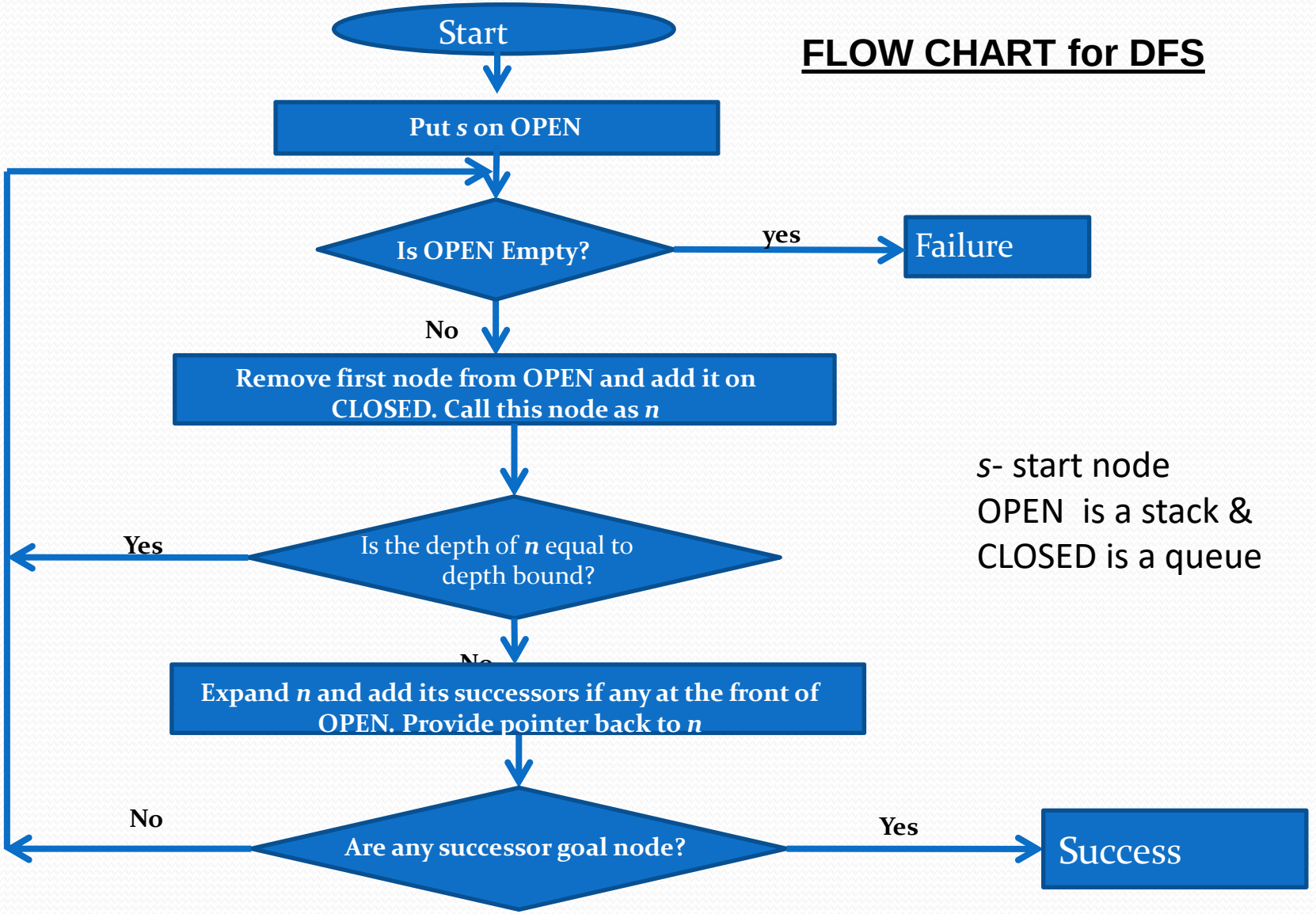
Step-2: Repeat the following until the stack becomes empty or the goal node has been reached:

a. If first element at the front of the queue is a goal node then “do nothing”

b. If the first element is not the goal node, then remove the first element from the queue and add the first element successors if any to the top of the stack

Step-3: If the goal node has been found then announce **SUCCES**S, otherwise **FAILURE**

FLOW CHART for DFS



s- start node
OPEN is a stack &
CLOSED is a queue

Breadth First Searching Technique(BFS)

Principle: All the nodes on one level of the tree are examined before examining any other node in the next level.

Two major merits of BFS:

1. BFS is guaranteed to find the goal node if one exist, provided there are finite number of branches in the tree
2. BFS will also find the goal node with shortest path from the starting node.

Three major demerits of BFS:

1. It requires lot of memory to store the nodes because the number of nodes at each level increases exponentially with level number and all of them must be stored.
2. It may require a lot of work , when the shortest path is too long and the number of nodes increases exponentially with the length of the path.
3. Irrelevant or redundant operators(production rules) will greatly increases the number nodes that must be explored.

BFS Algorithm

Step-1: Form single element queue consist of the root node representing starting state.

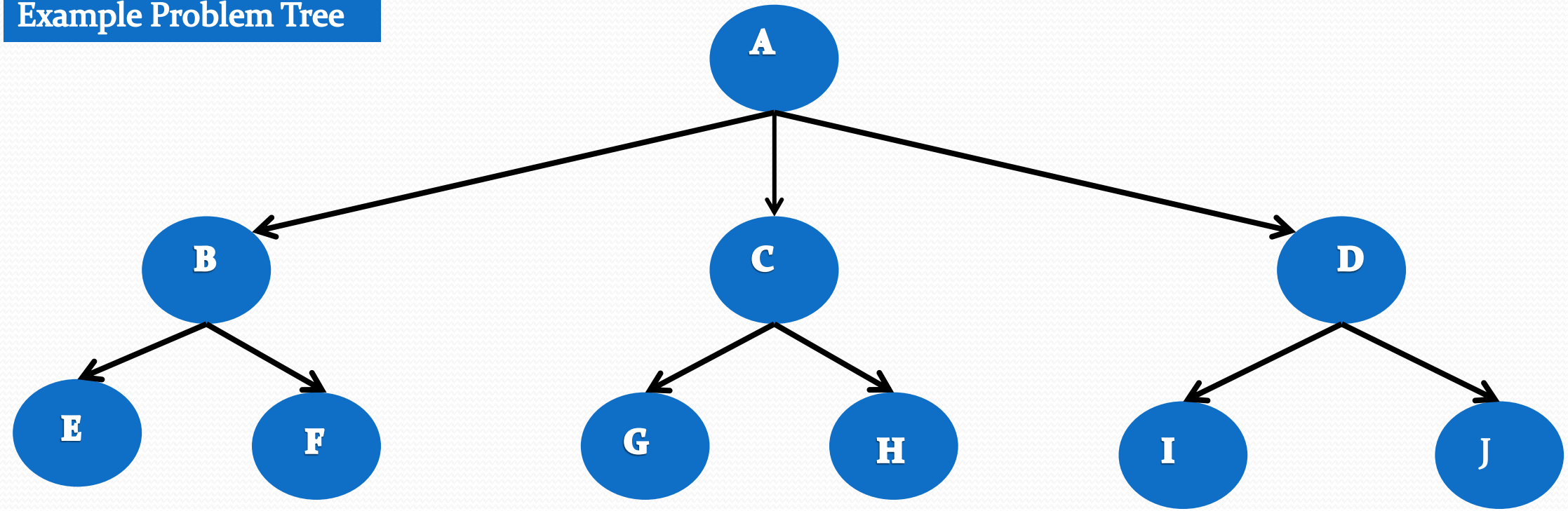
Step-2: Repeat the following until the queue becomes empty or the goal node has been reached:

a. If first element at the front is a goal node then “do nothing”

b. If the first element is not the goal node, then remove the first element from the queue and add the first element successors if any to the rear(end) of the queue

Step-3: If the goal node has been found then announce **SUCCES**S, otherwise **FAILURE**

Example Problem Tree



Path followed by DFS: *A B E F C G H D I J*

Path followed by BFS: *A B C D E F G H I J*

Informed and Uninformed Methods

A searching method is **informed**, if it uses additional information about nodes that have not yet been explored to decide which node to examine next.

If the search method is not informed then it is **uninformed or blind.**

Examples:

Hill Climbing and Best First Search are *Informed search methods.*

Depth First Search and Breadth First Search are *Uninformed search methods*.

CS-208:Artificial Intelligence
Lectures-06
Informed Searching Methods

Hill Climbing Search Technique

This algorithm searches the tree in a depth first manner, at each step choosing paths that appear to be most likely to lead to the goal.

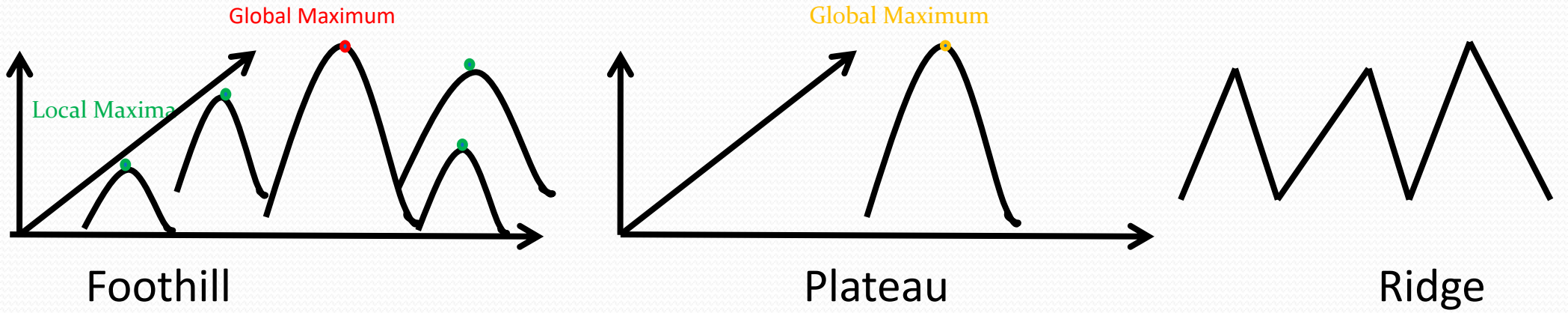
Hill climbing uses heuristic to identify paths efficiently but does not necessarily identify the best path.

Hill climbing will have difficulties in three types of search spaces namely: foothills, plateau and ridges.

Foothills are often called local maxima by mathematician. A local maxima is a part of the search space that appears to be preferable to the parts around it, but which is the fact just a foothill of a large hill. In other words, a local maxima is a state that is better than all its neighbours but not better than some other states farther away. The search space has a single global maximum surrounded by a number of foothills or local maxima. These local maxima draw the hill climbing procedure like magnet. An optimal point is found, but it is local not global and the user is left with false sense of accomplishment.

A Plateau is a region in the search space where all the values are the same. In this case, although there may well be a suitable globl maximum value somewhere nearby, there is no indication from local terrain of which direction to go to find it. Hill climbing search method could well find itself stuck in the plateau with no clear indication of where to go to find a good solution.

A ridge a long thin region of high land with low land on either side. A hill climbing would determine that any point on the top of the ridge was maximum because the hill falls away in all four directions. The correct direction is a very narrow one that leads to the top of the ridge, but identifying this direction using hill climbing could be very tricky.



Hill Climbing Algorithm

Step-1: Form single element Stack consist of the root node representing starting state.

Step-2: Repeat the following until the Stack becomes empty or the goal node has been reached:

a. If first element at the front is a goal node then “do nothing”.

b. If the first element is not the goal node, then remove the first element from the queue and sort the first element successors if any with respect to estimated remaining distance to the goal node before adding them to the top of the Stack.

Step-3: If the goal node has been found then announce **SUCCESS**, otherwise **FAILURE.**

Best First Search Technique

It is the way of combing the advantage of both depth first search and breadth first search into single method.

At each step, the best first search process will select the most promising node that have generated so far.

- ✓ expand the best partial path
- ✓ forward motion is from the best OPEN node so far and no matter where it is in the partially developed tree.
- ✓ An evaluation function is used to provide a means for order nodes on OPEN.

Best First Search Algorithm

Step-1: Form single element queue (called OPEN) consist of the root node representing starting state.

Step-2: Repeat the following until the queue becomes empty or the goal node has been reached:

a. If first element at the front is a goal node then “do nothing”.

b. If the first element is not the goal node, then remove the first element from the queue and add the first element successors if any to the queue. After that sort the entire queue with respect estimated remaining distance to the goal node.

Step-3: If the goal node has been found then announce **SUCCESS**, otherwise **FAILURE.**

CS-208:Artificial Intelligence
Lectures-07
Properties of Search Methods

Properties of Search Methods

Five Important Properties of Search Methods:

a. Complexity: It is useful to describe how efficient that a search method is over time and space.

   Time Complexity of a search method is related to the length of time the search method would take to find a goal node

   Space Complexity of a search method is related to the amount of memory need to use

   BFS time complexity $O(b^d)$ where, b is the Branching factor and d is the depth in which the goal node appears in the problem tree

b. Completeness: A search method is described as being complete, if it is guaranteed to find a goal state if one exist.
   BFS is complete and where as DFS is not complete
c. Optimality: A search method is optimal if it is guaranteed to find the best solution that exist that is it will find a path to a good solution by taking the least number of steps
d. Admissibility: A search method is admissible if it is guaranteed to find the best solution in the quickest possible time.
e. Irrevocability:
   - ❖ A search method that use backtracking are described as tentative for example DFS
   - ❖ A search method that do not use backtracking and which examine just one path are described as irrevocable for example Hill climbing.

CS-208:Artificial Intelligence
Lectures-08
A* Algorithm

Evaluation Function

A more flexible way to use heuristic information is to use some criteria to reorder all the nodes on OPEN at every step. In order to apply such an ordering procedure, we need some measure by which to evaluate the promise of a node. Such measures are called evaluation function.

The value of the evaluation function ***f'*** (***n***)at any node '***n***' is the cost of the path from start node '***s***' to the node '***n***' **PLUS** an estimate of the cost of the minimal cost path from the node '***n***' to the goal node.

$$f'(n) = g'(n) + h'(n)$$

f'(***n***) is an estimate of the cost of the minimal cost path constrained to go though '***n'***.

A* Algorithm

Step-1: Add the start node ***s*** to a list called ***OPEN*** and compute $f'(s)=0+h'(s)$. Set ***CLOSED*** as an empty list.

Step-2: If ***OPEN*** is empty then exit with ***Failure*** else continue.

Step-3: Pick the node on ***OPEN*** with the lowest ***f'*** value. Call it as ***Best_Node*** and remove from ***OPEN*** and add into ***CLOSED.***

Step-4: If the ***Best_Node*** is a goal node exit with solution path which is obtained by tracing back through the pointers else continue.

Step-5: Expand the ***Best_Node*** by generating all of its successors. For each successor do the following:

a) Set a pointer back from the successor to the ***Best_Node.*** (this will be useful for recovering the path, once goal node has been found).

b) Compute ***g'(***successor) = ***g'***(***Best_Node***) + the cost of getting from ***Best_Node*** to successor***.***

c) Check if the successor is same as any node on ***OPEN***. If so, call that node as ***OLD*** and add it to the list of successors to the ***Best_Node***. Find whether it is cheaper to get ***OLD*** via its current parent or to get successor via ***Best_Node.*** IF ***OLD*** is cheaper then do nothing. If successor is cheaper then reset the ***OLD***'s parant link to point the ***Best_Node***. Record the new cheaper path by updating ***g'***(***OLD***) and ***f'***(***OLD***).

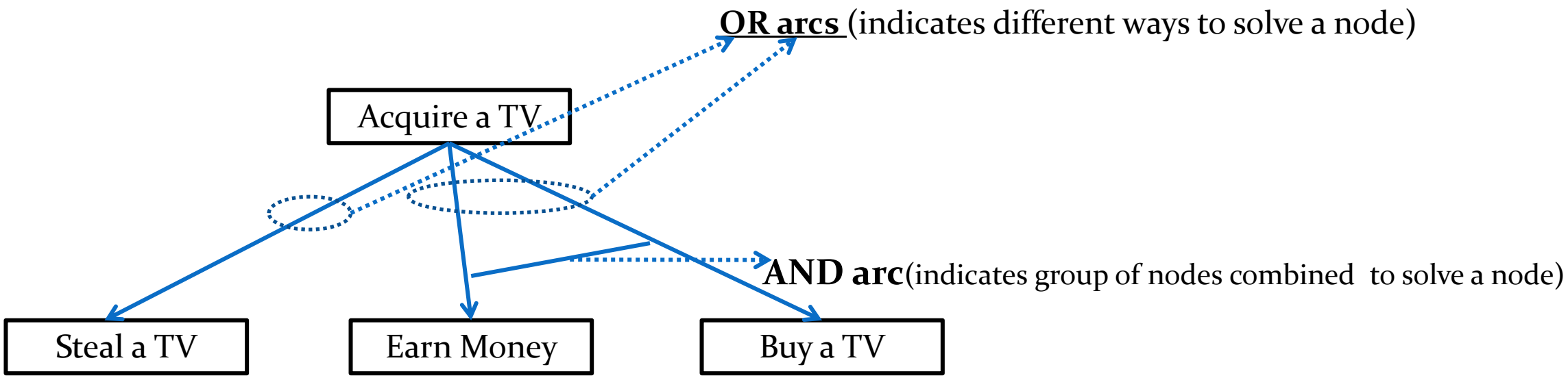
d) If the successor was not on ***OPEN*** then check if it is on ***CLOSED***. If so, call that node as ***OLD*** and add it to the list of successors to the ***Best_Node***. Find whether it is cheaper to get ***OLD*** via its current parent or to get successor via ***Best_Node.*** IF ***OLD*** is cheaper then do nothing. If successor is cheaper then reset the ***OLD***'s parant link to point the ***Best_Node***

-Perform DFS staring at ***OLD*** and update each node's ***g'*** value that is visited. Terminate each branch when either at node with no successor or a node to which equivalent or better path has been found is reached.

e) If the successor was not already on either ***OPEN*** or ***CLOSED***, then add it to ***OPEN***. Compute ***f'***(successor) = ***g'***(successor)+ ***h'***(successor).

f) Go to the Step-2

CS-208:Artificial Intelligence
Lectures-09
Problem Reduction



According to this example,

- one way to acquire TV is to steal a TV
- another way to acquire a TV is to earn sufficient money and buy the TV. Here a line is used to connect the arcs pointing earn money and Buy a TV indicating that both are the sub-problems. This connecting line is called **AND arc**

Problem Reduction

The searching techniques, we have discussed so far DFS, BFS, Hill Climbing, Best First Search and A* algorithm are applicable only for OR-graph.

AND-OR Graphs

The AND-OR Graph (or tree) is useful for representing the solution of problems that can solved by decomposing them into a set of smaller problems, all of which must then be solved.

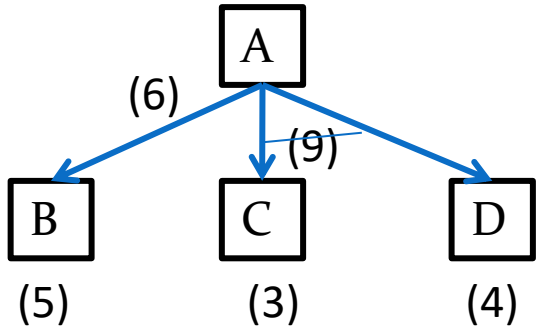
This decomposition, or reduction, generates arcs that are called AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.

Just as in an OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND-graph but rather an AND-OR graph (tree)

Why A* algorithm is not adequate for searching AND-OR Graph?

Explanation through an Illustration:

Consider the following **AND-OR** graph:



Assumptions

1. The number at each node represents ***f*** value. Here we will consider ***f*(n) = *h*(*n*)** only and ignore ***g*(*n*)**.
2. Every operation has a uniform cost for simplicity
   - Each arc with single successor has the cost of 1
   - Each AND arc with multiple successors has a cost of 1 for each of its component arcs it connects.

Here the problem is that the choice of which node to expand not only depends on the *f* value of the node, but also on whether that node is part of the current best path from the initial node.

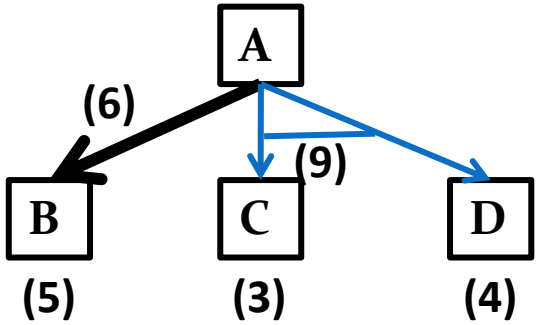


Figure-01

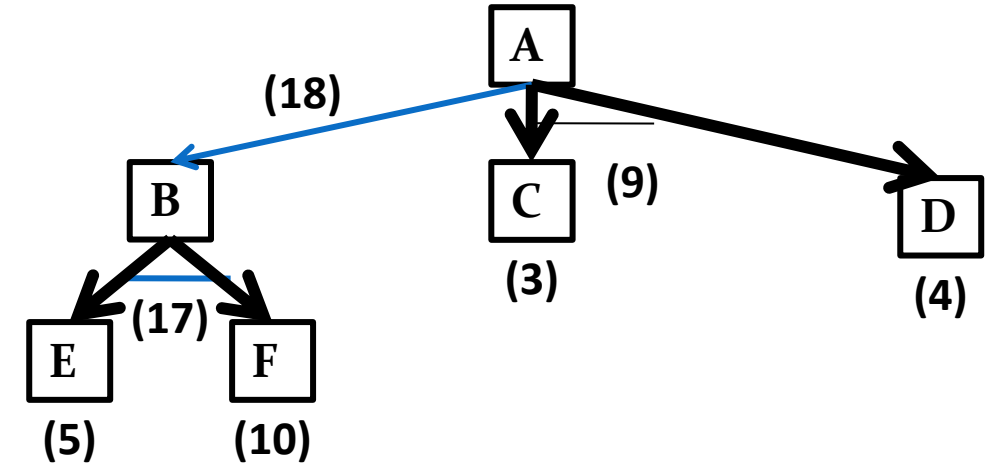


Figure-02

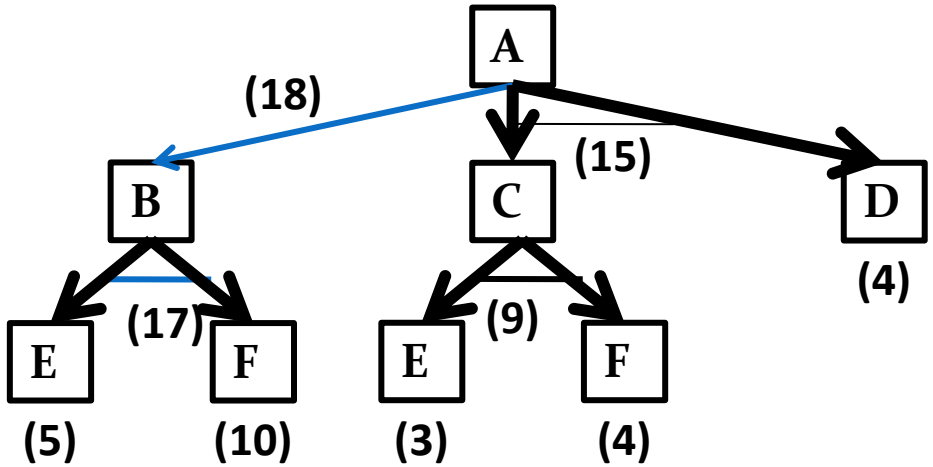


Figure-03

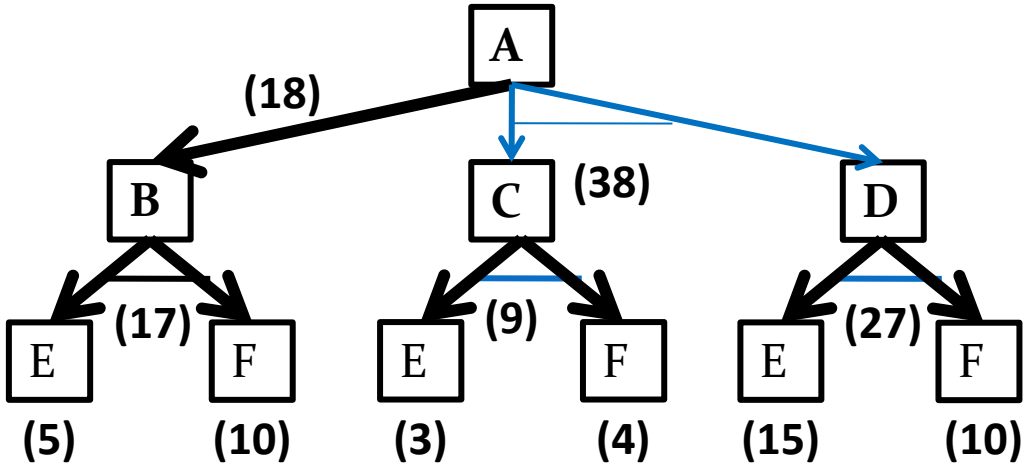


Figure-04

- Darkened Arrows indicate the current best path from each of the expanded nodes.
- In Figure-2 the current best path from the initial node A→ B where B is the only unexpanded node that was being expanded. After this expansion the cost of this path is changed from 6 to 18 and become no longer the current best path from the initial node(because the other path from A have the cost of 9.
- In Figure-3 the current best path from the initial node are A→ C and A→ D where C and D are the unexpanded nodes. Here C is selected for expansion first. After the expansion, this current best remains the same because its new cost is 15 which is still less than the cost of the other path 18.
- In Figure-4 After the expansion of D, the cost of this path becomes 38 and this makes the change in the current best path to the path which having lesser cost of 18.

Before describing an algorithm for searching AND_OR graph, we need to exploit a value called *FUTILITY*. This corresponds to a threshold value , any solution with cost above it, is too expensive to be practical. when the estimated cost of the solution becomes greater than the value of *FUTILITY,* then the search will be abandoned.

Problem Reduction Algorithm:

1. Initialize a graph **G** with only *starting node* named as *INIT.*
2. Repeat the followings until INIT is labeled as SOLVED or h(INIT) becomes greater than *FUTILITY.*
   a) Traverse the graph G starting from INIT and following the current best path, accumulate the set of nodes that are on that path and have not expanded or labeled as SOLVED
   b) Pick up one of these unexpanded node and call it as NODE. Generate the successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each successor that is not an ancestor of NODE do the following:
      i. Add successor to G.
      ii. If the successor is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0, otherwise compute its h.

c) Change the h(NODE) to reflect the new information provided by its successors. Propagate this newly discovered information up through the graph G by doing the following:

i. If any node contains a successor arc whose descendent are all SOLVED then label the node itself labeled as SOLVED.

ii. At each node that is visited while going up in the graph G, decide which of its successor arc is most promising and mark it as the part of the current best path. This may cause the current beat path subject to change.

CS-208:Artificial Intelligence
Lectures-10
AO* Algorithm

AO* Algorithm

Preliminaries.

- AO* algorithm uses a single structure graph **G**.
- Each node in the graph **G** will point both down to its immediate successors and up to its immediate predecessor.
- Each node in the graph **G** will also have associated with it an ***h*** value (an estimated cost of the path from itself to a set of solution nodes)

AO* Algorithm

Step-1: Initialize the graph **G** consist of anode representing the initial state and call it as *INIT.* Compute ***h***(*INIT*).

Step-2: Until INIT is labeled as SOLVED or ***h***(*INIT*) becomes greater than *FUTILITY,* repeat the following

a) Trace the labeled arc from *INIT* and select for expansion of the as yet unexpanded nodes that occurs on this path. Call the selected node as *NODE*.

b) Generate the successors of the *NOD*E. If there are none then assign FUTILITY as ***h*** value of *NODE*(this equivalent to say that *NODE* is not Solvable). If there are successors then for each successor that is not the ancestor of the *NODE* do the following:

   i. Add the successor to the graph **G**.

   ii. If the successor is the terminal node then label it as SOLVED and ***h***(successor)=0.

   iii. If the successor is not a terminal node then compute ***h***(successor)

c) Propagate the newly discovered information up in the graph **G** by doing the following:

Let ***S*** be a set of nodes that have labeled as SOLVED or whose h value have been changed. So this information need to be propagated back to their parents. Initialize ***S*** to NODE. Until ***S*** becomes empty repeat the following:

i. If possible select a node from ***S*** such that none of its descendents in Graph occurs in ***S.*** If there is no such node then select any node. Call this node as CURRENT and remove it from ***S.***

ii. Compute the cost of each arc emerging from CURRENT. Assign the minimum of the costs just computed for arcs emerging from CURRENT to h(CURRENT).

iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost that were computed in the previous step.

iv. Mark CURRENT as SOLVED if all the nodes connected to it though the newly labeled arc has been labeled as SOLVED.

v. If CURRENT has been SOLVED or the cost of CURRENT was just changed then its new status must be propagated up in the Graph. So add all the ancestors of the CURRENT to ***S.***

CS-208:Artificial Intelligence
Lectures-11
Game Playing

Game Playing

Games appeared to be a good domain to explore machine intelligence because of two reasons

1. *Games provide a structured task in which it is very easy to measure success or failure*.
2. *Games generally do not require large amount of knowledge*(this is true for simplest games).

In games, the player know what they have done and what they can do. Here we are interested in two players game with the objective that one of the two players has to win or at least draw the game.

The problem reduction approach can also be used to find a winning strategy through the process of proving that the game can be won.

Finding the Winning Strategy

Suppose we name the two players as PLUS and MINUS. Consider the problem of finding the strategy for the PLUS PLAYER starting with a given configuration represented by $\mathbf{X^t}$.

Where

t stands for either **+** or **-**.

$\mathbf{X^+}$ represents a configuration in which it is the PLUS turn to make the next move.

$\mathbf{X^-}$ represents a configuration in which it is the MINUS turn to make the next move.

We would like to prove that PLUS can win from $\mathbf{X^t}$ or at least PLUS can draw from $\mathbf{X^t}$ and let us describe the problem of proving that PLUS can win from the configuration X^t by the expression $W(\mathbf{X^t})$.

Suppose it is PLUS turn to move next from the configuration $\mathbf{X^+}$ and there are N legal moves. It will result in configuration $W(X_1^-)$, $W(X_2^-)$, $W(X_3^-)$, . . .$W(X_N^-)$. The problem reduction operator will generate a game tree.

Illustration: Grundy's Game

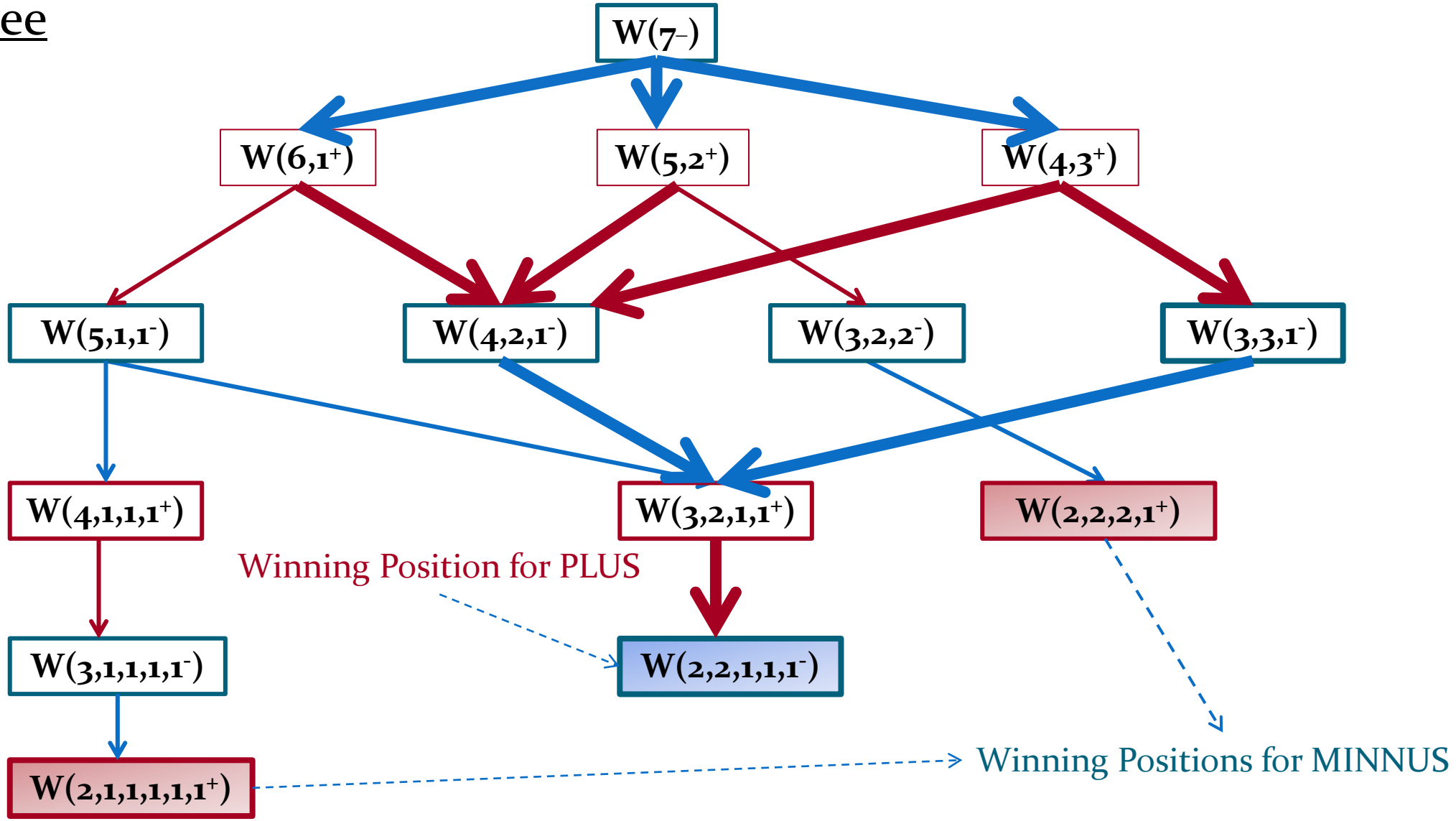
The Rules of the Grundy's Game: Two players have a single pile of objects say a stack of coins in front of them. The first player divides the original stacks in to two stacks that must be unequal. Each player alternatively does the same to some single stack until every stack has either just one penny or two. The player who first can not play is the lose

Problem: *Generate the Game Tree from the Starting Configuration* **$W(7^+)$,** *and Find the Winning strategy*

<u>Two Things can be done to improve the effectiveness of the search based problem solving:</u>

I. Improve the generate procedure, so that only good move are generated.
II. Improve the test procedure so that the best path will be recognized and explored.

Game Tree


W(7-)
W(6,1+)
W(5,2+)
W(4,3+)
W(5,1,1-)
W(4,2,1-)
W(3,2,2-)
W(3,3,1-)
W(4,1,1,1+)
W(3,2,1,1+)
W(2,2,2,1+)
Winning Position for PLUS
W(3,1,1,1,1-)
W(2,2,1,1,1-)
W(2,1,1,1,1,1+)
Winning Positions for MINNUS

Minimax Procedure

-It is a look ahead procedure

- It is a depth limited depth first procedure

Suppose a situation analyzer that converts overall judgments about a situation in to a single quality number. where

- +ve number by convention favours to one player
- -ve number by convention favours to other player
- The degree of favour goes with the absolute value of the number.

The procedure of determining the quality number is called *Static Evaluation*. At the end of limited exploration of move possibilities , the static evaluation score is produced by the situation analyzer called *Static Evaluator*. The player hoping for +ve number is called **maximizing player** and his opponent is the **minimizing player**.

Minimax Algorithm (It is a Recursive Procedure)

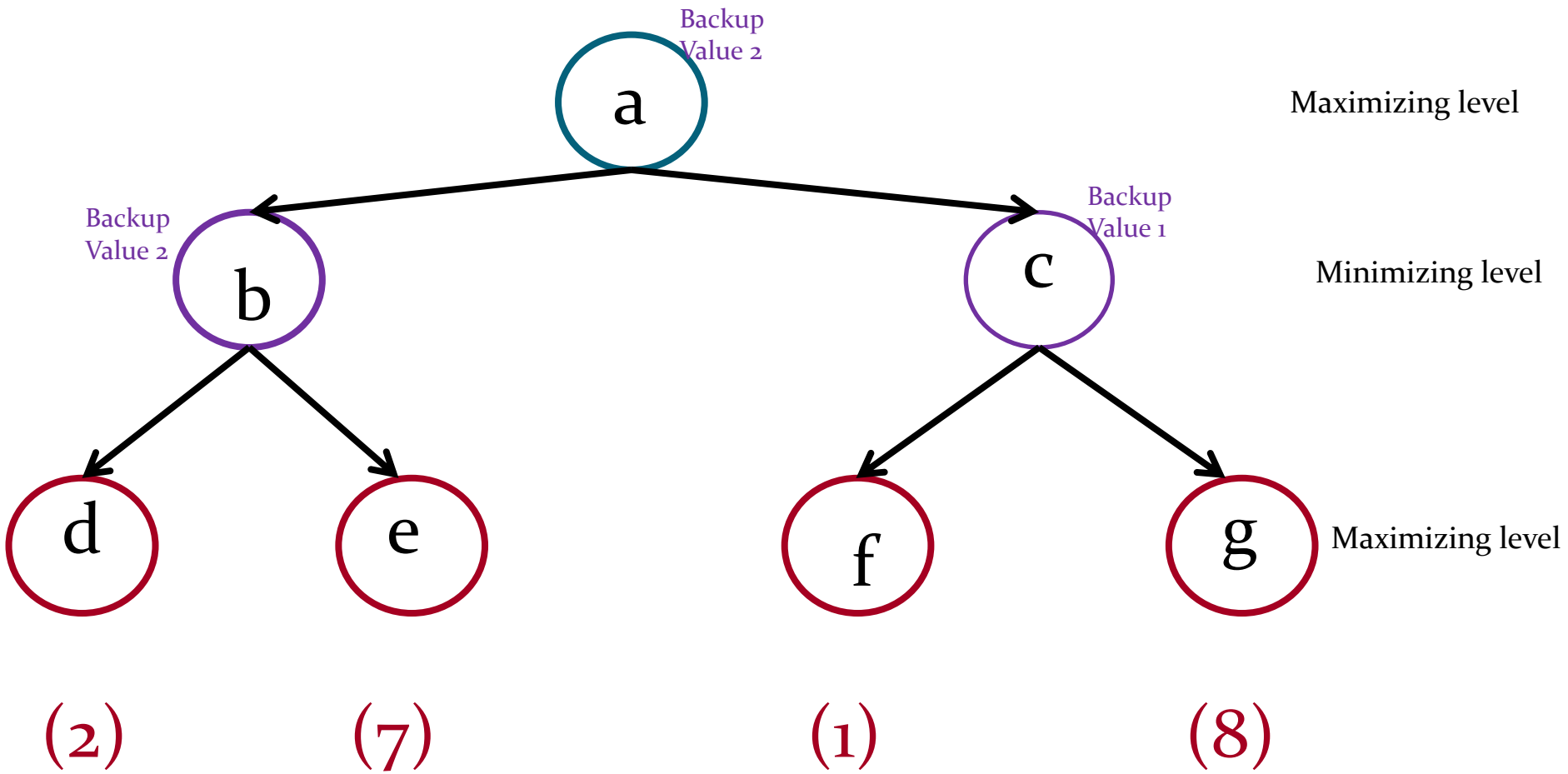
1. Determine if the limit of search has been reached **or**
   If the current level is minimizing level **or**
   If the current level is maximizing level
   a) If the limit of search has been reached then
   compute the static score of the current position relative to the appropriate player and report the result.
   b) If the level is minimizing level then
   *Apply **Minimax** on each successor of the current position and report the minimum of the results*.
   c) If the level is maximizing level *then*
   *Apply **Minimax** on each successor of the current position and report the maximum of the results*.

A good first move can be extracted by the Minimax procedure. If the maximizing player has to choose one among the tip nodes, the player would choose the node having the largest static score value. Therefore the parent of the tip nodes is assigned a back-up value equal to the maximum of the evaluations of the tip nodes

Minimax Algorithm (Refined Version)

1. If the current position is a final one then
    a) Compute the static value of the current position.
    b) Return the static value.
2. If Min is on move then
    a) Generate the successors of the current position
    b) Apply Minimax to each of these successors with Max to move next.
    c) Return the minimum of the results.
3. If Max is on move then
    a) Generate the successors of the current position
    b) Apply Minimax to each of these successors with Min to move next.
    c) Return the maximum of the results.

Example Game Tree



Minimax(Current_Node, Current_Player, Current_Level, Depth_Limit)

Minimax(a, Max, 1 , 3)

Player=Max

Successors = {b, c}

Minimax (b, Min, 2, 3)

Player=Min

Successors = {d, e}

Minimax(d, Max, 3,3)

Report Static Score 2

Minimax(e, Max, 3,3)

Report Static Score 7

Return smallest Static Score (2, 7) =2

Minimax (c, Min, 2, 3)

Player=Min

Successors = {f, g}

Minimax(f, Max, 3,3)

Report 1

Minimax(g, Max, 3,3)

Report 8

Return smallest Static Score (1, 8) = 1

Return Largest Reported value (2, 1) = 2

Minimax with Alpha-Beta Pruning

Minimax with Alpha-Beta Pruning Algorithm (It is also a Recursive Procedure)

1. Determine if the level is the top level **or**
   if the limit of search has been reached **or**
   if the current level is minimizing level **or**
   if the current level is maximizing level

   a) If the level is the top level then
   Alpha be $-\infty$ and Beta be $+\infty$

   b) If the limit of search has been reached then
   Compute the static score of the current position relative to the appropriate player and report the result.

c) If the level is minimizing level then

Repeat the following steps until all the successors of the current position are examined with Minimax or alpha is greater than beta

i. *Set Beta to the smaller of the current beta value and the smallest value so far reported by Minimax working on the current position's successors*

ii. *Use Minimax on the next successor of the current position handling this new application of Minimax and current Alpha Beta values*

Report Beta

d) If the level is maximizing level then

Repeat the following steps until all the successors of the current position are examined with Minimax or alpha is greater than beta

i. *Set Alpha to the larger of the current Alpha value and the biggest value so far reported by Minimax working on the current position's successor*

ii. *Use Minimax on the next successor of the current position handling this new application of Minimax and current Alpha Beta values*

Report Alpha

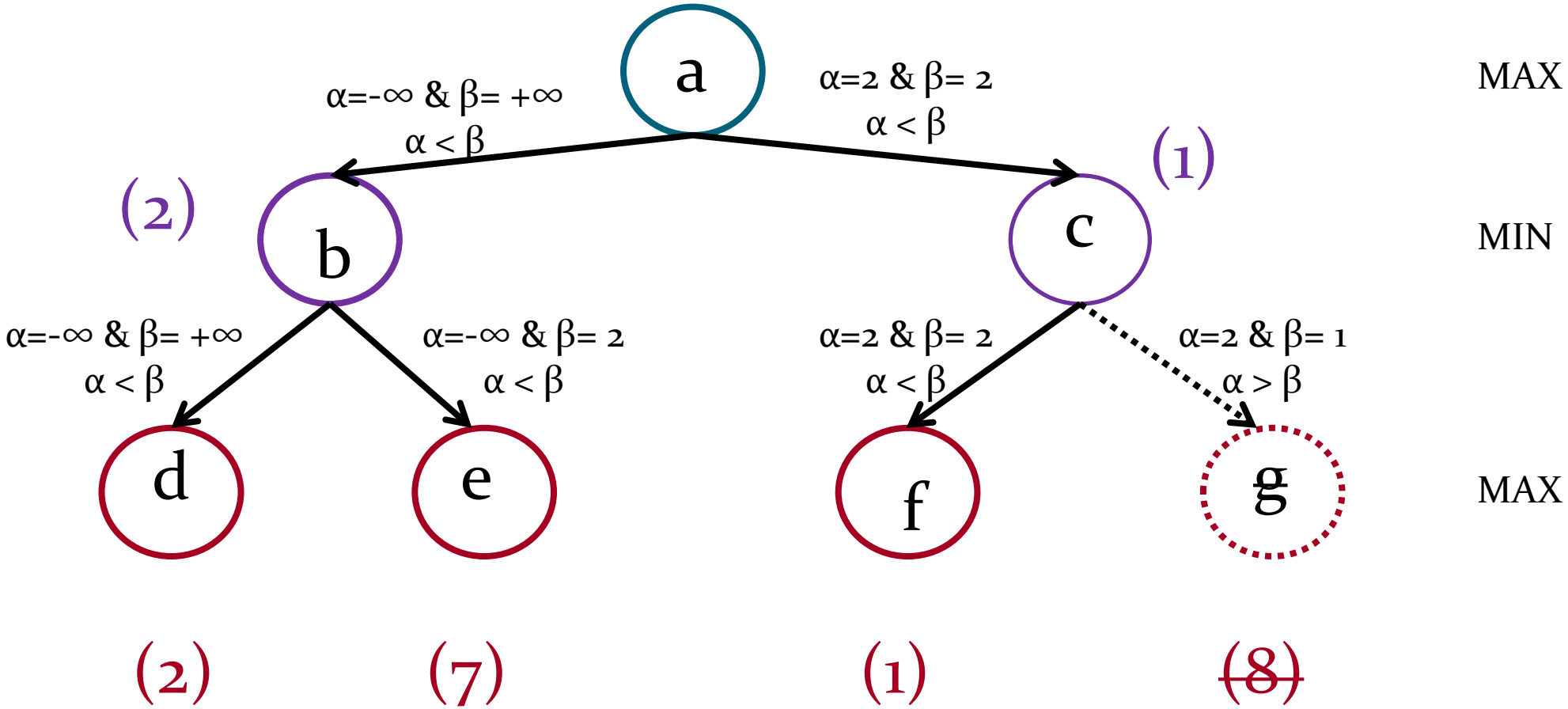
Minimax with Alpha-Beta Pruning (Refined Version)

AlphaBeta()

1. If current_level = top level then
   $\alpha = -\infty$ and
   $\beta = +\infty$
2. If current_level = limit_of _search then
   Compute the static score of the current position relative to the appropriate player and return the result.
3. If Min is on Move then
   a. Generate a list of successors of the current position.
   b. If ($\alpha > \beta$) or (successor_list = [ ]) then
      Terminate
      Return β
   c. β_c = Alpha Beta (successor_list [1])
   d. β = Minimum(β_c , β)
   e. Delete the first element from the successor_list and go back to 3b

4. If Max is on Move then
   a. Generate a list of successors of the current position.
   b. If $(\alpha > \beta)$ or $(\text{successor\_list} = [\ ])$ then
      Terminate
      Return α
   c. α_c = Alpha Beta (successor_list [1])
   d. α = Maximum(α_c , α)
   e. Delete the first element from the successor_list and go back to 4b

- Example Game Tree



AlphaBeta(Current_Node, Current_Player, Current_Level, Depth_Limit)

AlphaBeta((a, Max, 1 , 3)
Player=Max
α = - maxint
B = + maxint
Successors = {b, c}
AlphaBeta((b, Min, 2, 3)
Player=Min
Successors = {d, e}
AlphaBeta((d, Max, 3,3)
Report Static Score 2
β_c = 2
β = min (+maxint, 2) =2
AlphaBeta((e, Max, 3,3)
Report Static Score 7
β_c = 7
β = min (2, 7) =2
Return smallest Static Score =2
α_c = 2

α = Max(-maxint, 2) = 2
AlphaBeta((c, Min, 2, 3)
Player=Min
Successors = {d, e}
AlphaBeta(f, Max, 3,3)
Report Static Score 1
β_c = 1
β = min (2, 1) =1
→Here α>β Pruning Take place
Return Static Score = 1
Return Largest Reported value (2, 1) = 2

CS-208:Artificial Intelligence
Lectures-12
Knowledge Representation
Using
Propositional Logic & Predicate Logic

Knowledge Representation

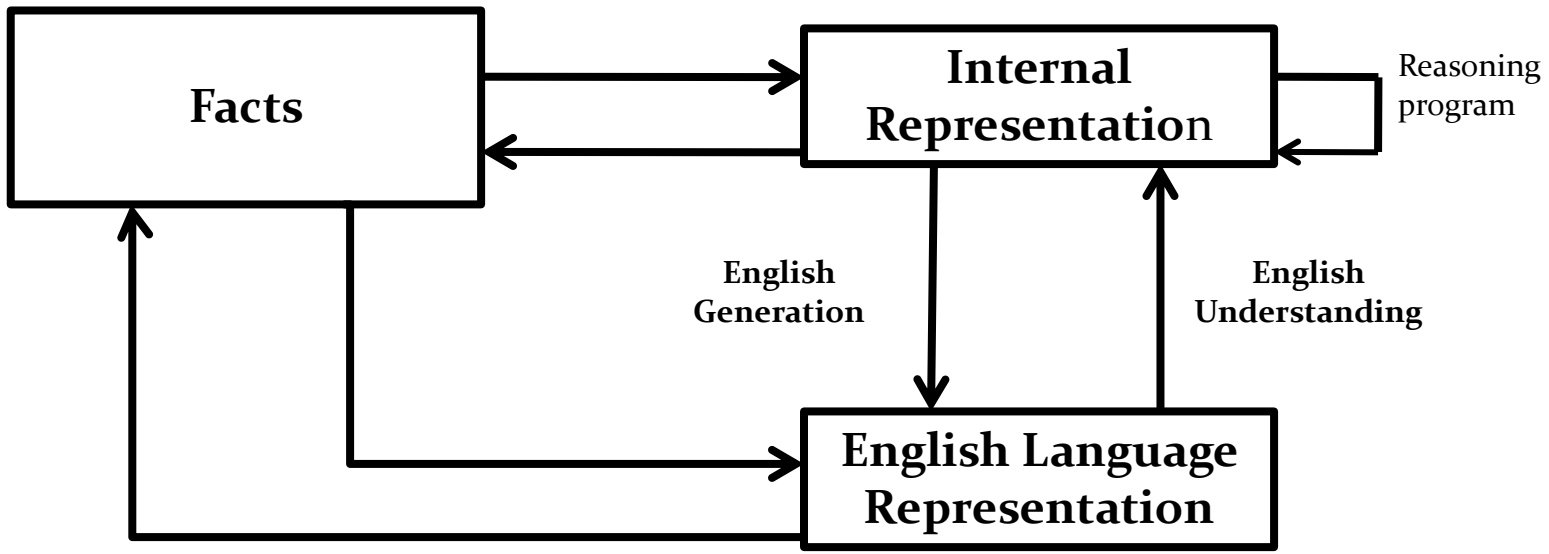
In the previous lectures discussed about the general methods of manipulating the knowledge using search. Now we will look the various ways of representing the knowledge.

There are two entities that are pertained to all kinds of knowledge representations.

- **Facts**: Truth in some relevant world (these are the things we want to represent)
- **Representation of Facts** in some chosen formalism(these are things we actually able to manipulate)

Each representation must associates with some Mapping functions. These functions map from facts to representation and from representation to facts.

An illustrative for Mapping Funtions:



An Illustrative Example

Knowledge Representation Using Mathematical Logic

1. Spot is a dog — *dog(Spot)*
2. All dogs have tails — $\forall x: dog(x) \rightarrow hastail(x)$

A new representation can be generated using the deduction mechanism of logic

hastail(Spot)

An English sentence can be generated from hastail (Spot) using an appropriate mapping function

Spot has a tail

It is important to note that available mapping functions are not always one to one.

- All dog have tails: implies *Every dog has at least one tail* or *Each dog has several tails*
- Every dog has a tail: implies *Every dog has at least one tail* or *There is a tail that every dog has*
  - Here both represent same facts *Every dog has at least one tail*

Knowledge representation using Propositional logic

Knowledge representation using propositional logic is appealing for two reasons.

- It is simple to deal with
- There exist a deduction procedure for it

	English Sentence	**Propositions**
1	It is raining	**Raining**
2	It is sunny	**Sunny**
3	It is windy	**Windy**
4	If it is raining then it is not sunny	**Raining** $\rightarrow \neg$**Sunny**

Limitations-1

Amit is a student **AmitStudent**

Anand is student **AnandStudent**

These two are completely separate assertions and it is not possible to draw any conclusion about similarities between Amit and Anand

Limitations-2

It is also very difficult to represent a simple fact using propositional logic like

"***All men are mortal***"

Knowledge Representation using Predicate Logic

The prime limitations of representing knowledge by using propositional logic can be overcome by using predicate logic. For examples

English Sentence	**Predicate Statement**
Amit is a student	**Student(Amit)**
Anand is student	**Student(Anand)**
All men are mortal	$\forall x$: **Man**$(x) \rightarrow$ **mortal**(x)

- Here the structure of representation reflects the structure the knowledge itself

Convert the following English sentences to Predicate logic statements

1. Marcus was a man
2. Marcus was a Pompeian
3. All Pompeian were Romans
4. Caesar was a ruler
5. All Romans were either loyal to Caesar or hate him
6. Everyone is loyal to someone
7. People only try to assassinate ruler they are not loyal to
8. Marcus tried to assassinate Caesar

Consider the statement:

All Romans were either loyal to Caesar or hate him

$\forall x$: Roman(x)$\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar)

Here the use of inclusive-or is not a correct mapping (or interpretation) because both loyalto(x, Caesar) and hate(x, Caesar) can not be true simultaneously. So we need to use Exclusive-OR instead of Inclusive-OR.

Exclusive-OR Operations on A and B:

$A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$

$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$

Predicate logic statements

1. Man(Marcus)
2. Pompeian(Marcus)
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x)
4. Ruler(Caesar)
5. $\forall$x: Roman(x) $\rightarrow$[(loyalto(x, Caesar) $\vee$hate(x,Caesar))$\wedge$ $\neg$(loyalto(x, Caesar) $\wedge$hate(x,Caesar))]

   or

   $\forall$x: Roman(x) $\rightarrow$[(loyalto(x, Caesar) $\wedge$ $\neg$hate(x,Caesar))$\vee$ ($\neg$loyalto(x, Caesar) $\wedge$hate(x,Caesar))]
6. $\forall$x: $\exists$y: loyalto(x, y)
7. $\forall$x: $\forall$y: Person(x) $\wedge$Ruler(y) $\wedge$trytoassassinate(x, y) $\rightarrow$ $\neg$loyalto(x, y)
8. trytoassassinate(Marcus, Caesar)
9. $\forall$x: Man(x) $\rightarrow$ Person(x)

Suppose, we want to prove by using these predicate statements to answer the question ***Was Marcus loyal to Caesar?*** *Here we will be able to prove Marcus was not loyal to Caesar by using the statements 7 & 8*

Formal Proof

(by reasoning backward from the desired goal)

¬loyal(Marcus, Caesar)

↓ *(substitution-7)*

Person(Marcus)∧ Ruler(Caesar)∧ trytoassassinate(Marcus, Caesar)

↓ *(substitution-4)*

Person(Marcus)∧ trytoassassinate(Marcus, Caesar)

↓ *(substitution-8)*

Person(Marcus)

↓ *(substitution-9)*

Man(Marcus)

↓ **(***substitution-1)*

Nil

In order to prove the goal we need to use **Rule of Inference (***A given goal is transformed in to another goal that in turn to be transformed until no unsatisfied goals remains*).

In order to complete the proof we need to add another fact

9. All men are people

After adding this fact, the last goal has been satisfied and able to produce the proof that Marcus was not loyal to Caesar

Three important issues related to the process of converting the English sentences to logical statements and then using those statements to deduce new ones

1. Many English sentences are ambiguous and choosing the correct interpretation may be difficult.

2. There are often choice of ways of representing the knowledge. Simple representations are desirable but they may preclude certain kinds of reasoning.

3. Even in very simple situations, a set of sentences is unlikely to contain all information necessary to reason about the topic at hand. In order to be able to use set statements effectively, it is usually necessary to have access to another set of statements that represents the fact that people consider too obvious to mention.

Adoption of the Strategies used in Producing the Formal Proof

It is very difficult to know in advance which sentence is to be deduced for the question **Was Marcus loyal to Caesar?** Either ¬**loyalto**(*Marcus, Caesar*) or **loyalto**(*Marcus, Caesar*).

- Forward reasoning (start reasoning from given goal) or Backward reasoning (staring reasoning from the desired goal)
- Using some sort of heuristic rules can be used to decide which answer is more likely.
- Trying to prove simultaneously both the answers and stop when one effort becomes successful.
- Trying both to prove and disprove one answer by using the information gained one of the process to guide the other.

CS-208:Artificial Intelligence
Topic-13 Computational Predicate

Computational Predicates

When the number of facts are not very large and these facts are sufficiently unstructured among themselves then we can express as a combination of individual predicates.

But, suppose we want to represent simple facts such as

gt(1,0)←*1 is greater than 0*	lt(0,1) ← *0 is less than 1*
gt(2,0)	lt(0,2)
gt(2,1)	lt(1,2)
etc.	

- ✗ The above types of facts are infinitely many of them.
- ✗ It would be extremely inefficient to store explicitly as large set of statements.
- ✓ Easy way is to compute each one of these facts as we need it and express them as computational predicates(these computational predicates invoke a procedure instead of searching in the database).

Example for usage of Computational Predicates

1. Marcus was a man
2. Marcus was a Pompeian
3. Marcus was born in 40 AD
4. All men are mortal.
5. All Pompeian died when the volcano erupted in 79AD
6. No mortal lives longer than 150 years.
7. It is now 2020

Is Marcus alive ?

Converting in to predicate statements

1. Man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus, 40)
4. $\forall x$: Man(x) $\rightarrow$ mortal(x)
5. erupted(volcano, 79) $\wedge$ $\forall x$: [Pompeian(x) $\rightarrow$ died(x, 79)]

   5a. erupted(volcano, 79) 5b. $\forall x$: Pompeian(x) $\rightarrow$ died(x, 79)
6. $\forall x$: $\forall t_1$: $\forall t_2$: mortal(x)$\wedge$ born(x, t_1) $\wedge$ gt(t_2-t_1, 150) $\rightarrow$ dead(x, t_2)
7. now = 2020
8. $\forall x$: $\forall t_1$: $\forall t_2$: died(x, t_1) $\wedge$ gt(t_2, t_1) $\rightarrow$ dead(x, t_2)
9. $\forall x$: $\forall t$: alive(x, t) $\rightarrow$ $\neg$dead(x, t)

Formal Proof -1

(by reasoning backward from the desired goal)

alive(Marcus)

↓ (9)

dead(Marcus, now)

↓(8)

died(Marcus, t_1) ∧ gt(now, t_1)

↓(5b)

Pompeian(Marcus) ∧ gt(now, 79)

↓(2)

gt(now, 79)

↓(7)

gt(2020, 79)

↓

Nil

In order to complete the proof we need to add two more facts

8. If some on dies, then he is dead at all later time

9. Alive means not dead

After adding this fact, the last goal has been satisfied and able to produce the proof that Marcus is not alive now

The term Nil at the end indicates that the proof has succeeded.

The sentence *All Pompeian died when the volcano erupted in 79AD* clearly asserts two facts represented below :

erupted(volcano, 79)

∀x: Pompeian(x) → died(x, 79)

Formal Proof -2

(by reasoning backward from the desired goal)

¬ alive(Marcus)

↓ (9)

dead(Marcus, now)

↓(6)

mortal(Marcus)∧ born(Marcus, t_1) ∧ gt(now-t_1, 150)

↓(4)

Man(Marcus)∧ born(Marcus, t_1) ∧ gt(now-t_1, 150)

↓(1)

born(Marcus, t_1) ∧ gt(now-t_1, 150)

↓(3)

gt(now-40, 150)

↓(7)

Gt(2020-40, 150)

↓

Gt (1980, 150)

↓

Nil

$a \wedge b \rightarrow c$

Here a and b are independent sub-goals if they do not share the same bound variables

Even very simple conclusions require many steps to prove,

A variety of process (such as Matching, Substitutions and application of modus ponens/modus tolens) are involved in the production of proof.

CS-208:Artificial Intelligence
Topic -14 Mathematical Logic

Revisiting Basics of Predicate Logic

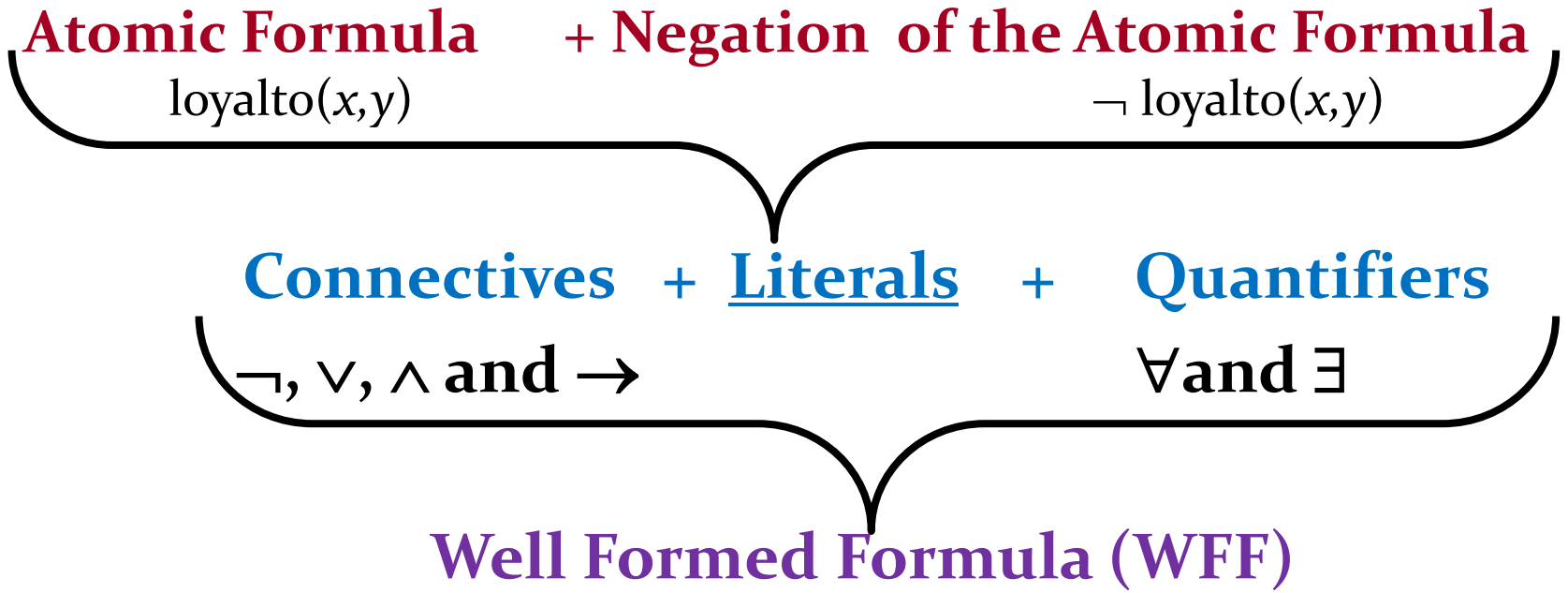
Predicates *are function that map object arguments into True or False*.

Terms *are only things that appear as arguments to predicate*.

- Domain's Objects are Terms
- Variables ranging over a domain's object are Terms
- Functions are Terms. (The arguments to functions and the value returned are objects)

Atomic Formula (*individual predicates together with arguments*)

Examples: **On**(*x, tables*); **loyalto**(*x*, **Ruler**(*Rome*))



Literals *are atomic formula and negated atomic formula*

***Universal* Quantifier:** *(for all)* ∀

Existential Quantifier: *(there exist)* ∃

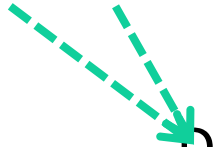
Well Formed Formulas abbreviated by WFFs are defined recursively as follows:

- Literals are WFF.
- WFF connected together by connectives ($\vee$,$\wedge$, $\neg$ and$\rightarrow$) are WFF.
- WFF surrounded by quantifiers($\forall$ and $\exists$) are also WFF.

A $\wedge$ B is called Conjunction

Conjuncts

A $\vee$ B is called Disjunction



Disjuncts

Modus Ponens: *If there is an axiom of the form $A \rightarrow B$ and another axiom of the form A then B logically follows.*

Modus Tolens: *If there is an axiom of the form $A \rightarrow B$ and another axiom of the form $\neg A$ then $\neg B$ logically follows.*

Resolution: *If there is. an axiom of the form $\boldsymbol{A} \vee B$ and another axiom of the form $\neg B \vee C$ then $A \vee C$ logically follows.*

The expression $A \vee C$ is called the ***resolvent*** *of $\boldsymbol{A} \vee B$ and $\neg B \vee C$*

CS-208:Artificial Intelligence
Topic-15: Resolution

Resolution Proof

The resolution theorem proving strategy is to show that the negation of a theorem can not be true.

Step1: *Assume that the negation of the theorem is true.*

Step2: *Show that axioms and assumed negation of the theorem together determines something to be true that can not be true*.

Step3: Conclude that the assumed theorem can not be true, since it leads to contradiction.

Step4: *Conclude that the theorem is true since the assumed negation of the theorem can not be true*.

Proving a theorem by showing its negation can not be true is called

Proof by Refutation

Resolution Procedure

- The resolution operates by taking two clauses that contain the same literal which must occur +ve form in one clause and-ve form in the other.
- The resolvent is obtained by combing all the literals from both the two parent clauses except the ones that cancels.
- If the clause that is produced is an empty clause then contradiction has been found.
- If no contradiction exist then it is possible that the resolution procedure will never terminates.

Conversion of WFF into Clause Form or Canonical Form or Conjunctive Normal Form

Step1: Eliminate $\rightarrow$: By using $a \rightarrow b$ is equivalent to $\neg a \vee b$

Step2: Reduce the scope of ¬ by using

1. $\neg(\neg a) = a$
2. De Morgan's Laws
   i. $\neg(a \vee b) = \neg a \wedge \neg b$
   ii. $\neg(a \wedge b) = \neg a \vee \neg b$
3. Standard correspondence between quantifiers
   i. $\neg \forall x: P(x) = \exists x: \neg P(x)$
   ii. $\neg \exists x: P(x) = \forall x: \neg P(x)$

Step3: Standardize Variables by binding unique variable to each quantifier

$\forall x: P(x) \wedge \forall x: Q(x)$ has to be changed in to $\forall x: P(x) \wedge \forall y: Q(y)$

Step4: Move all the quantifiers to the left of the formula

Prefix of quantifiers followed by a matrix

At this point the formula is in Prenex Normal Form

Step5: Eliminate the existential quantifiers by using Skolem Constant and Skolem Function.

$\exists y$:**Presidnt**(y) → **President**(***S1***)

(Here S1 is a function with no arguments and produce a value that satisfies President.)

If the existential quantifier occurs within the scope of universal quantifier, then the value that satisfy the predicate will depend on the value of the universally quantified variable

$\forall x: \exists y:$ **fatherof**(y,x) → $\forall x:$ **fatherof**(**s2**(x), x)

(Here the function S2 that takes the argument *x* and produce the value which satisfies the predicate.)

These generated functions S1 and S2 are called Skolem functions. Sometime ones with no argument are called Skolem constants.

Step6: Drop the prefix of quantifiers: At this point all the variables are universally quantified so prefix can be dropped

Step7: Convert the matrix into conjunction of disjuncts: $(\_ \vee \_ \vee \_ \vee \_) \wedge (\_ \vee \_ \vee \_) \wedge (\_ \vee \_ \vee \_)$ … This conversion can be done by using the associative and distributive laws. $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

Step8: Call each conjuncts as a separate clause: In order for WFF to be true, all the clauses that are generated from it must be true.

Step9: Standardize apart the variables in the set of clause generated: Here we rename the variables so that no two clauses make reference to the same variable.

CS-208:Artificial Intelligence

Topic-16: Resolution in Propositional Logic

Resolution in Propositional Logic

Input: Procedure for producing proof by resolution of proposition ***S*** with respect to given set of axioms ***F***

Step1: Convert all the propositions of F in to Clause Form

Step2: Negate S and convert the result in to clause form. Add it to the set of clauses obtained in step1.

Step3: Repeat Until either a contradiction is found or no progress can be made

a) Select two clauses. call them as the parent clause.

b) Resolve the parent clauses. The resulting clause is called resolvent will be the disjunction of all the literals from both the parent clauses with following exception: If there are any pairs of literals L and ¬L such that one of the parent clauses contains L and the other contains ¬L then eliminate both L and ¬L from the resolvent.

c) if the resolvent is an empty clause then a contradiction has been found . If it is not, then add the resolvent to the set clauses available to the procedure.

An Illustrative Example for Resolution in Propositional Logic

Given set of Axioms

1. p
2. $(p \wedge q) \rightarrow r$
3. $(s \vee t) \rightarrow q$
4. t

Prove: r

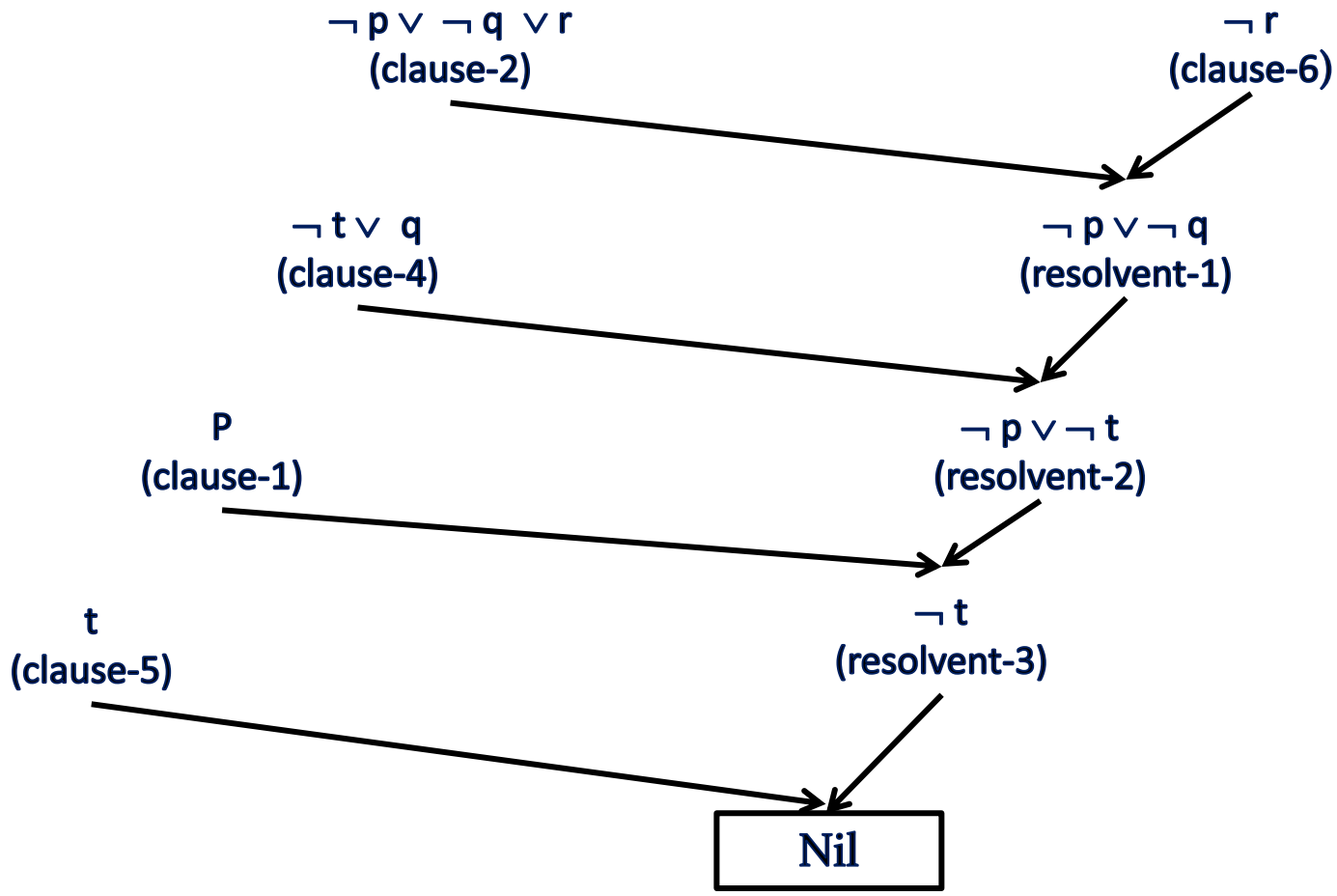
Resolution Step1: convert given set of axioms in to clause form

	Conversion steps	Description	Clauses
1	p	Already in the clause form	Clause-1
2	$(p \wedge q) \rightarrow r$		
	$\neg(p \wedge q) \vee r$	By conversion step-1	
	$(\neg p \vee \neg q) \vee r$	By conversion step-2	Clause-2
3	$(s \vee t) \rightarrow q$		
	$\neg(s \vee t) \vee q$	By conversion step-1	
	$(\neg s \wedge \neg t) \vee q$	By conversion step-2	
	$(\neg s \vee q) \wedge (\neg t \vee q)$	By conversion step-7	
	$\neg s \vee q$	By conversion step-8	Clause-3
	$\neg t \vee q$	By conversion step-8	Clause-4
4	t	Already in the clause form	Clause-5

Resolution Step2: Negate r and convert ¬ r into clause form

	Conversion steps	Description	Clauses
1	¬r	Already in the clause form	Clause-6

Resolution Step3:



An empty clause indicates ¬***r*** can not be true as it was concluded with full of contradictions. So ***r*** must be true.

CS-208:Artificial Intelligence
Topic-17: Resolution in Predicate Logic

Resolution in Predicate Logic

In propositional logic, it is easy to determine that two literals can not be true at the same time. But in predicate logic it is not easy as the arguments are also need to be considered

Man(Marcus) and ¬Man(Marcus) is a contradiction

Man(Marcus) and ¬Man(Spot) is not a contradiction

So we need a matching process that compares two literals and discover whether there exist a set of substitution that matches them identical.

Basic idea of Unification

- ✗ Different constant or predicate can not match
- ✓ Identical constant or predicate can match
- ✓ A variable can match with
  - Another variable
  - Any constant
  - predicate expression (except that it should not contain any instance of the variable being matched).

Complication in finding the consistent substitutions

Finding the consistent substitution is very complex process:

Wrong way of finding the substitutions:

P(*x*, *x*) $\xrightarrow{y/x}$ P(*y*, *x*) $\xrightarrow{z/x}$ P(y, z)

P(*y*, *z*) P(*y*, *z*) P(y, z)

? *y*/*x* and *z*/*x* can not be the correct substitutions because two different variables *y* and *z* can not be substituted for a same single variable *x*.

Right way of finding the substitution:

Find the first substation and apply to the rest of the arguments and the find other substitution in the similar way

P(x, x) $\xrightarrow{y/x}$ P(*y*, *y*) $\xrightarrow{z/y}$ P(*y*, *z*)

P(*y*, *z*) P(*y*, *z*) P(*y*, *z*)

Objective of the Unification Procedure

The objective is to find at least one substitution that causes two literals to match.

Example:

Hate(x, y)

Hate(Marcus, y)

Two sets of substitutions that can match these two literals:

- ✓ Marcus/ x, z/y
- ✓ Marcus/x, y/z

Unification Algorithm

Unify(L_1,L_2)

// unifies two literals L_1 and L_2

1. If L_1 or L_2 is a variable or constant, then:
   a) If L_1 and L_2 are identical, then return NIL.
   b) Else if L_1 is a variable, then if L_1 occurs in L_2 then return FAIL, else return {(L_2/ L_1 }.
   c) Else if L_2 is a variable, then if L_2 occurs in L_1 then return FAIL, else return {(L_1 /L_2)}.
   d) Else return FAIL.
2. If the initial predicate symbols in L_1 and L_2 are not identical, then return FAIL.
3. If L_1 and L_2 have a different number of arguments, then return FAIL
4. Set SUBST to NIL.
5. For i ← 1 to number of arguments in L_1 do
   a) Call Unify with the i^{th} argument of L_1 and the i^{th} argument of L_2, putting result in S.
   b) If S = FAIL then return FAIL.
   c) If S ≠ NIL then:
      i. Apply S to the remainder of both L_1 and L_2.
      ii. SUBST := APPEND(S, SUBST).
6. Return SUBST.

- At the end of the procedure **SUBST** will contain all substitutions used to unify L_1 and L_2.
- If **SUBST** is an empty list then it indicates that match was found without any substitution.
- If **SUBST** contains a single value Fail then it indicates that unification procedure has failed.

Resolution Procedure in Predicate Logic

Step1: Convert all the propositions (axioms) of *F* to clause form.

Step2: Negate *S* and convert the result to clause form. Add it to the set of clauses obtained in Step1.

Step3: Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.

a. Select two clauses. Call these the parent clauses.

b. Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals $T1$ and $\neg T2$ such that one of the parent clauses contains $T1$ and the other contains $\neg T2$ and if $T1$ and $T2$ are unifiable, then neither $T1$ nor $\neg T2$ should appear in the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

c. If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

An illustrative Example for Proving that *Marcus hated Caesar* using Resolution

.

English Sentence (Given set of axioms)	Predicate WFF
1) Marcus was a man	Man(Marcus)
2) Marcus was a Pompeian	Pompeian(Marcus)
3) All Pompeian were Romans	$\forall x$: Pompeian(x)$\rightarrow$ Roman(x)
4) Caesar was a ruler	Ruler(Caesar)
5) All Romans were either loyal to Caesar or hate him	$\forall x$:{Roman(x)$\rightarrow$[(loyalto(x,Caesar) $\vee$ hate(x,Caesar))$\wedge\neg$(loyalto(x,Caesar)$\wedge$hate(x,Caesar))]}
6) Everyone is loyal to someone	$\forall$x: $\exists$:y loyalto(x,y)
7) Men only try to assassinate ruler they are not loyal to	$\forall x$:$\forall y$:Man(x) $\wedge$ Ruler(y) $\wedge$ trytoassassinate(x,y) $\rightarrow$ $\neg$loyalto(x,y)
8) Marcus tried to assassinate Caesar	trytoassassinate(Marcus, Caesar)

Resolution Step-1

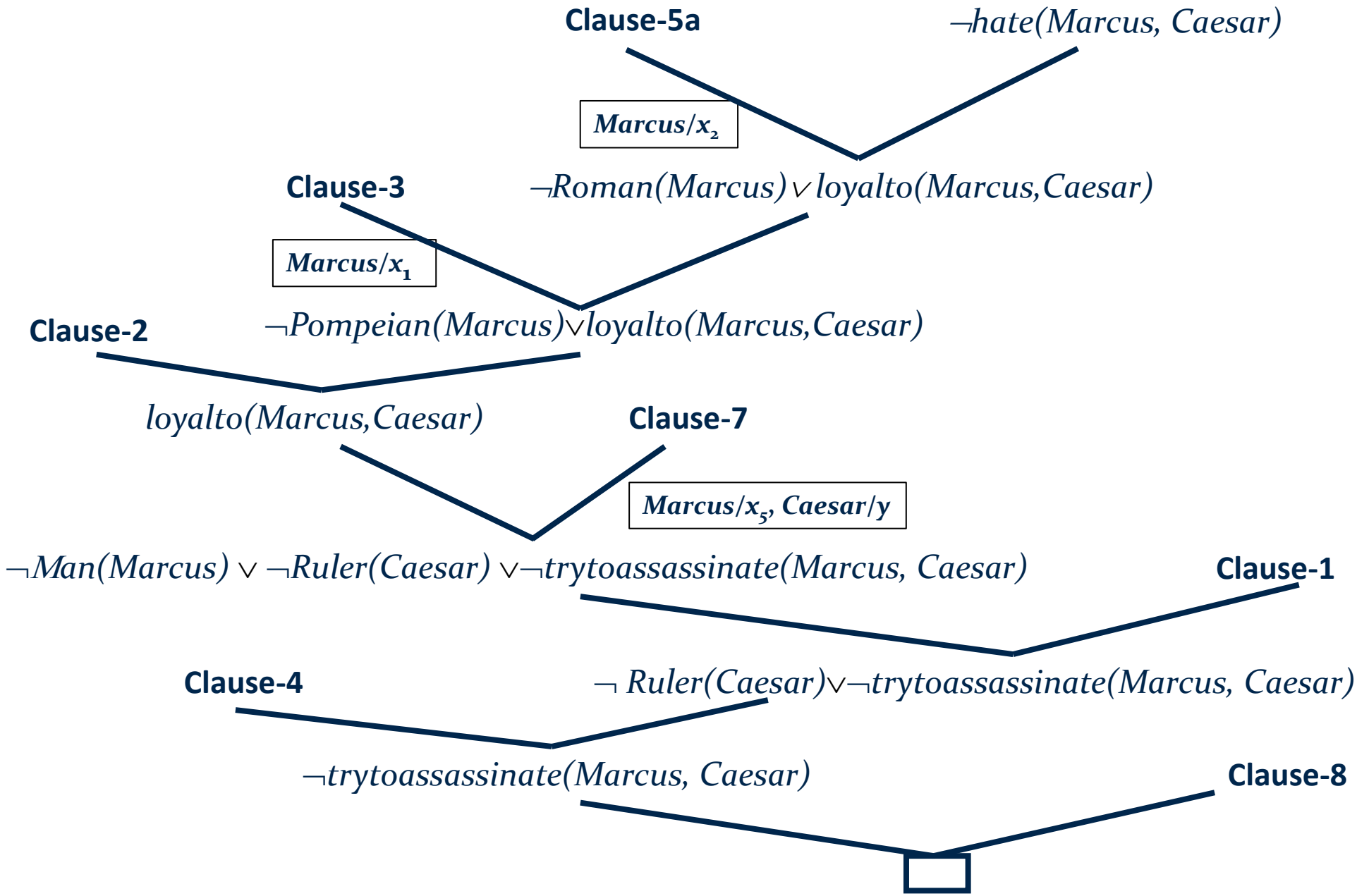
.

Predicate WFF	Clauses Form	
Man(Marcus)	Man(Marcus)	Clause-1
Pompeian(Marcus)	Pompeian(Marcus)	Clause-2
$\forall x$: Pompeian(x)$\rightarrow$ Roman(x)	$\neg$ Pompeian(x_1)$\vee$Roman(x_1)	Clause-3
Ruler(Caesar)	Ruler(Caesar)	Clause-4
$\forall x$:{Roman(x)$\rightarrow$[(loyalto(x, Caesar)$\vee$hate(x, Caesar))$\wedge\neg$(loyalto(x, Caesar)$\wedge$hate(x, Caesar))]}	$\neg$Roman(x_2) $\vee$ loyalto(x_2, Caesar) $\vee$ hate(x_2, Caesar)	Clause-5a
	$\neg$Roman(x_3) $\vee$ $\neg$loyalto(x_3, Caesar) $\vee$ $\neg$hate(x_3, Caesar)	Clause-5b
$\forall x$: $\exists$:y loyalto(x,y)	loyalto(x_4, S1(x_4))	Clause-6
$\forall x$:$\forall$y:Man(x) $\wedge$ Ruler(y) $\wedge$ trytoassassinate(x,y) $\rightarrow$ $\neg$loyalto(x,y)	$\neg$Man(x_5) $\vee$ $\neg$Ruler(y) $\vee$ $\neg$trytoassassinate(x_5, y) $\vee$ $\neg$loyalto (x_5, y)	Clause-7
trytoassassinate(Marcus, Caesar)	trytoassassinate(Marcus, Caesar)	Clause-8

Resolution Proof for Marcus hated Caesar

Statement to be Proved	WFF	Negate the WFF	Clause Form	
Marcus hated Caesar	hate(Marcus, Caesar)	¬hate(Marcus, Caesar)	¬hate(Marcus, Caesar)	**Clause-9**

Resolution Proof



Question Answering using Resolution

It was shown that resolution can be used to answer yes-no questions.

It can also be shown that the resolution can be used to answer fill-in-the-blank questions. This can be done by adding an additional expression to the one which is used to try to find a contradiction. This new additional expression (dummy expression) will simply the one that is to be proved true (it will be the negation of the expression that is actually used in the resolution).

This dummy expression will not interfere with the resolution process and it is tagged or underlined to indicate that it is a dummy expression. It will carry along the resolution and each time the unification is done, the variables in the dummy expression is also be bound just as the ones in the clause that are actively being used.

Instead of terminating on reaching the nil expression, the resolution will terminate when it will only left with the dummy expression. At this point, the bindings of variables in the dummy expression will provide the answer.

An illustrative example for answering fill-in-the-blank questions by using resolution

When did Marcus die?

To find answer	WFF	Negate	Adding the dummy Clause	Beginning parent Clause
When did Marcus die?	died(Marcus, t)	¬died(Marcus, t)	¬died(Marcus, t) ∨died(Marcus, t)	¬died(Marcus, t) ∨died(Marcus, t)

Translating the Sentences into Formulas in Predicate Logic.

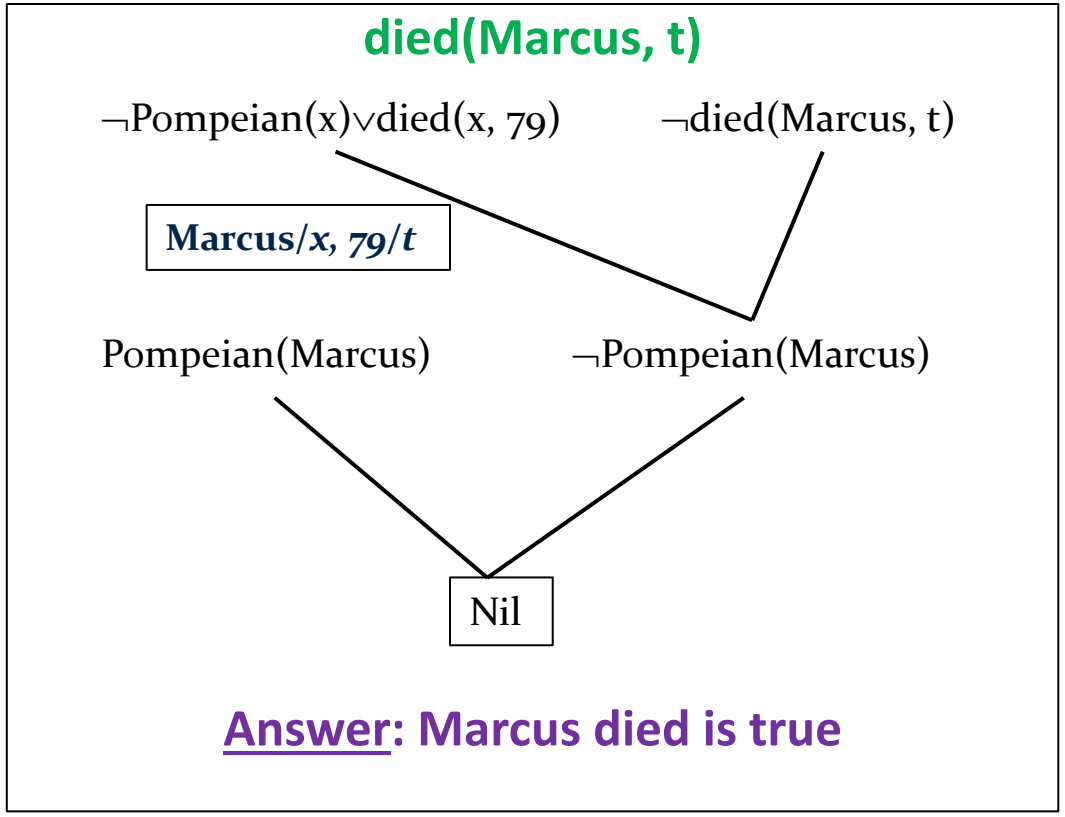
English Sentence Given set of axioms	WFF
Marcus was a man	Man(Marcus)
Marcus was a Pompeian	Pompeian(Marcus)
All Pompeian died when the volcano erupted in 79AD	erupted(volcano, 79)$\wedge\forall$x:Pompeian(x)$\rightarrow$ died(x, 79)

Convert the Formulas into Clause Form

WFF	Clause Form	
Man(Marcus)	Man(Marcus)	**Clause-1**
Pompeian(Marcus)	Pompeian(Marcus)	**Clause-2**
erupted(volcano, 79)$\wedge\forall$x:Pompeian(x)$\rightarrow$ died(x, 79)	erupted(volcano, 79)	**Clause-3**
	$\neg$Pompeian(x) $\vee$died(x, 79)	**Clause-4**

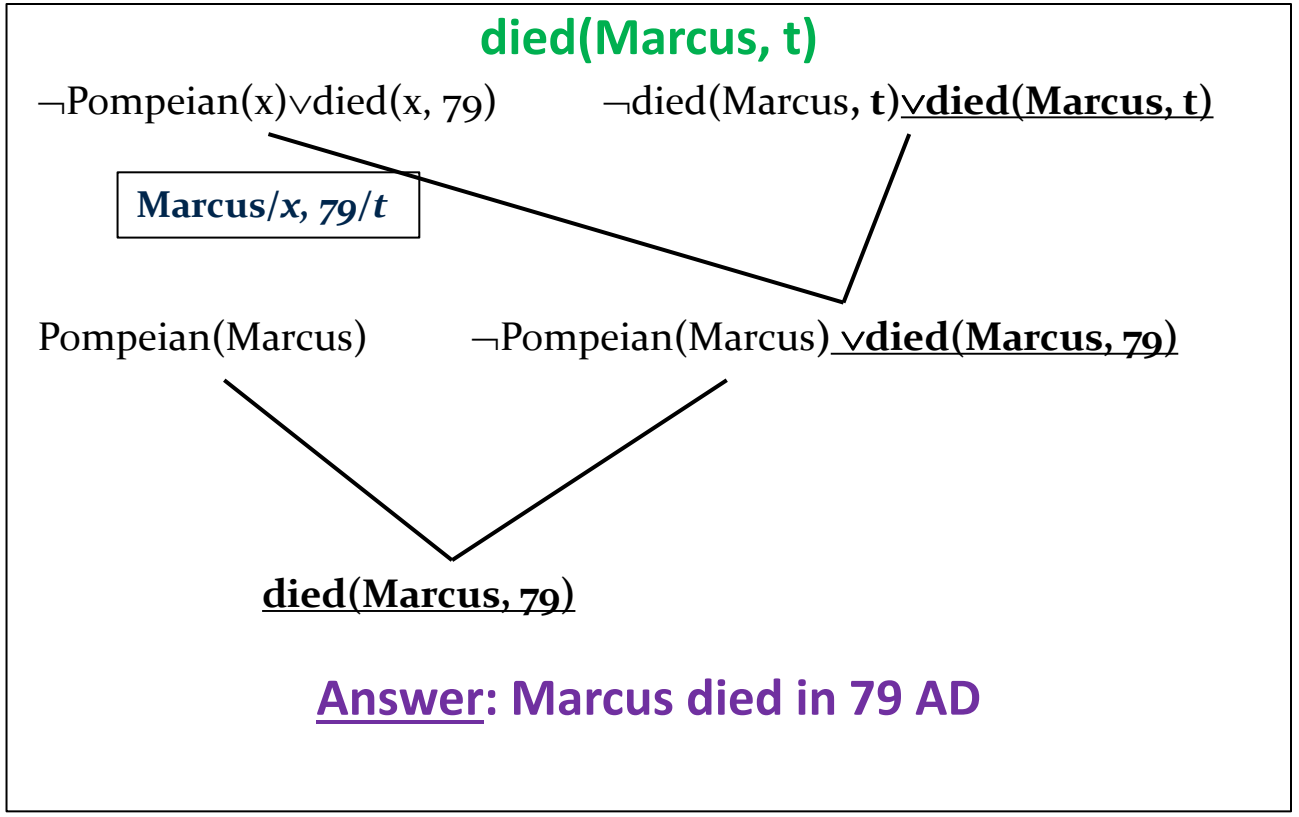
Comparative example for answering yes-no & fill-in-the-blank question by using Resolution

Answering yes-no question using resolution



Answer: Marcus died is true

Answering fill-in-the-blank question using resolution



Answer: Marcus died in 79 AD

CS-208:Artificial Intelligence
Topic-18: Declarative Representation

Declarative Knowledge Representation

The methods of representing knowledge using predicate and propositional logics are often useful for representing simple facts. The major advantage of these methods is that they can be combined with powerful inference mechanism called resolution. But it is very difficult to represent complex knowledge using these representations.

Properties of Good Knowledge Representation System

Representational Adequacy	**The ability to represent all kinds of knowledge that are needed in a problem domain.**
Inferential Adequacy	**The ability to manipulate the representational structure in such a way to drive new structure corresponds to new knowledge.**
Inferential Efficiency	**The ability to incorporate in to knowledge structure the additional information that can be used to focus attention of the inference mechanism in a most promising direction.**
Acquisitional Efficiency	**The ability to acquire new information easily.**

Several techniques have been developed to meet these properties and they are classified in to two types

Declarative Methods	Procedural Methods
1. Each fact need only be stored once regardless of number of different ways it can be used. 2. It is easy to add new facts to the system without changing other facts or small procedure.	1. It is easy to represent knowledge of how to do things. 2. It is easy to represent the knowledge that does not fit in to many declarative scheme. 3. It is easy to represent heuristic knowledge of how to do things efficiently.

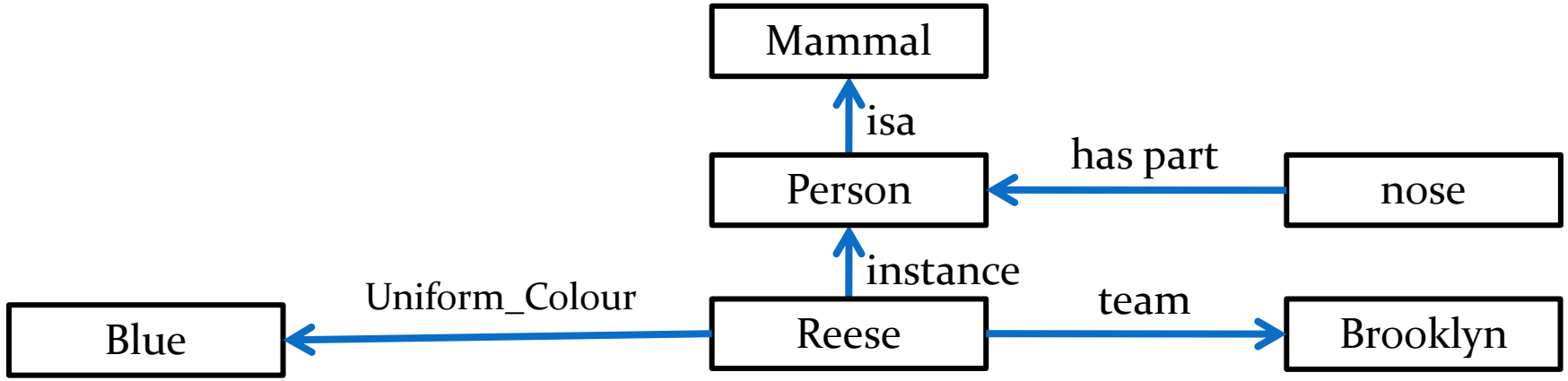
There are many declarative mechanism for representing knowledge and describe the following in detail:

1. Semantic Net
2. Frames
3. Conceptual Dependency
4. Scripts

Semantic Net

It was originally designed to represent the meanings of the English words. The idea behind the semantic net is that the meaning of a concept comes from the ways in which it is connected to other concepts.

A semantic network consist of nodes and labeled arcs, where each node represents a physical or mental concept that is the information is represented as a set of nodes and each arc represents the relationship between the nodes it connects. An example of semantic net representation is given below:



Intersection Search: Semantic nets were used to find the relationships among the objects by spreading the activation from each of the two nodes and seeing where the activation met. This process is called intersection search. Using this process we can find the connection between Brooklyn and blue.

Representing Non-binary Predicates

Semantic nets are a natural way to represent the relationship that would appear as ground instances of binary predicates in predicate logic.

isa(person, mammal)

instance(Reese, person)

team(Reese, Brooklyn)

Uniform_colour(Reese, blue)

Representing Unary Predicates

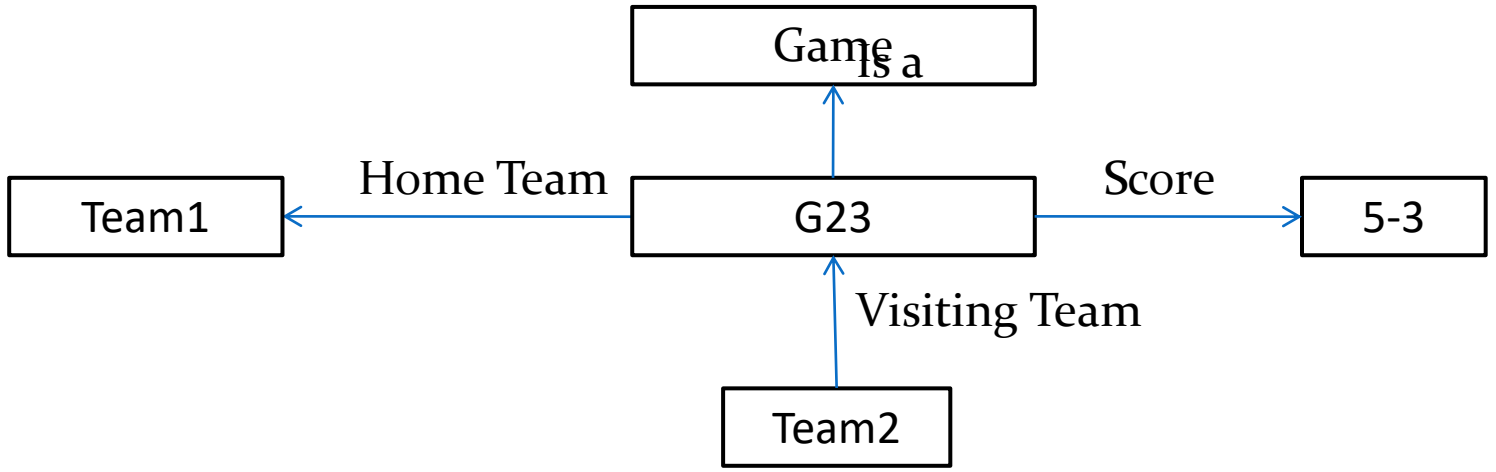
One place predicate can be converted into two place predicates

Example: Man(Marcus) can be converted in to instance(Man, Marcus)

Representing Three or More Place Predicates

Three or more place predicates can also be converted into a binary form by creating new objects representing the entire predicate statemnt.

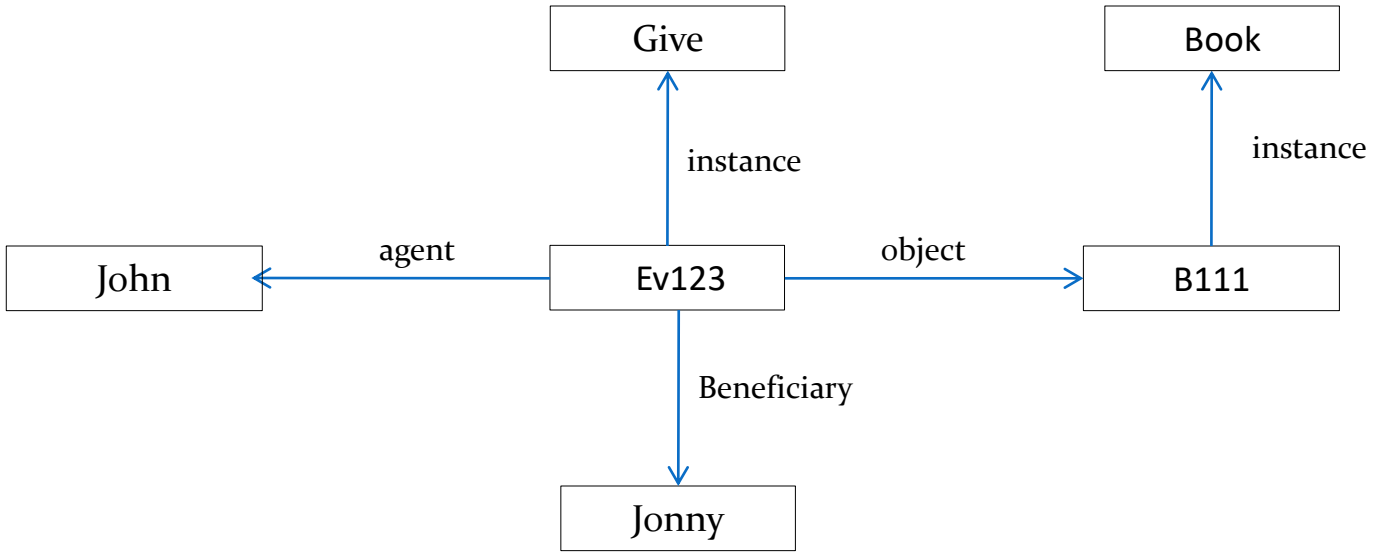
Example: Score(Team1, Team2, 5-3)



Semantic Net representing a sentence

We can represent a sentence in semantic net

Example: John gave the book to Johnny



Frames

Idea: there exist a great deal of evidence that people do not analyze new situations from the scratch and then build new knowledge to describe those situations. Instead they have available in memory a large collection of structures representing the previous experience with objects, locations situations and people.

To analyze new experience they invoke appropriate stored structure and then fill them in with details of current events.

A general mechanism designed for computer representation of such common knowledge is called Frame.

According to Minsky a frame is a static data structure used to represent well understood, stereotyped situations. Frame like structures seem to organize our own knowledge of the world. We adjust to ever new situations by calling up information structured by past experience. We then specially fashion or revise the details of these past experience to represent the individual differences for the new situation.

Frame Description : Frames represent knowledge as structured objects. A frame is a collection labeled slots with their values that together describe a sterotyped object, act or events in the world.

The internal designed structure make them useful in specific kinds of problem solving task.

Name of the Frame	
Slot	**Slot's Value**

Representing Knowledge Using Frames

A frame describe a class of objects and it consists of a collection of slots that describe aspects of the objects. These slots are filled by frame describe by particular object.

Associated with each slot may be a set of conditions that must be met any fills for it.

Each slot is filled with a default value sothat in the absence of a specific information to the contrary, things can be assumed to be as they usually are.

Procedural information may be associated with particular slots

Conceptual Dependency (CD)

Conceptual Dependency: It is a theory about how to represent the meaning of natural language sentences in a way that

i. Facilitate the drawing of inference from sentences
ii. independent of the language in which the sentences were originally stated.

Representing Knowledge using CD

The CD representation is built out of conceptual primitives that can be combined to form the meanings of words in any particular language.

CD provides both a structure and a specific set of primitives out of which representation of any particular piece of information can be constructed.

Primitive Acts

ATRANS -- Transfer of an abstract relationship. E*xample: give*.

PTRANS -- Transfer of the physical location of an object. E*xample: go*.

PROPEL -- Application of a physical force to an object. E*xample: push*.

MTRANS -- Transfer of mental information. E*xample: tell*.

MBUILD-- Construct new information from old. E*xample: decide*.

SPEAK -- Utter a sound. E*xample say*.

ATTEND -- Focus a sense on a stimulus. E*xample: listen, watch*.

MOVE -- Movement of a body part by owner. E*xample: punch, kick*.

GRASP -- Actor grasping an object. E*xample: clutch*.

INGEST -- Actor ingesting an object. E*xample: eat*.

EXPEL-- Expulsion of something out of body of an animal in to the world : cry

Six primitive conceptual categories to provide building blocks

PP -- Real world objects.

ACT -- Real world actions.

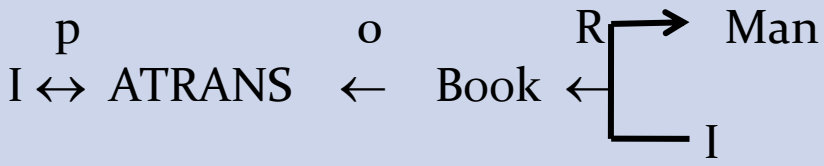
PA -- Attributes of objects.

AA -- Attributes of actions.

T -- Times.

LOC-- Locations.

Example: I gave the man *a book*



→ Arrows indicate the direction of dependency.

↔ Double arrows indicate *two-way* links between the actor (PP) and action (ACT).

p indicate past tense

o object case relation

R recipient-donor relation

Conceptual Dependency not only provide a structure through which knowledge can be represented but also a specific set of building block from which representation can be built.

Second set of building block is set of allowable dependencies among conceptualization described in a sentence. There are four primitive conceptual categories

ACT's– Actions

PP's – Objects (picture producer)

AA's – Modifier of actions (action aider)

PA's – Modifier of PP's (Picture aider)

Example: *John ran* p

Rule1: PP ⟺ ACT's John ⟺ PTRANS

Scripts

Definition A script is a structured representation describing a stereotyped sequence of events in a particular context. Scripts consist of a set slots. Associated with each slot

A script is composed of following components:

- Entry conditions that must be true for the script to be called.
- Results or facts those are true once the script has terminated.
- Props or the “things” that make up the content of the script.
- Roles are the actions that the individual participants perform.
- Scenes which present temporal aspects of the script.

Scripts: an example

Script Restaurant

Props
- •Tables
- •Menu
- •F = Food
- •Check
- •Money

Roles
- •P = Customer
- •O = Waiter
- •V = Cook
- •K = Cashier
- •S = Owner

Entry conditions
- •P is hungry
- •P has money

Results
- •P has less money
- •P is not hungry
- •P is pleased (optional)
- •S has more money

Scene 1: Entering

P PTRANS P into restaurant
P ATTEND eyes to tables
P MBUILD where to sit
P PTRANS P to table
P MOVE P to sitting position

Scene 2: Ordering

(Menu on table)
O brings menu)
P PTRANS menu to P

(S asks for menu)
S MTRANS signal to O
O PTRANS O to table
P MTRANS "need menu" to O
O PTRANS O to menu

O PTRANS O to table
O ATRANS menu to P

P MTRANS food list to P
* P MBUILD choice of F
P MTRANS signal to O
O PTRANS O to table
P MTRANS 'I want F' to O

O PTRANS O to V
O MTRANS (ATRANS F) to V

V MTRANS 'no F' to O
O PTRANS O to P
O MTRANS 'no F' to P
(go back to *) or
(go to Scene 4 at no pay path)

V DO (prepare F script)
to Scene 3

Scene 3: Eating

V ATRANS F to O
O ATRANS F to P
P INGEST F

Option: Return to Scene 2 to order more; otherwise, go to Scene 4

Scene 4: Exiting

P MTRANS to O
(O ATRANS check to P)

O MOVE write check
O PTRANS O to P
O ATRANS check to P
P ATRANS tip to O
P PTRANS P to K
P ATRANS money to K
P PTRANS P to out of restaurant

No pay path

Schank un Abelson, 1977

CS-208:Artificial Intelligence

Topic-19: Constraint Satisfaction Algorithm

Constraint Satisfaction

Many Problems in AI can be viewed as problems of constraint satisfaction in which the goal is to discover some problem state that satisfies a given set of constraints.

Examples:

- Crypt arithmetic Problem
- Map Colouring Problem
- Some real world problem perceptual labeling problem
- Many design task in which design must be created within fixed limits on time, cost and materials
- etc

Constraint Satisfaction Problems

- In general a CSP is a problem composed of a finite set of variables each of which has a finite domain of values and a set of constraints Each constraint is defined over some subset of the original set of variables and restricts the values these variables can simultaneously take The task is to find an assignment of a value for each variable such that the assignments satisfy all the constraints In some problems the goal is to find all such assignments.
- A CSP is called an n-ary CSP when n is the maximum number of distinct variables over which a constraint is specified. For instance a binary CSP has constraints involving two variable only. It is possible that any n-ary CSP can be converted to a binary CSP.

Formal Definition

Formally a binary CSP can be defined to consist of a triple (V,D,R), where

- V is a set $\{V_1, \ldots, V_i, \ldots, V_n\}$ of n variables;

- D is a set $(D_1, \ldots, D_i, \ldots, D_n\}$ of domains such that $\forall i \ 1 \leq i \leq n$ *and* $D_i = \{ v^i_1, \ldots, v^i_j , \ldots, v^i_1 \}$ is the finite set of possible values for V*i*.

- R is a sequence $\{\ldots, R_{ij}, \ldots\}$ of binary constraint relations such that $\forall\ R_{ij} \in R$, where R_{ij} constrains the two variables V_i and V_j and is defined by a subset of the Cartesian product $D_i \times D_j$ .The set of pairs of values in R specifies the allowed pairs of values for variables V_i and V_j . If $(v^i_l, v^j_m) \in R_{ij}$ we say that the assignment $\{V_i \leftarrow v^i_l , V_j \leftarrow v^j_m \}$ is consistent.

A solution to CSP is a complete assignment that satisfies all the constraints

Defining of CSP in AI Context

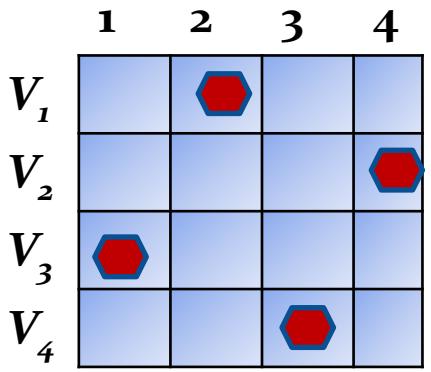
For solving Constraint Satisfaction Problem

- The state is defined by variables V_i with values from domain D_i
- The goal test is a set of constraints specifying allowable combinations of values for subsets of variables.

Example: N-Queens

- **Problem**: *The N-queens problem can be modeled as a CSP. Given an integer N, the problem is to place N queens on N distinct squares in an N×N chess board satisfying the constraint that no two queens should threaten each other. Two queens threaten each other if and only if they are on the same row, column or diagonal.*

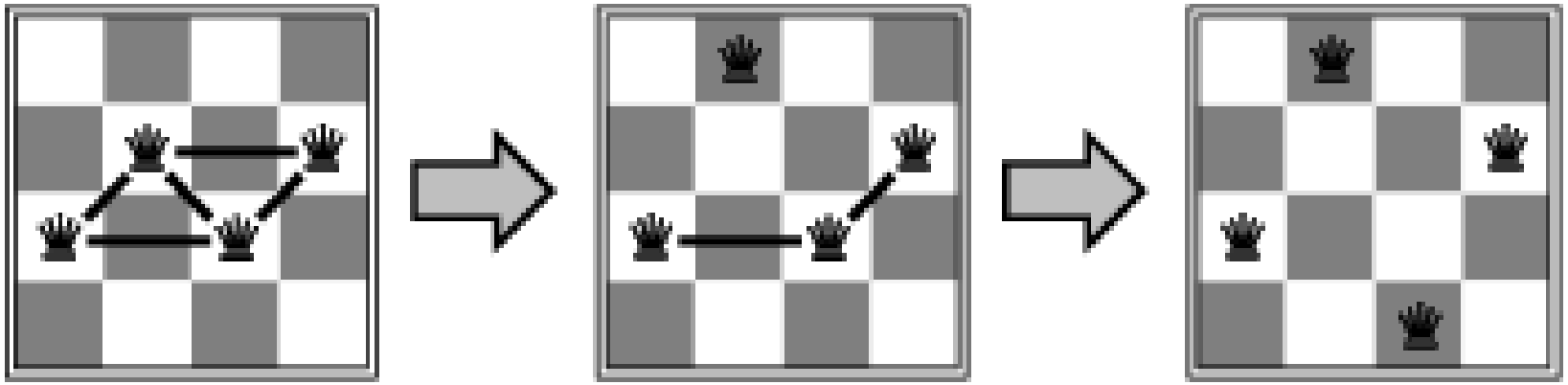
A possible Solution to 4-queens problem



Illustrative Example: 4-Queens

Problem: Place the four queens on a 4 ×4 grid in such a way that no two queens are on the same diagonal, row or column.

- States: 4 queens in 4 columns ($4^4 = 256$ possible states)
- Actions: Move queen in column
- Goal test: No attacks
- Evaluation: $h(n)$ = number of attacks



Encoding of 4-queens problem as a CSP

- Make each row a variable $V = \{V_1, V_2, V_3, V_4\}$. The value of each variable will represent column in which the queen is row$_i$ $(1 \leq i \leq 4)$ is placed;
- Domain: $D = \{D_1, D_2, D_3, D_4\}$. Each of the four variables can take one the four columns as its value with label 1 to 4 . Therefore the domains of the four variables are $D_1 = D_2 = D_3 = D_4 = \{1, 2, 3, 4\}$;
- Constraints: R = { R_{ij} | i < j and $(1 \leq i \leq 4)$ }. For each constraint R_{ij}
- No two queens on the same row: This constraints become trivial given the variable encoding.
- No two queens on the same column: $V_i \neq V_j$
- No two queens on the same diagonal: $| i - j | \neq |V_i - V_j|$

Standard Approaches for CSP Solving

There are three standard approaches for solving CSP

- **Tree Search:** It is a Standard technique for solving CSPs. The basic algorithm is simple backtracking(BT), a general strategy which has been widely used in problem solving. It also serves as the basis for many other algorithm.
- **Constraint Propagation**: It is aimed at transforming a CSP into an equivalent problem that is easier to solve. Constraint propagation works by reducing the domain size of the variable in such a manner that no solution are ruled out. It involves removing redundant values from the domain of the variable and propagating the effect of these removal throughout the constraint.
- **Combined Approach of Backtracking and Constraint Propagation**.

Backtracking (BT) Algorithm

- In BT variables are instantiated one by one.
- When a variable is instantiated a value from its domain is picked and assigned to it. Then constraint checks are performed to make sure that the new instantiation is compatible with all the instantiations made so far.
    - If all the completed constraints are satisfied, this variable has been instantiated successfully and we can move on to instantiate the next variable.
    - If it violates certain constraints then an alternative value when available is picked.
- If all the variables have a value, noting that all the assignments are consistent, then the problem is solved.
- If at any stage no value can be assigned to a variable without violating a constraint, backtracking occurs. That means the most recent instantiated variable has its instantiation revised and a new value if available is assigned to it. At this point we continue on to try instantiating the other variables or we backtrack farther. This carries on until either a solution is found or all the combinations of instantiation have been tried and have failed which means that there is no solution.

```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

Pseudo Code for BT Algorithm-PART 1

$$function \quad CONSISTENT\left(V_i, v_l^i\right)$$

$$for \quad each\left(V_j, v_m^j\right) \in solution$$

$$if \quad R_{ij} \in R \quad and \quad \left(v_l^i, v_m^j\right) \notin R_{ij}$$

$$return \quad FALSE$$

$$return \quad TRUE$$

Pseudo Code for BT Algorithm-PART 2

$$
\begin{aligned}
&function \quad Search\_BT(Vars, Level) \\
&\quad select \quad a \quad variable \quad V_i \in Vars \\
&\quad for \quad each \quad value \quad v_l^i \in D_i \\
&\quad\quad if \quad CONSISTENT(V_i, v_l^i) \\
&\quad\quad\quad Solution \leftarrow Solution + (V_i, v_l^i) \\
&\quad\quad\quad if \quad V_i \quad is \quad the \quad only \quad variable \quad in \quad Vars \\
&\quad\quad\quad\quad return \quad TRUE \\
&\quad\quad\quad else \\
&\quad\quad\quad\quad if \quad SEARCH\_BT(Vars \setminus \{V_i, Level + 1) \\
&\quad\quad\quad\quad\quad return \quad TRUE \\
&return \quad FALSE
\end{aligned}
$$

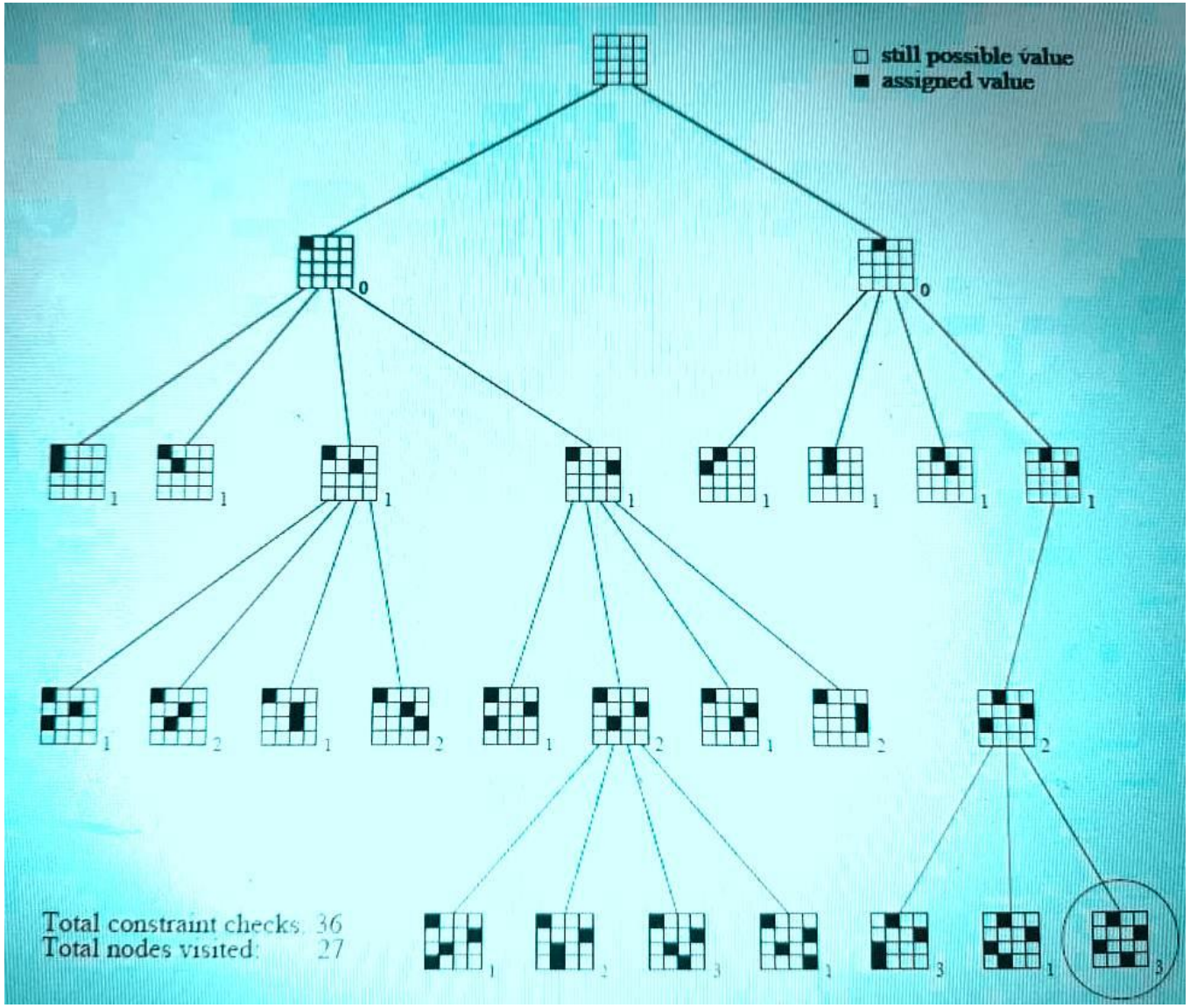
Pseudo Code for BT Algorithm-PART 3

$$function \quad BT$$

$$Solution \quad \leftarrow \quad 0$$

$$return \quad SEARCH\_BT(V,1)$$

Search Tree of the 4-queens problem using BT



CS-208:Artificial Intelligence

Topic-19A: Means-Ends Analysis

An Introduction: Means-Ends Analysis

- Means-Ends Analysis (MEA) is problem-solving techniques used in Artificial intelligence for limiting search in AI problem solving.
- It is a mixture of Backward and Forward Search Techniques.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon.

  Typically, a planner needs to find a correct set of actions (a plan) that will take it from one state to another state say goal state. One approach is to consider the differences between goal state and the current state and select the actions that aim to lessen those differences – this is called means- end analysis.
- The MEA process is centered on the evaluation of the difference between the current state and the goal state.

The Major Steps involved in the Means-Ends Analysis

1. Until the goal is reached or no more procedures are available:
   - Evaluate the differences between Current State and Goal State.
     - Select the various operators which can be applied for each difference.
     - Apply the operator at each difference, which reduces the difference between the current state and goal state .
1. If goal is reached then **success** otherwise **failure**.

Description of the Means-End Analysis process

- The means-ends analysis process centers round the detection of difference between the current state and the goal state.
- Once the difference is isolated, an operator that can reduce the difference must be found
- If the operator cannot be applied to the current state then we set up a sub-problem of getting to a state in which it can be applied (this kind of backward chaining in which the operator are selected and then sub-goals are set up to establish the precondition of the operator is called operator subgoaling),

- If the operator does not produce exactly the goal state, then have a second sub-problem of getting the state it does produce the goal state
- If the difference was chosen correctly and the operator is really effective at reducing the difference then the two sub-problems should be esier to solve the original problem.
- The means-ends analysis can be applied recursively..
- For effectively solving big problem, first the differences can be assigned with priority levels and then the differences of higher priority can be considered before lower priority ones.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the sub-problem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Means-Ends Analysis Algorithm

Function **MEA**(CURRENT, GOAL) # Here the Current state is named as CURRENT and Goal State as GOAL

Compare CURRENT to GOAL

if *there are no differences between CURRENT and GOAL* **then**

return Success

Exit.

else

select the most significant difference and reduce it by doing the following steps **until** the success or failure is **signaled**

Select a new operator O which is applicable for the current difference,

if there is no such operator, **then**

signal Failure.

else

Attempt to apply operator O to CURRENT.

Make a description of two states.

O-Start # a state in which O?s preconditions are satisfied.

O-Result # the state that would result if O were applied In O-start.

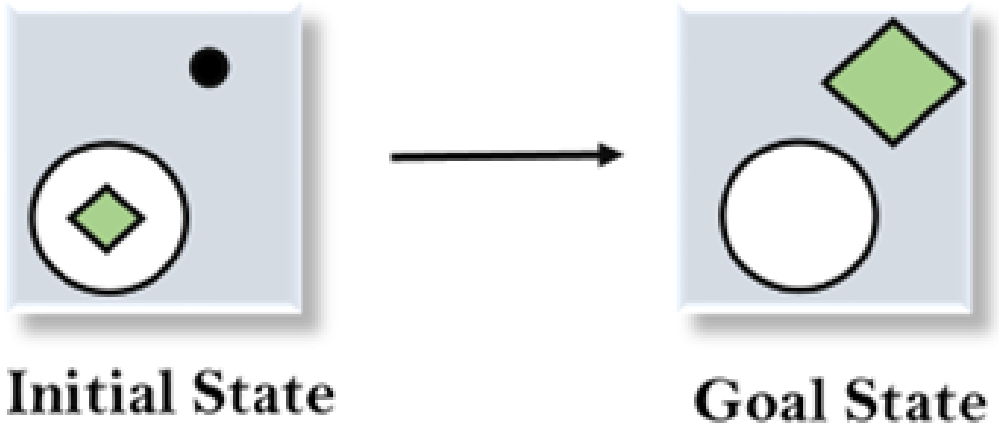
If (*First-Part* ← **MEA** *(CURRENT, O-START*) and (*LAST-Part* ← **MEA** *(O-Result, GOAL*) are successful, then

signal Success

return the result of combining FIRST-PART, O, and LAST-PART.

Illustrative Example

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.

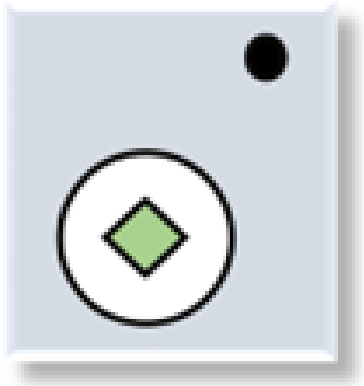


Solution: Step1

To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move, Delete and Expand**

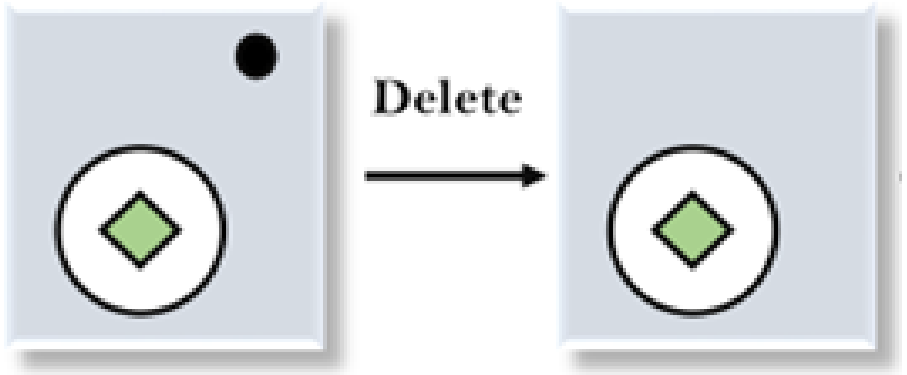
Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



Initial state

Solution: Step2

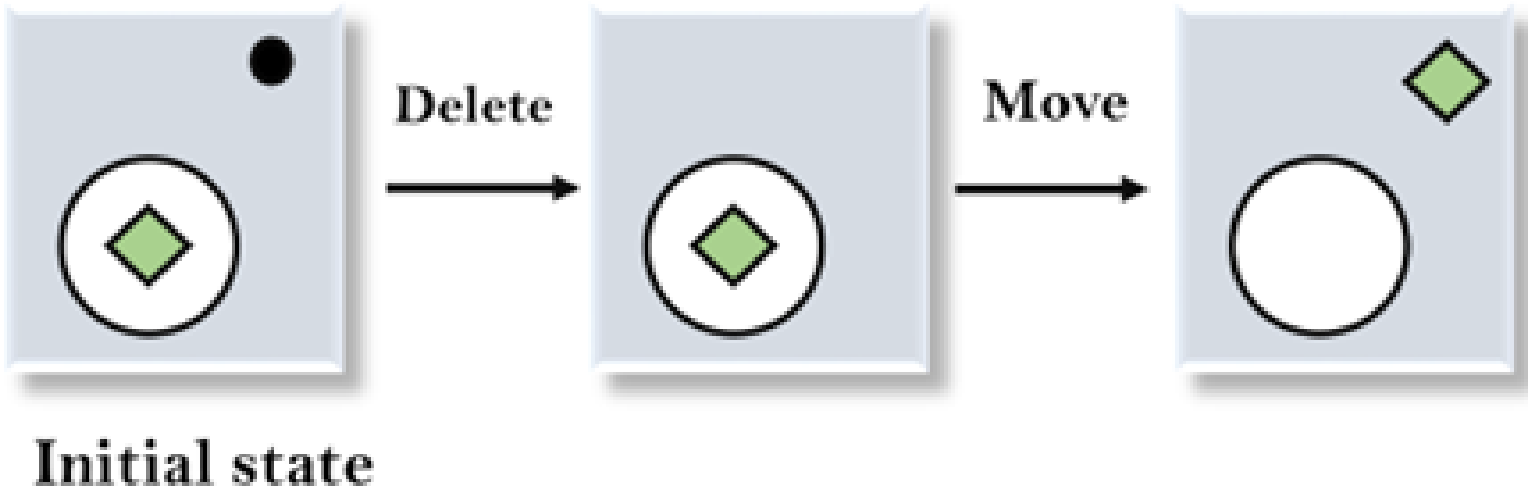
2. **Applying Delete operator:** As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



Initial state

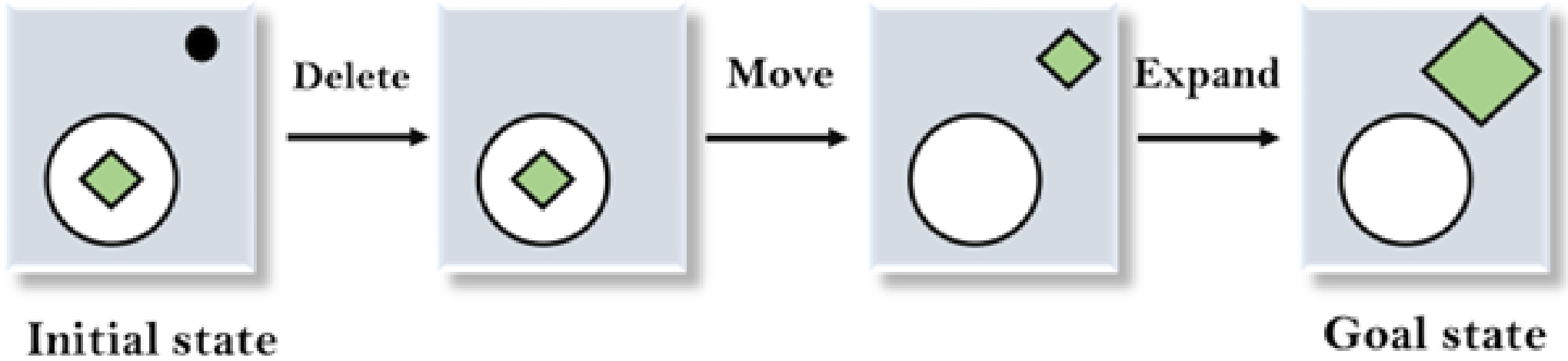
Solution: Step3

Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



Solution: Step4

Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



CS-208:Artificial Intelligence

Topic-20: Natural Language Processing &Expert System

Natural Language Processing

Natural Language Processing (NLP) refers to AI method of communicating with an intelligent system using a natural language such as English. The field of NLP involves making computers to perform useful tasks with the natural languages humans use. The input and output of an NLP system can be either *Speech* or *Written Text*

Two components of NLP

- **Natural Language Understanding (NLU)** involves mapping the given input in natural language into useful representations and analyzing different aspects of the language.
- **Natural Language Generation (NLG)** is the process of producing meaningful phrases and sentences in the form of natural language from some internal representation and involves
    - (i) Text planning: It includes retrieving the relevant content from knowledge base.
    - (ii) Sentence planning: It includes choosing required words, forming meaningful phrases, setting tone of the sentence and
    - (iii) Text Realization − It is mapping sentence plan into sentence structure.

Difficulties in NLU

The NLU is harder than NLG. NL has an extremely rich form and structure. It is very ambiguous. There can be different levels of ambiguity −

- **Lexical ambiguity:** It is at very primitive level such as word-level.

  For example, treating the word “board” as noun or verb?

- **Syntax Level ambiguity:** A sentence can be parsed in different ways.

  For example, “He lifted the beetle with red cap.” − Did he use cap to lift the beetle or he lifted a beetle that had red cap?

- **Referential ambiguity** : Referring to something using pronouns.

  For example, Rima went to Gauri. She said, “I am tired.” − Exactly who is tired?

One input can mean different meanings.

Many inputs can mean the same thing.

NLP Terminology

Phonology : It is study of organizing sound systematically.

Morphology: It is a study of construction of words from primitive meaningful units.

Morpheme : It is primitive unit of meaning in a language.

Syntax : It refers to arranging words to make a sentence. It also involves determining the structural role of words in the sentence and in phrases.

Semantics: It is concerned with the meaning of words and how to combine words into meaningful phrases and sentences.

Pragmatics: It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.

Discourse: It deals with how the immediately preceding sentence can affect the interpretation of the next sentence.

World Knowledge: It includes the general knowledge about the world.

STEPS in NLP

Lexical Analysis: It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.

Syntactic Analysis (Parsing): It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as “The school goes to boy” is rejected by English syntactic analyzer.

Semantic Analysis: It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as “hot ice-cream”.

Discourse Integration: The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.

Pragmatic Analysis: During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

Expert systems

The expert systems are the computer programs developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise. Expert system can solve problem that typically requires an expert knowledge.

Like human experts, it should provide two basic services

- The system should be able to provide realistic solution from relatively incomplete data.
- The system should be able to explain how and why it arrived at a particular solution.

Capabilities of Expert Systems

- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
- Suggesting alternative options to a problem

Incapability of Expert Systems

- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Components of Expert Systems

- Knowledge Base
- Interface Engine
- User Interface

Knowledge Base

It contains domain-specific and high-quality knowledge. Knowledge is required to exhibit intelligence. The success of any ES mainly depends upon the collection of highly accurate and precise knowledge. Knowledge is collection of facts. The information is organized as data and facts about the task domain. Data, information, and past experience combined together are termed as knowledge.

- **Components of Knowledge Base:** The knowledge base of an ES is a store of both, factual and heuristic knowledge. Factual Knowledge is the information widely accepted by the Knowledge Engineers and scholars in the task domain. Heuristic Knowledge is about practice, accurate judgment, one's ability of evaluation, and guessing.
- **Knowledge Acquisition***:* The success of any expert system mainly depends on the quality, completeness, and accuracy of the information stored in the knowledge base. The knowledge base is formed by readings from various experts, scholars, and the Knowledge Engineers. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills. He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Interface Engine

Use of efficient procedures and rules by the Interface Engine is essential in deducting a correct, flawless solution. In case of knowledge-based ES, the Interface Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution. In case of rule based ES, it applies rules repeatedly to the facts, which are obtained from earlier rule application and resolves rules conflict when multiple rules are applicable to a particular case. The Interface Engine Strategies are:

- **Forward Chaining:** It is a strategy of an expert system to answer the question, “What can happen next?” Here, the interface engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution. This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.
- **Backward Chaining:** With this strategy, an expert system finds out the answer to the question, “Why this happened?” On the basis of what has already happened, the interface engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason, for example, diagnosis of blood cancer in humans.

User Interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence. It explains how the ES has arrived at a particular recommendation. The explanation may appear in the form of Natural language displayed on screen.

- Verbal narrations in natural language.
- Listing of rule numbers displayed on the screen.
- The user interface makes it easy to trace the credibility of the deductions. Requirements of Efficient ES User Interface
- It should help users to accomplish their goals in shortest possible way.
- It should be designed to work for user's existing or desired work practices.
- Its technology has following forms:
  - should be adaptable to user's requirements; not the other way round.
  - It should make efficient use of user input.

Applications of Expert System

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.