CS8691 – ARTIFICIAL INTELLIGENCE

QUESTION BANK

As per anna university regulation 2017

Unit-1:

Two-mark questions:

1. Define A.I or what is A.I?   **May03,04**

   Artificial intelligence is the branch of computer science that deals with the automation of intelligent behavior. AI gives basis for developing human like programs which can be useful to solve real life problems and thereby become useful to mankind.

2. What is meant by robotic agent?   **May 05**

   A machine that looks like a human being and performs various complex acts of a human being. It can do the task efficiently and repeatedly without fault. It works on the basis of a program feeder to it; it can have previously stored knowledge from environment through its sensors. It acts with the help of actuators.

3. Define an agent?   **May 03,Dec-09**

   An agent is anything (a program, a machine assembly) that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

4. Define rational agent?   **Dec-05,11, May-10**

   A rational agent is one that does the right thing. Here right thing is one that will cause agent to be more successful. That leaves us with the problem of deciding how and when to evaluate the agent's success.

5. Give the general model of learning agent?   **Dec-03**

   Learning agent model has 4 components –
   1) Learning element.
   2) Performance element.
   3) Critic
   4) Problem Generator.

6. What is the role of agent program?   **May-04**

   Agent program is important and central part of agent system. It drives the agent, which means that it analyzes date and provides probable actions agent could take.

   An agent program is internally implemented as agent function.

   An agent program takes input as the current percept from the sensor and returns an action to the effectors.

7. List down the characteristics of intelligent agent?   **May-11**

   The IA must learn and improve through interaction with the environment.
   The IA must adapt online and in the real time situation.
   The IA must accommodate new problem-solving rules incrementally.
   The IA must have memory which must exhibit storage and retrieval capabilities.

8. Define abstraction?   **May-12**

   In AI the abstraction is commonly used to account for the use of various levels in detail in a given representation language or the ability to change from one level to another level to another while preserving useful properties. Abstraction has been mainly studied in

9   State the concept of rationality?   **May-12**

Rationality is the capacity to generate maximally successful behavior given the available information. Rationality also indicates the capacity to compute the perfectly rational decision given the initially available information. The capacity to select the optimal combination of computation – sequence plus the action, under the constraint that the action must be selected by the computation is also rationality.

Perfect rationality constraints an agent's actions to provide the maximum expectations of success given the information available.

10   What are the functionalities of the agent function?   **Dec-12**

Agent function is a mathematical function which maps each and every possible percept sequence to a possible action.

The major functionality of the agent function is to generate the possible action to each and every percept. It helps the agent to get the list of possible actions the agent can take. Agent function can be represented in the tabular form.

11   Define basic agent program?   **May-13**

The basic agent program is a concrete implementation of the agent function which runs on the agent architecture. Agent program puts bound on the length of percent sequence and considers only required percept sequences. Agent program implements the functions of percept sequence and action which are external characteristics of the agent.

Agent program takes input as the current percept from the sensor and return an action to the effectors (Actuators)

12   what are the four components to define a problem? Define them. **May-13**

1. initial state: state in which agent starts in.
2. A description of possible actions: description of possible actions which are available to the agent.
3. The goal test: it is the test that determines whether a given state is goal state.
4. A path cost function: it is the function that assigns a numeric cost (value) to each path. The problem-solving agent is expected to choose a cost function that reflects its own performance measure.

16 Mark questions:

1.   **Explain properties of environment.**   **Dec -2009**

Properties of Environment

The environment has multifold properties –

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

   o   If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.

- o A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- o An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- o If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- o A stochastic environment is random in nature and cannot be determined completely by an agent.
- o In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- o In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- o However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- o If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- o However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- o The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- o If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- o Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- o However for dynamic environment, agents need to keep looking at the world at each action.
- o Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- o If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- o A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- o A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- o Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- o In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- o It is quite possible that a known environment to be partially observable and an

8. Accessible vs Inaccessible

- o If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- o An empty room whose state can be defined by its temperature is an example of an accessible environment.
- o Information about an event on earth is an example of Inaccessible environment.

2. **Explain in detail, the structure of different intelligent agents. Dec -2012, May-2012**

TYPES OF INTELLIGENT AGENT
The Structure of Intelligent Agents

Agent's structure can be viewed as −

- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.
- Agent Program = an implementation of an agent function.

Different forms of Agent

As the degree of perceived intelligence and capability varies to frame into four categories as,
A.   Simple Reflex Agents
B.   Model Based Reflex Agents
C.   Goal Based Agents
D.   Utility Based agents
(A)Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.

Condition-Action Rule − It is a rule that maps a state (condition) to an action.
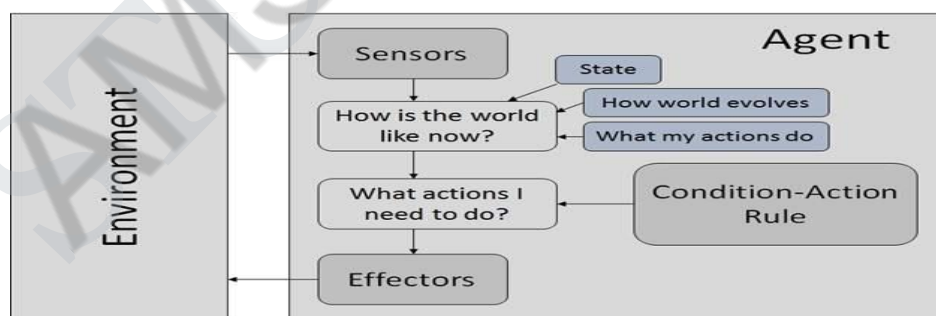

```
Procedure : SIMPLE - REFLEX - AGENT
Input : Percept
Output : An action.
Static : Rules, a set of condition - action rules.
1.      State ← INTERPRET - INPUT (percept)
2.      rule ← RULE - MATCH (state, rules)
3.      action ← RULE - ACTION (rule)
4.      return action.
```
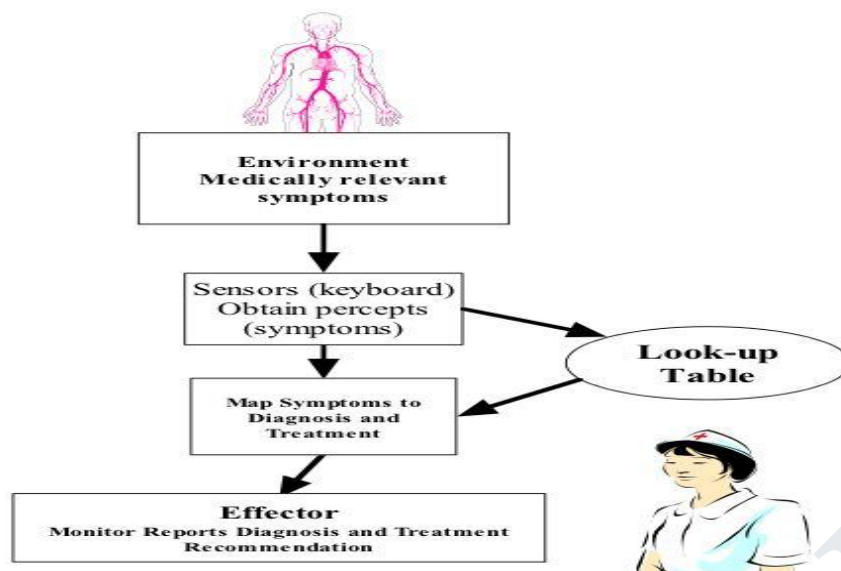
Example1: ATM system if PIN matches with given account number than customer get

Example2:



(B) Model Based Reflex Agents

They use a model of the world to choose their actions. They maintain an internal state.

Model − The knowledge about how the things happen in the world.

Internal State − It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about −

* How the world evolves.
* How the agent's actions affect the world.
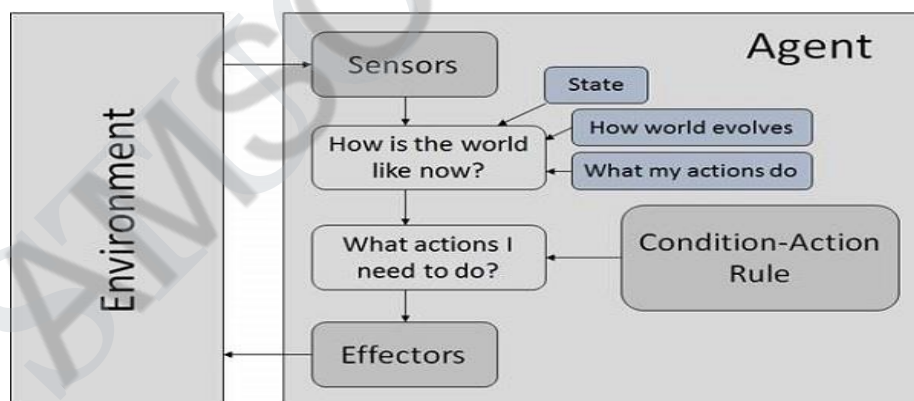

```
Procedure : REFLEX-AGENT-WITH-STATE

Input : Percept

Output : An action.

Static : State, a description of the current world state, rules, a set of condition-
         action rules, action, the most recent action, initially none.

1. State ← UPDATE-STATE (state, action, percept)
2. Rule ← RULE-MATCH (state, rules)
3. Action ← RULE-ACTION (rule)
4. return action.
```

Example: Car driving agent which maintains its own internal state and then take action as environment appears to it.

Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

Goal − It is the description of desirable situations.
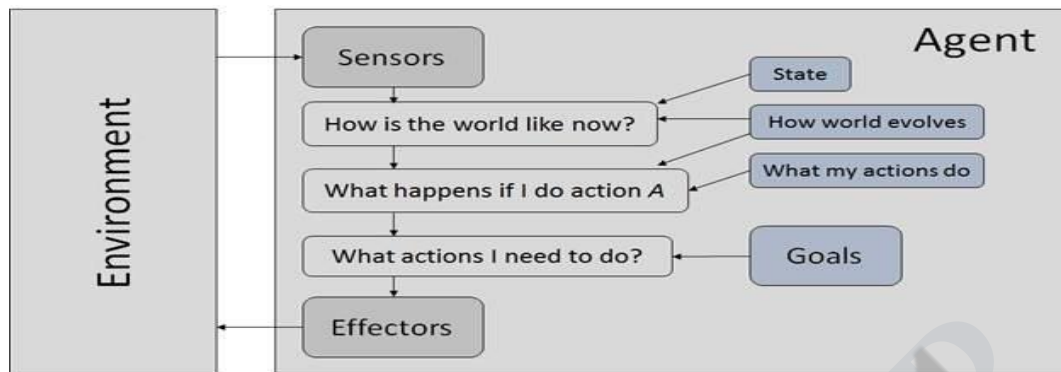

Example: **Searching solution for 8-queen puzzle**.

Utility Based Agents

They choose actions based on a preference (utility) for each state. Goals are inadequate when −

- There are conflicting goals, out of which only few can be achieved.

- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.

- Example: Millitary planning robot which provides certain plan of action to be taken.
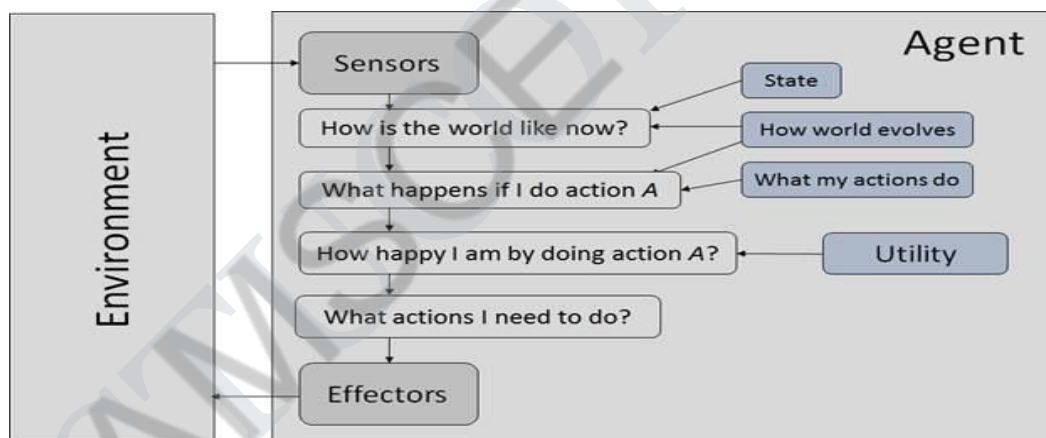
3. **What is an agent? Explain the basic kinds of agent program.**   **Dec -2014**

AGENT

Introduction

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

An **agent** is anything that can perceive its **environment** through **sensors** and acts upon that environment through **actuators.**

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.

- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.

- A **software** agent has encoded bit strings as its programs and actions.
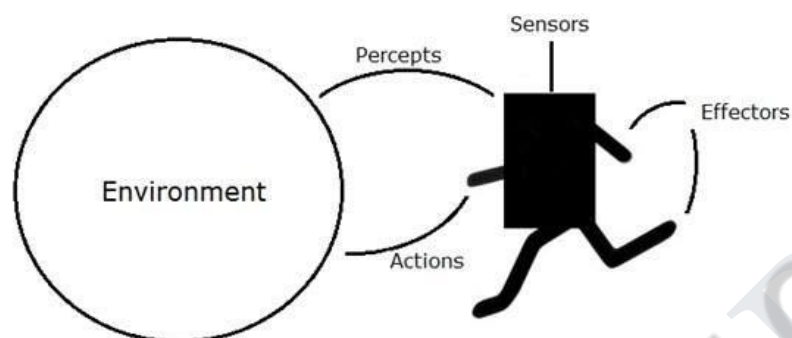

Agent Terminology

- **Performance Measure of Agent** − It is the criteria, which determines how successful an agent is.

- **Behavior of Agent** − It is the action that agent performs after any given sequence of percepts.

- **Percept** − It is agent's perceptual inputs at a given instance.

- **Percept Sequence** − It is the history of all that an agent has perceived till date.

- **Agent Function** − It is a map from the precept sequence to an action.
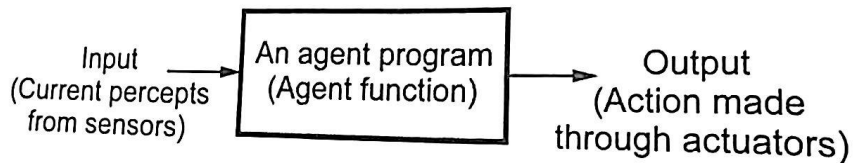
AI Agent Action Process:

Following diagram illustrated the agent action process, a specified by architecture.
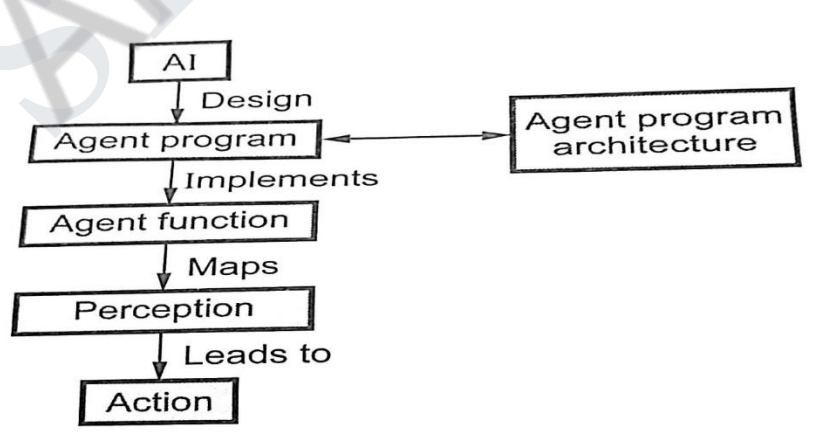
Role of an Agent Program

- An agent program is internally implemented as agent function.
- An agent program take input as the current percept from the sensor and return an action to

ctuato   rs)

Fig. 2.5.1

Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

4. **How a problem is formally defined? List down the components of it? Dec 14,May 14, Dec 10**

Problems are the issues which comes across any system. A solution is needed to solve that particular problem.
The process of solving a problem consists of five steps. These are:
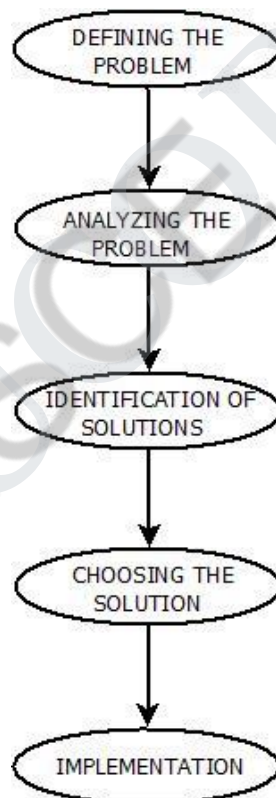
Problem Solving in Artificial Intelligence

1. Defining The Problem: The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.
2. Analyzing The Problem: Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.
3. Identification Of Solutions: This phase generates reasonable amount of solutions to the given problem in a particular range.
4. Choosing a Solution: From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.
tion: After choosing the best solution, its implementation is done.

Components of Planning System

The important components of planning are

- Choosing the best rule to apply next based on the best variable available heuristic information.
- Applying the chosen rule to compute the new problem state that arises from its application.
- Detecting when a solution has been found.
   Detecting dead ends so that they can be abandoned and the system's effort directed in correct direction.Repairing an almost correct solution.

1.Choosing rules to apply:

First isolate a set of differences between the desired goal state and the current state.
Detect rules that are relevant to reduce the differences.

If several rules are found, a variety of heuristic information can be exploited to choose among them.This technique is based on the means end analysis method.

2. Applying rules:

Applying the rules is easy.

Each rule specifies the problem state that would result from its application.

We must be able to deal with rules that specify only a small part of the complete problem state.Different ways to do this are Describe, for each action, each of the changes it makes the state description.A state was described by a set of predicates representing the facts that were true in that state. Each state is represented as a predicate.The manipulation of the state description is done using a resolution theorem prover.

3. Detecting a solution

A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.Any of the corresponding reasoning mechanisms could be used to discover when a solution has been found.

4. Detecting dead ends

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.

If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.

If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or little progress is being made.

In reasoning backward, each goal is decomposed into sub goals. Each of them may lead to a set of additional sub goals. Sometimes it is easy to detect that there is now way that the entire sub goals in a given set can be satisfied at once. Other paths can be pruned because they lead nowhere.

5. Repairing an almost correct solution

Solve the sub problems separately and then combine the solution to yield a correct solution. But it leads to wasted effort.

The other way is to look at the situations that result when the sequence of operations corresponding to the proposed solution is executed and to compare that situation to the desired goal. The difference between the initial state and goal state is small. Now the problem solving can be called again and asked to find a way of eliminating a new difference. The first solution can then be combined with second one to form a solution to the original problem.

When information as possible is available, complete the specification in such a way that no conflicts arise. This approach is called least commitment strategy. It can be applied in a variety of ways.

- To defer deciding on the order in which operations can be performed.

- Choose one order in which to satisfy a set of preconditions, we could leave the order unspecified until the very end. Then we could look at the effects of each of the sub solutions to determine the dependencies that exist among them. At that point, an ordering can be chosen.

Unit-2:

Two-mark questions:

1. How will you measure the problem-solving performance?   **May 10**
   Problem solving performance is measured with 4 factors.
   1) Completeness - Does the algorithm (solving procedure) surely finds solution if really the solution exists.
   2) Optimality – If multiple solutions exits then do the algorithm returns optimal amongst them.
   3) Time requirement.
   4) Space requirement.

2. What is the application of BFS?   **May 10**
   It is simple search strategy, which is complete i.e. it surely gives solution if solution exists. If the depth of search tree is small then BFS is the best choice. It is useful in tree as well as in graph search.

3. State on which basis search algorithms are chosen?   **Dec 09**
   Search algorithms are chosen depending on two components.
   1) How is the state space – That is, state space is tree structured or graph?
      Critical factor for state space is what is branching factor and depth level of that tree or graph.
   2) What is the performance of the search strategy? A complete, optimal search strategy with

4. Evaluate performance of problem-solving method based on depth-first search algorithm?

Dec 10

DFS algorithm performance measurement is done with four ways –
1) Completeness – It is complete (guarantees solution)
2) Optimality – it is not optimal.
3) Time complexity – It's time complexity is O (b).
4) Space complexity – its space complexity is O (b d+1).

5. What are the four components to define a problem? Define them? **May 13**
The four components to define a problem are,
1) Initial state – it is the state in which agent starts in.
2) A description of possible actions – it is the description of possible actions which are available to the agent.
3) The goal test – it is the test that determines whether a given state is goal (final) state.
4) A path cost function – it is the function that assigns a numeric cost (value) to each path. The problem-solving agent is expected to choose a cost function that reflects its own performance measure.

6. State on what basis search algorithms are chosen? **Dec 09**
Refer Question 3

7. Define the bi-directed search? **Dec 13**
As the name suggests bi-directional that is two directional searches are made in this searching technique. One is the forward search which starts from initial state and the other is the backward search which starts from goal state. The two searches stop when both the search meet in the middle.

8. List the criteria to measure the performance of search strategies? **May 14**
Refer Question 3

9. Why problem formulation must follow goal formulation? **May 15**
Goal based agent is the one which solves the problem. Therefore, while formulating problem one need to only consider what is the goal to be achieved so that problem formulation is done accordingly. Hence problem formulation must follow goal formulation.

10. mention how the search strategies are evaluated?
Search strategies are evaluated on following four criteria
1. completeness: the search strategy always finds a solution, if one exits?
2. Time complexity: how much time the search strategy takes to complete?
3. Space complexity: how much memory consumption search strategy has?
4. Optimality: the search strategy finds a highest solution.

11. define admissible and consistent heuristics?
Admissible heuristics: a heuristic is admissible if the estimated cost is never more than actual cost from the current node to the goal node.
Consistent heuristics:
A heuristic is consistent if the cost from the current node to a successor node plus the estimated cost from the successor node to the goal is less than or equal to estimated cost from the current node to the goal.

12. What is the use of online search agent in unknown environment? **May-15**
Ans: Refer Question 6

13. list some of the uninformed search techniques?

The uninformed search strategies are those that do not take into account the location of the goal. That is these algorithms ignore where they are going until they find a goal and report success. The three most widely used uninformed search strategies are

1.depth-first search-it expands the deepest unexpanded node

2.breadth-first search-it expands shallowest unexpanded node

3.lowest -cost-first search (uniform cost search)- it expands the lowest cost node

14. When is the class of problem said to be intractable? **Dec– 03**

The problems whose algorithm takes an unreasonably large amount of resources (time and / or space) are called intractable.

For example – TSP

Given set of 'N' points, one should find shortest tour which connects all of them. Algorithm will consider all N! Orderings, i.e. consider $n = 16 \therefore 16! > 2^{50}$ which is impractical for any computer?

15. What is the power of heuristic search? or **Dec-04**

Why does one go for heuristics search? **Dec-05**

Heuristic search uses problem specific knowledge while searching in state space. This helps to improve average search performance. They use evaluation functions which denote relative desirability (goodness) of a expanding node set. This makes the search more efficient and faster. One should go for heuristic search because it has power to solve large, hard problems in affordable times.

16. What are the advantages of heuristic function? **Dec- 09,11**

Heuristics function ranks alternative paths in various search algorithms, at each branching step, based on the available information, so that a better path is chosen.

The main advantage of heuristic function is that it guides for which state to explore now, while searching. It makes use of problem specific knowledge like constraints to check the goodness of a state to be explained. This drastically reduces the required searching time.

17. State the reason when hill climbing often gets stuck? **May -10**

Local maxima are the state where hill climbing algorithm is sure to get struck.

Local maxima are the peak that is higher than each of its neighbor states, but lower than the global maximum. So we have missed the better state here. All the search procedure turns out to be wasted here. It is like a dead end.

18. When a heuristic function h is said to be admissible? Give an admissible heuristic function for TSP? **Dec-10**

Admissible heuristic function is that function which never over estimates the cost to reach the goal state. It means that h(n) gives true cost to reach the goal state 'n'.

The admissible heuristic for TSP is

a. Minimum spanning tree.

b. Minimum assignment problem.

19. What do you mean by local maxima with respect to search technique? **May -11**

Local maximum is the peak that is higher than each of its neighbor states, but lowers than the global maximum i.e. a local maximum is a tiny hill on the surface whose peak is not as high as the main peak (which is a optimal solution). Hill climbing fails to find optimum solution when it encounters local maxima. Any small move, from here also makes things worse (temporarily). At local maxima all the search procedure turns out to be wasted here. It is like a dead end.

20. How can we avoid ridge and plateau in hill climbing? **Dec-12**

Ridge and plateau in hill climbing can be avoided using methods like backtracking,

big jumps. Backtracking and making big jumps help to avoid plateau,

21. What is CSP?   **May-10**

CSP are problems whose state and goal test conform to a standard structure and very simple representation. CSPs are defined using set of variables and a set of constraints on those variables. The variables have some allowed values from specified domain. For example – Graph coloring problem.

22. How can minimax also be extended for game of chance?

Dec-10

In a game of chance, we can add extra level of chance nodes in game search tree. These nodes have successors which are the outcomes of random element.

The minimax algorithm uses probability P attached with chance node d_i based on this value. Successor function $S(N,d_i)$ give moves from position N for outcome d_i

16-mark questions:

1. Discuss any 2 uninformed search methods with examples.   **Dec 09,Dec 14,May-13,May-17**

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

Step 3: Traverse all neighbours of neighbours of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: Place the root node inside the queue.

Step 2: If the queue is empty then stops and return failure.

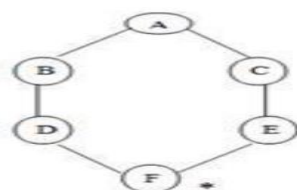Step 3: If the FRONT node of the queue is a goal node then stop and return success.

Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in readystate then place them inside the queue in any order.

Step 5: Go to Step 3.

Step 6: Exit.

Implementation:

Let us implement the above algorithm of BFS by taking the following suitable example.

node (*)

Step 1: Place the root node inside the queue i.e. A

Step 2: Now the queue is not empty and also the FRONT node i.e. A is not our goal node. So move to step 3.

Step 3: So remove the FRONT node from the queue i.e. A and find the neighbour of A i.e. B and C B C A

Step 4: Now b is the FRONT node of the queue .So process B and finds the neighbours of B i.e. D. C D B

Step 5: Now find out the neighbours of C i.e. E

D B E

Step 6: Next find out the neighbours of D as D is the FRONT node of the queue

E F D

Step 7: Now E is the front node of the queue. So the neighbour of E is F which is our goal node. F E

Step 8: Finally F is our goal node which is the FRONT of the queue. So exit. F

Advantages:

☐ In this procedure at any way it will find the goal.

☐ It does not follow a single unfruitful path for a long time. It finds the minimal solution in case of multiple paths.

Disadvantages:

☐ BFS consumes large memory space. Its time complexity is more.

☐ It has long pathways, when all paths to a destination are on approximately the same search depth.

Depth First Search (DFS)

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or last in

first out (LIFO) is used for DFS. One interesting property of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

Step 3: Traverse any neighbour of neighbour of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the starting node into the stack.

Step 2: If the stack is empty then stop and return failure.

Step 3: If the top node of the stack is the goal node, then stop and return success.
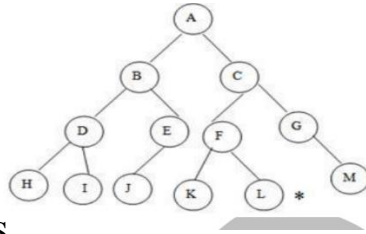
Step 4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready stateand PUSH them into the stack in any order.

Step 5: Go to step 3.

Step 6: Exit.

Implementation:

Let us take an example for implementing the above DFS algorithm.

Examples of DFS

Consider A as the root node and L as the goal node in the graph figure

Step 1: PUSH the starting node into the stack i.e.A

Step 2: Now the stack is not empty and A is not our goal node. Hence move to next step.

Step 3: POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.

B C A

Step 4: Now C is top node of the stack. Find its neighbours i.e. F and G.

B F G C

Step 5: Now G is the top node of the stack. Find its neighbour i.e. M

B F M G

Step 6: Now M is the top node and find its neighbour, but there is no neighbours of M in the graph soPOP it from the stack. B F M

Step 7: Now F is the top node and its neighbours are K and L. so PUSH them on to the stack. B K L F

Step 8: Now L is the top node of the stack, which is our goal node.

B K L

Advantages:

☐ DFSconsumes very less memory space.

☐ It will reach at the goal node in a less time period than BFS if it traverses in a right path.

☐ It may find a solution without examining much of search because we may get the desired solution in the very first go

Disadvantages:

☐ It is possible that may states keep reoccurring. There is no guarantee of finding the goal node.

☐ Sometimes the states may also enter into infinite loops

2. **Explain the A* search and give the proof of optimality of A***   May-10,Dec-09,May-13

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search. In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula

.f (n) + g (n) = h (n)

Where g (n): The actual cost path from the start state to the current state.

h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and

OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

The implementation of A* algorithm is 8-puzzle game.

Advantages:

☐ It is complete and optimal.

☐ It is the best one from other techniques. It is used to solve very complex problems.

☐ It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

☐ This algorithm is complete if the branching factor is finite and every action has fixed cost.

☐ The speed execution of A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h (n).It has complexity problems.

3. **Explain AO* algorithm with a suitable example. State the limitations in the algorithm?**

AO* Search: (And-Or) Graph   **Dec-11,12**

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved.If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable,then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

4. **Explain the nature of heuristics with example. What is the effect of heuristics accuracy?May-13,May-16**

We can also call informed search as Heuristics search. It can be classified as below

Best first search

Branch and Bound Search

A* Search

AO* Search

Hill Climbing

Constraint satisfaction

Means end analysis

Heuristic is a technique which makes our search algorithm more efficient. Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it.

Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always. Heuristic search algorithm uses information about the problem to help directing the path through the search space.

These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient. A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number.

The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.

Generally heuristic incorporates domain knowledge to improve efficiency over blind search.

In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case.

For example in case of travelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing.

This technique is useful in solving though problems which could not be solved in any other way. Solutions take an infinite time to compute.

Classifications of heuristic search.

Best first search is an instance of graph search algorithm in which a node is selected for expansion based o evaluation function f (n). Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal. Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values.

This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Concept:

Step 1: Traverse the root node

Step 2: Traverse any neighbor of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.

Step 3: Traverse any neighbor of neighbor of the root node, that is maintaining a least distance fromthe root node and insert them in ascending order into the queue

**Step 4:** This process will continue until we are getting the goal node

Algorithm:

Step 1: Place the starting node or root node into the queue.

Step 2: If the queue is empty, then stop and return failure.

Step 3: If the first element of the queue is our goal node, then stop and return success.

Step 4: Else, remove the first element from the queue. Expand it and compute the estimated goal distancefor each child. Place the children in the queue in ascending order to the goal distance. **Step 5:** Go to step-3

Implementation:
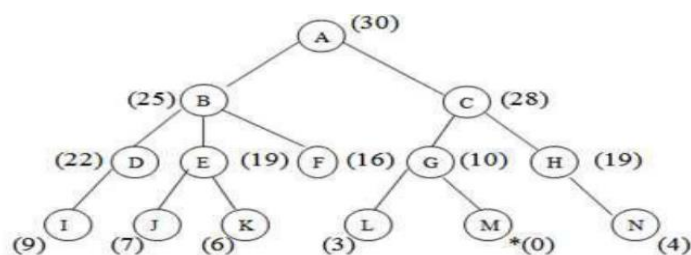
Step 1: Consider the node A as our root node. So the first element of the queue is A whish is not our goal node, so remove it from the queue and find its neighbor that are to inserted in ascending order. A

Step 2: The neighbors of A are B and C. They will be inserted into the queue in ascending order. B C A

Step 3: Now B is on the FRONT end of the queue. So calculate the neighbours of B that are maintaining a least distance from the roof. F E D C B **Step 4:**Now the node F is on the FRONT end of the queue. But as it has no further children, so remove it from the queue and proceed further. E D C B **Step 5:**Now E is the FRONT end. So the children of E are J and K. Insert them into the queue in ascending order.K J D C E

Step 6: Now K is on the FRONT end and as it has no further children, so remove it and proceed further J D C K

Step7: Also, J has no corresponding children. So remove it and proceed further.D C J

Step 8: Now D is on the FRONT end and calculates the children of D and put it into the queue. I C D

Step9: Now I is the FRONT node and it has no children. So proceed further

after removing this node   from the queue.   C I

Step 10: Now C is the FRONT node .So calculate the neighbours of C that are to be inserted in ascending order into the queue.G H C

Step 11: Now remove G from the queue and calculate its neighbour that is to insert in ascending order into the queue. M L H G

Step12: Now M is the FRONT node of the queue which is our goal node. So stop here and exit. L H M

Advantage:

It is more efficient than that of BFS and DFS.

Time complexity of Best first search is much less than Breadth first search.

The Best first search allows us to switch between paths by gaining the benefits of both breadth first and depth first search. Because, depth first is good because a solution can be found without computing all nodes and Breadth first search is good because it does not get trapped in dead ends.

Disadvantages:

Sometimes, it covers more distance than our consideration.

Branch and Bound Search

Branch and Bound is an algorithmic technique which finds the optimal solution by keeping the

nd so   far. If partial solution can't improve on the best it is abandoned, by this

method the number of nodes which are explored can also be reduced. It also deals with the optimization problems over a search that can be presented as the leaves of the search tree.

The usual technique for eliminating the sub trees from the search tree is called pruning. For Branch and Bound algorithm we will use stack data structure.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node that is maintaining least distance from the root node.

Step 3: Traverse any neighbour of the neighbour of the root node that is maintaining least distance fromthe root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the root node into the stack.

Step 2: If stack is empty, then stop and return failure.

Step 3: If the top node of the stack is a goal node, then stop and return success.
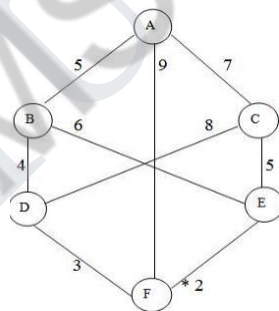
Step 4: Else POP the node from the stack. Process it and find all its

successors. Find   out the   path   containing all its successors as well   as

predecessors and then PUSH the successors which are belonging to the

minimum or shortest path.

Step 5: Go to step 5.

Step 6: Exit.

Implementation:

Let us take the following example for implementing the Branch and Bound algorithm.

Step 1:

Consider the node A as our root node. Find its successors i.e. B, C, F.

Calculate the distance from the root and PUSH them according to least

distance.

A:Stating Node

E. 0+5 = 5 (The cost of A is 0 as it is the starting node) F:0+9 = 9

C:0+7 = 7

Here B (5) is the least distance.

AB

Step 2:

Now the stack will be

C FBA

As B is on the top of the stack so calculate the neighbors of B.

F. 0+5+4 = 9

G. 0+5+6 = 11

The least distance is D from B. So it will be on the top of the stack.

A5B 4C

Step 3:

As the top of the stack is D. So calculate neighbours of D.

CFDB

H. 0+5+4+8 = 17

I. 0+5+4+3 = 12

The least distance is F from D and it is our goal node. So stop and return success.

Step 4:

C F D

Hence the searching path will be A-B -D-F

Advantages:

As it finds the minimum path instead of finding the minimum successor so there should not

Disadvantages:

The load balancing aspects for Branch and Bound algorithm make it parallelization difficult.

The Branch and Bound algorithm is limited to small size network. In the problem of large networks, where the solution search space grows exponentially with the scale of the network, the approach becomes relatively prohibitive.

A* SEARCH

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search.

In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state.

This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula

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

Where **g (n): The actual cost path from the start state to the current state.**

h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and

CLOSE.

OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and

place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

The implementation of A* algorithm is 8-puzzle game.

Advantages:

J. It is complete and optimal.

K. It is the best one from other techniques. It is used to solve very complex problems.

L. It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula

$$.\textbf{f (n) + g (n) = h (n)}$$

Where **g (n): The actual cost path from the start state to the current state.**

 h (n): The actual cost path from the current state to goal state.

 f (n): The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and

CLOSE.

OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and

place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

The implementation of A* algorithm is 8-puzzle game.

Advantages:

M. It is complete and optimal.

N. It is the best one from other techniques. It is used to solve very complex problems.

O. It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

P. This algorithm is complete if the branching factor is finite and every action has fixed cost.

Q. The speed execution of A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h (n).It has complexity problems.

AO* Search: (And-Or) Graph

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

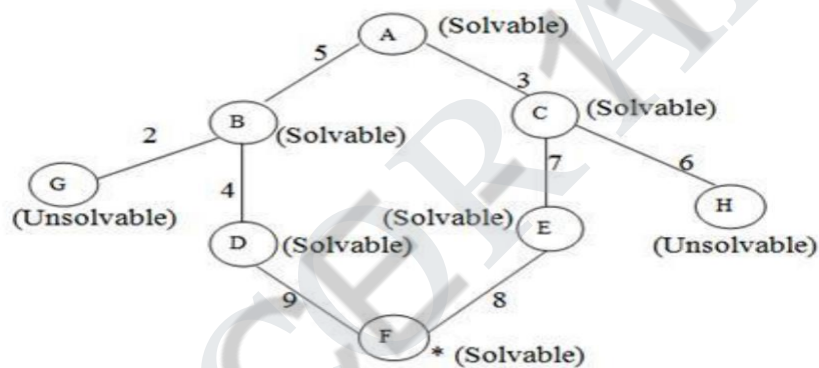**Step 3:** Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved.If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable,then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Step 1:

In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

Step 2:

The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.

Step 3:

Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.

Step 4:

As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

Step 5:

Now we have been reached at our goal state. So place F into CLOSE.

Step 6:

Success and Exit

    Advantages:

    R. It is an optimal algorithm.

    S. If traverse according to the ordering of nodes. It can be used for both OR and AND graph

    .**Disadvantages:**

    a. Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

5. **Explain the following types of hill climbing search techniques**   **Dec-18, May 16,Dec 17**
    1. Simple hill climbing
    2. Steepest hill climbing
    3. Simulated annealing

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a ―peak‖ where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method.

6.

The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches at the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm:

Step 1: Evaluate the starting state. If it is a goal state then stop and return success.

Step 2: Else, continue with the starting state as considering it as a current state.

Step 3: Continue step-4 until a solution is found i.e. until there are no new states left

to be applied in the current state.

Step 4**:**

    T. Select a state that has not been yet applied to the current state and apply it to produce a new state.

    U. Procedure to evaluate a new state.
- a. If the current state is a goal state, then stop and return success.

- b. If it is better than the current state, then make it current state and proceed further.

- c. If it is not better than the current state, then continue in the loop until a solution is found.

Step 5:Exit.

Advantages:

Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management. It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function. It requires much less conditions than other search techniques.

Disadvantages:

The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement. There are some reasons by which hill climbing often gets suck which are stated below.

Local Maxima:

A local maxima is a state that is better than each of its neighbouring states, but not better than some other states further away. Generally this state is lower than the global maximum. At this point, one cannot decide easily to move in which direction! This difficulties can be extracted by the process of backtracking i.e. backtrack to any of one earlier node position and try to go on a different event direction.

To implement this strategy, maintaining in a list of path almost taken and go back to one of them. If the path was taken that leads to a dead end, then go back to one of them.
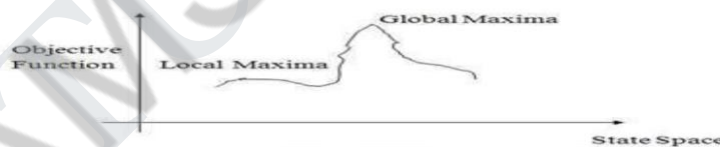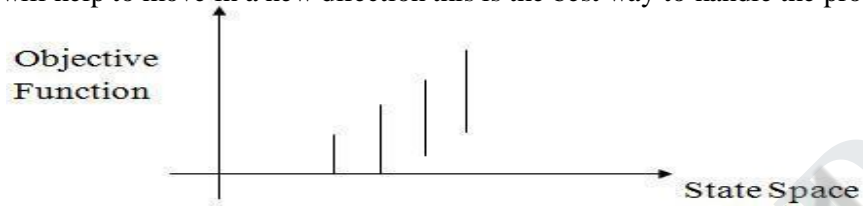

Figure Local Maxima

It is a special type of local maxima. It is a simply an area of search space. Ridges result in a sequence of local maxima that is very difficult to implement ridge itself has a slope which is difficult to traverse. In this type of situation apply two or more rules before doing the test. This will correspond to move in several directions at once.s

Plateau:

It is a flat area of search space in which the neighbouring have same value. So it is very difficult to calculate the best direction. So to get out of this situation, make a big jump in any direction, which will help to move in a new direction this is the best way to handle the problem like plateau.

Figure Plateau

To overcome these problems we can

> V. Back track to some earlier nodes and try a different direction. This is a good

way of dealing with local maxim.

> W. Make a big jump an some direction to a new area in the search. This can be

done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plate and ridges. Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it is a useful when combined with other methods.

Steepest Descent Hill Climbing

> X. This is a variation of simple hill climbing which considers all the moves from the

current state and   selects the best one as the next state.

> Y. Also known as Gradient search

Algorithm

Z. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.

AA. Loop until a solution is found or until a complete iteration produces no change to current state:

- Let SUCC be a state such that any possible successor of the current state will be better than SUCC
- For each operator that applies to the current state do:
- Apply the operator and generate a new state
- Evaluate the new state. If is is a goal state, then return it and quit. If not, compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
- If the SUCC is better than the current state, then set current state to SYCC,

7. **Discuss about constraint satisfaction problem with a algorithm for solving a cryptarithmetic Problem   Dec-09, Dec-13,May 15,May-14,Dec 17, Dec-18,May-15**
A constraint search does not refer to any specific search algorithm but to a Layer of complexity added to existing algorithms that limit the possible solution set. Heuristic and acquired knowledge can be combined to produce the desired result a constraint satisfaction problem is a special kind of search problem in which states are defined by the values of a set of variables and the goal state specifies a set of constraints that the value must obey. There are many problems in AI in which the goal state is not specified in the problem and it requires to be discovered according to some specific constraint.
Examples of some constraint satisfaction search include design problem, labeling graphs, robot path planning and cryptarithmetic problem etc.
Algorithm:

☐ Open all objects that must be assigned values in a complete solution.
☐ Repeat until all objects assigned valid values.
☐ Select an object and strengthen as much as possible. The set of constraints that
apply to object.
☐ If set of constraints is different from previous set then open all objects that share any
of these constraints. Remove selected objects.
☐ If union of constraints discovered above defines a solution, return solution.
☐ If union of constraints discovered above defines a contradiction, return failure.
☐ Make a guess in order to proceed. Repeat until a solution is found.
☐ Select an object with a number assigned value and try strengthening its constraints.

Consider the problem of solving a puzzle

E+V+O+L=AI, If A = 3 && 5<L<O<V<E<10, Find the value of I?

Many AI problems can be viewed as problems of constraint satisfaction.

As compared with a straightforward search procedure, viewing a problem as one of constraint satisfaction can reduce substantially the amount of search.

Constraint Satisfaction

☐ Operates in a space of constraint sets.

☐ Initial state contains the original constraints given in the problem.

☐ A goal state is any state that has been constrained —enough‖.

Two-step process:

☐ Constraints are discovered and propagated as far as possible.

☐ If there is still not a solution, then search begins, adding new constraints.

Two kinds of rules:

☐ Rules that define valid constraint propagation.

☐ Rules that suggest guesses when necessary.

Constraints

☐ The simplest type is the unary constraint, which constraints the values of just one variable.

☐ A binary constraint relates two variables.

☐ Higher-order constraints involve three or more variables. Cryptarithmetic puzzles are an example:

Cryptarithmetic puzzles

•Variables:F, T, U, W, R, O, X1, X2, X3

•Domains: {0,1,2,3,4,5,6,7,8,9}

•Constraints:

–Alldiff(F,T,U,W,R,O)

–$O + O = R + 10 \cdot X1$

–$X1+ W + W = U + 10 \cdot X2$

–$X2+ T + T = O + 10 \cdot X3$

–$X3= F, T \neq 0, F \neq 0$

If 2+2 = 4, F=1, then how many fours ?

928+928=1856

867+867=1734

846+846=1692

836+836=1672

765+765=1530

734+734=1468

TWO

TWO

FOUR

938+938=1876

12.Explain alpha-beta pruning algorithm and the Minmax game playing algorithm with example? Dec-03,Dec-04,May-10,May-10, May-09, May 17, May 19,Dec-04, May-10, May-10,Dec-10 ,May 17

ALPHA-BETA pruning is a method that reduces the number of nodes explored in Minimax strategy. It reduces the time required for the search and it must be restricted so that no time is to be wasted searching moves that are obviously bad for the current player.

The exact implementation of alpha-beta keeps track of the best move for each side as it moves throughout the tree.

We proceed in the same (preorder) way as for the minimax algorithm. For the MIN nodes, the score computed starts with +infinity and decreases with time.

For MAX nodes, scores computed starts with –infinity and increase with time.

The efficiency of the Alpha-Beta procedure depends on the order in which successors of a node are examined. If we were lucky, at a MIN node we would always consider the nodes in order from low to high score and at a MAX node the nodes in order from high to low score. In general it can be shown that in the most favorable circumstances the alpha-beta search opens as many leaves as minimax on a game tree with double its depth.

Alpha-Beta algorithm: The algorithm maintains two values, alpha and beta, which represents the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller. When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.

Pseudocode for the alpha-beta algorithm.

evaluate (node, alpha, beta)

if node is a leaf

return the heuristic value of node

if node is a minimizing node

for each child of node

beta = min (beta, evaluate (child, alpha, beta))

if beta <= alpha

return beta

return beta

if node is a maximizing node

for each child of node

alpha = max (alpha, evaluate (child, alpha, beta))

if beta <= alpha

return alpha

Min Max Algorithm

The Min-Max algorithm is applied in two player games, such as tic-tac-toe, checkers, chess, go, and so on.

There are two players involved, MAX and MIN. A search tree is generated, depth-first, starting with the current game position upto the end game position. Then, the final game position is evaluated from MAX's point of view, as shown in Figure 1. Afterwards, the inner node values of the tree are filled

bottom-up with the evaluated values. The nodes that belong to the MAX player receive the maximum value of it's children. The nodes for the MIN player will select the minimun value of it's children.

The values represent how good a game move is. So the MAX player will try to select the move with highest value in the end. But the MIN player also has something to say about it and he will try to select the moves that are better to him, thus minimizing MAX's outcome.

Algorithm

```
MinMax (GamePosition game) {

return MaxMove (game);

}

MaxMove (GamePosition game)

 {

if (GameEnded(game))

 {

return EvalGameState(game);

}

else

{

best_move <- {};

moves <- GenerateMoves(game);
ForEach moves
```

```
    {

    move <- MinMove(ApplyMove(game));

    if (Value(move) > Value(best_move))

    {

    best_move <- move;

    }

    }

    return best_move;

    }

    }

MinMove (GamePosition game) {

best_move <- { };

moves <- GenerateMoves(game);

ForEach moves {

move <- MaxMove(ApplyMove(game));

if (Value(move) > Value(best_move)) {

best_move <- move;
```

}

}

return best_move;

}

Optimization

                    BB.price

                    CC.Limit the depth of the tree.

Speed up the algorithm

This all means that sometimes the search can be aborted because we find out that the search subtree won't lead us to any viable answer. This optimization is known as alpha-beta cutoffs.

The algorithm Have two values passed around the tree nodes: The alpha value which holds the best MAX value found; The beta value which holds the best MIN value found.

At MAX level, before evaluating each child path, compare the returned value with of the previous path with the beta value. If the value is greater than it abort the search for the current node;

At MIN level, before evaluating each child path, compare the returned value with of the previous path with the alpha value. If the value is lesser than it abort the search for the current node.

i) Write an algorithm for converting to clause form. (June 2013)
        Convert the following well – formed formula into clause form with sequence of steps,
X: [Roman (x) □ Know (x, Marcus)] □ [hate (x, caeser) □ (∀ y : □z: hate(y,z) □

    Thinkcrazy (x,y))] **(MAY/JUNE 2016)**

To convert the axioms into conjunctive normal form (clause form)

DD.   Eliminate implications

EE. Move negations down to the atomic formulas

FF. Purge existential quantifiers

GG.   Rename variables if necessary

HH.   Move the universal quantifiers to the left

II.   Move the disjunctions down to the literals

JJ.   Eliminate the conjunctions

KK.   Rename the variables if necessary

LL. Purge the universal quantifiers

Example

Consider the sentence

—All music lovers who enjoy Bach either dislike Wagner or think that anyone who dislikes any composer is a philistine".

Use enjoy() for enjoying a composer, and for dislike.

"x[musiclover(x)   enjoy(x,Bach)

dislike(x,Wagner) ("y[$z[dislike(y,z)]   =>think-philistine(x,y)])]
Conversion

Step 1: Filter the expression to remove symbols.

"x [(musiclover(x) enjoy(x,Bach)) => dislike(x,Wagner) $\lor$ ("y[$z [dislike(y,z)]=> think-philistine(x,y)])].

Step 2: Filter using the following relationships:

(¬ P) <=> P;

¬ (a ∧ b) <=> ¬ a ∨ ¬ b;

¬ (a ∨ b) <=> ¬ a ∧ ¬ b;

¬ ∀x P(x) <=> ∃x ¬ P(x); and

¬ ∃x P(x) <=> ∀x ¬ P(x).

Now the expression becomes

∀x[¬ musiclover(x) ∨ ¬ enjoy(x,Bach) ∨ dislike(x,Wagner) ∨ ∀y[∀z[¬ dislike(y,z)] ∨ think-philistine(x,y)]].

Step 3: Standardize variables so that each quantifier binds a unique variable. ∀x Pred1(x) ∨ ∀x Pred2(x)

becomes

∀x Pred1(x) ∨ ∀y Pred2(y).

Step 4: Step 3 allows us to move all the quantifiers to the left in Step 4. The expression becomes

∀x∀y∀z[¬ musiclover(x) ∨ ¬ enjoy(x,Bach) ∨

dislike(x,Wagner) ∨ ¬ dislike(y,z) ∨ think-philistine(x,y)].

This is called the prenex normal form.

> Step 5: Eliminate existential quantifiers, by arguing that if $y Composer(y), then if we could actually find an Object S1 to replace the Variable x. So this gets replaced simply by Composer(S1).

Now, if existential quantifiers exist within the scope of universal quantifiers, we can't

use merely an object, but rather a function that returns an object. The function will

depend on the univeral quantifier.

"x$y tutor-of(y,x)

gets replaced by

"x tutor-of(S2(x),x).

This process is called Skolemization, and the S2 is a Skolem function.

Step 6: Any variable left must be universally quantified out on the left.

The expression becomes:

> $\neg$ musiclover(x) $\lor$ $\neg$ enjoy(x,Bach) $\lor$ dislike(x,Wagner) $\lor$ $\neg$ dislike(y,z) $\lor$ think-philistine(x,y).

Step 7: Convert everything into a conjunction of disjunctions using the associative,

commutative, distributive laws.

The form is

$(a \lor b \lor c \lor d \lor ...) \land (p \lor q \lor ...) \land ...$

Step 8: Call each conjunct a separate clause. In order for the entire wff to be true, each clause must be true separately.

Step 9: Standard apart the variables in the set of clauses generated in 7 and 8. This requires renaming the variables so that no two clauses make reference to the same variable. All variables are implicitly universally quantified to the left.

"x P(x) $\land$ Q(x) <=> "xP(x) $\land$ "xQ(x) <=> "xP(x) $\land$ "yQ(y)

This completes the Procedure.

After application to a set of wffs, we end up with a set of clauses each of which is a disjunction of literals.

Convert the following well – formed formula into clause form with sequence of steps,

X: [Roman (x) □ Know (x, Marcus)] □ [hate (x, caeser) □ (∀ y : □z: hate(y,z) □

Thinkcrazy (x,y))] (MAY/JUNE 2016)

Solution:

"All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy."

There are 9 simple steps to convert from Predicate logic to clause form. We will use the following example :All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy."

The well-formed formula for this statement is :

"x [Roman(x) and know(x,Marcus)] ® [hate(x,Caesar) Ú ("y($z hate(y,z))® thinkcrazy(x,y))]

Well-formed formula [wff] : "x [Roman(x) and

know(x,Marcus)] ®[hate(x,Caesar) Ú ("y($z hate(y,z)) ® think crazy(x,y))]

1. Eliminate

We will eliminate implication [®] by substituting it with its equivalent.

for e.g. a ® b = ~a v b .

Here 'a' and 'b' can be any predicate logic expression.

For the above statement we get :

"x ~[Roman(x) Ù know(x,Marcus)] Ú [hate(x,Caesar) Ú("y~($z hate(y,z)) Úthinkcrazy(x,y))]

2. Reduce the scope of ~

To reduce the scope we can use 3 rules :

~(~p) = p

DeMorgans Laws :   ~(aÚb) = ~aÙ~b

                         ~(aÙb) = ~aÚ~b

Applying this reduction on our example yields :

"x[~Roman(x) Ú~know (x,Marcus)] Ú [hate(x,Caesar) Ú ("y"z ~hate(y,z) Ú thinkcrazy(x,y))]

3.Change variable names such that, each quantifier has a unique name.

We do this in preparation for the next step. As variables are just dummy names, changing a variable name doesnot affect the truth value of the wff. Suppose we have

"xP(x) Ú "xQ(x) will be converted to "x(P(x) Ú "yQ(y)

4.Move all the quantifiers to the left of the formula without changing their relative order.

As we already have unique names for each quantifier in the previous step, this will not cause a problem.

Performing this on our example we get : "x"y"z [
~Roman(x)   Ú   ~know(x,Marcus)]   Ú   [
hate(x,Caesar)Ú(~hate(y,z)Úthinkcrazy(x,y))]

5. Eliminate existential quantifiers [$]

We can eliminate the existential quantifier by simply replacing the variable with a reference to a function that produces the desired value.

for eg. $y President(y) can be transformed into the formula President(S1)

If the existential quantifiers occur within the scope of a universal wuantifier, then the value that satisfies the predicate may depend on the values of the universally quantified variables.

For eg..   "x$y fatherof(y,x) will be converted to "x fatherof(S2(x),x)

6. Drop the Prefix

As we have eliminated all existential quantifiers, all the variables present in the wff are unversally quantified, hence for simplicity we can just drop the prefix, and assume that every variable is universally quantified. We have form our example :

[~Roman(x) Ú ~know(x,Marcus)] Ú [hate(x,Caesar)Ú(~hate(y,z)Úthinkcrazy(x,y))]

7. Convert into conjunction of disjuncts

as we have no ANDs we will just have to use the associative property to get rid of the brackets.

In case of ANDs we will need to use the distributive property.

We have :

~Roman(x) Ú ~know(x,Marcus) Ú hate(x,Caesar) Ú ~hate(y,z) Ú thinkcrazy(x,y)

8. Separate each conjunct into a new clause.

As we did not have ANDs in out example, this step is avoided for our example and the final output of the conversion is :

~Roman(x) Ú ~know(x,Marcus) Ú hate(x,Caesar) Ú ~hate(y,z) Úthinkcrazy(x,y)

Unit-3:

Two-marks problem:

1. What are the limitations in using propositional logic to represent the knowledge base? May-11
   Propositional logic has following limitations to represent the knowledge base.
   i. It has limited expressive power.
   ii. It cannot directly represent properties of individuals or relations between individuals.
   iii. Generalizations, patterns, regularities cannot easily be represented.
   iv. Many rules (axioms) are requested to write so as to allow inference.

2. Name two standard quantifiers.   **Dec – 09,May – 13**
   The two standard quantifier are universal quantifiers and existential quantifier.
   They are used for expressing properties of entire collection of objects rather just a single object.
   Eg. $\forall x$ Happy(x) means that "if the universe of discourse is people, then everyone is happy".

$\exists$x Happy(x) means that "if the universe of discourse is people, then this means that there is at-least one happy person."

3. What is the purpose of unification?   **May – 12,Dec – 12,Dec - 09**

It is for finding substitutions for inference rules, which can make different logical expression to look identical. It helps to match to logical expressions. Therefore it is used in many algorithm in first order logic.

4.What is ontological commitment (what exists in the world) of first order logic? Represent the sentence "Brothers are siblings" in first order logic?   **Dec - 10**

Ontological commitment means what assumptions language makes about the nature if reality.
Representation of "Brothers are siblings" in first order logic is
$\forall$ x, y [Brother (x, y) ➔ Siblings (x, y)]

5.Differentiate between propositional and first order predicate logic?   **May – 10 , Dec – 11**

Following are the comparative differences versus first order logic and propositional logic.
1) Propositional logic is less expressive and do not reflect individual object`s properties explicitly. First order logic is more expressive and can represent individual object along with all its properties.
2) Propositional logic cannot represent relationship among objects whereas first order logic can represent relationship.
3) Propositional logic does not consider generalization of objects where as first order logic handles generalization.
4) Propositional logic includes sentence letters (A, B, and C) and logical connectives, but not quantifier.
   First order logic has the same connectives as propositional logic, but it also has variables for individual objects, quantifier, symbols for functions and symbols for relations.

6.What factors justify whether the reasoning is to be done in forward or backward reasoning?

                                Dec - 11

Following factors justify whether the reasoning is to be done in forward or backward reasoning:

     a.   possible to begin with the start state or goal state?
     b.   Is there a need to justify the reasoning?
     c.   What kind of events trigger the problem - solving?
     d.   In which direction is the branching factor greatest? One should go in the direction with lower branching factor?

7. Define diagnostic rules with example?   **May – 12**

Diagnostics rules are used in first order logic for inference. The diagnostics rules generate hidden causes from observed effect. They help to deduce hidden facts in the world. For example consider the Wumpus world.
The diagnostics rule finding 'pit' is

"If square is breezy some adjacent square must contain pit", which is written as, ∀s Breezy(s) => ∃ Adjacent (r,s) ∩ pit (r).

8. Represent the following sentence in predicate form:

"All the children like sweets" **Dec – 12**

∀x child(x) ∩ sweet(y) ∩ likes (x,y).

5. what is Skolemization? **May - 13**

It is the process of removing existential quantifier by elimination. It converts a sentence with existential quantifier into a sentence without existential quantifier such that the first sentence is satisfiable if and only if the second is.

For eliminating an existential quantifier each occurrence of its variable is replaced by a skolem function whose argument are the variables of universal quantifier whose argument are the variables of universal quantifier whose scope includes the scope of existential quantifier.

6. Define the first order definite clause? **Dec – 13**

1) They are disjunctions of literals of which exactly one is positive.
2) A definite clause is either atomic sentence or is an implication whose antecedents (left hand side clause) is a conjunction of positive literals and consequent (right hand side clause) is a single positive literal.

For example:

Princess (x) ∧ Beautiful (x) ⇒ Goodhearted(x)

Princess(x)

Beautiful(x)

7. Write the generalized Modus ponens Rule? **May – 14**

1) Modus ponens :

If the sentence P and P →Q are known to be true, then modus ponens lets us infer Q

For example : if we have statement , " If it is raining then the ground will be wet" and "It is raining". If P denotes "It is raining" and Q is "The ground is wet" then the first expression becomes P →Q. Because if it is indeed now raining (P is true), our set of axioms becomes,

{ P →Q

 P }

Through an application of modus ponens, the fact that "The ground is wet" (Q) may be added to the set of true expressions.

2) The generalized modus ponens :

For atomic sentences P_i , P'_i and q, where there is a substitution Q such that SUBST $(θ,P'_i)$ = SUBST $(θ,P'_i)$,

For all i ,

$P'_1, P'_2 , …… P'_n , (P_1 ∧ P_2 ∧ …..∧ P_n ⇒ q)$

SUBST $(θ, q)$

There is n+ 1 premise to this rule: The 'n' atomic sentences P'_i and the one implication. The conclusion is the result applying the substitution θ to the consequent q.

8. Define atomic sentence and complex sentence? **Dec – 14**

Atomic sentences
1. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

For example: Stepsister (Cindrella, Drizella)

2. Atomic sentences can have complex terms as the arguments.

For example: Married (Father (Cindrella), Mother (Drizella))

3. Atomic sentences are also called atomic expressions, atoms or propositions.

For example: Equal (plus (two, three), five) is an atomic sentence.

Complex sentences

i) Atomic sentences can be connected to each other to form complex sentence.

Logical connectives, $\wedge$, $\vee$, $\neg$, $\rightarrow$ can be used to connect atomic sentences.

For example:

$\neg$ Princess (Drizella) $\rightarrow$ Princess (Cindrella)

ii) (foo(two, two, plus(two, three))) $\rightarrow$ (Equal (plus (three, two), five) $\equiv$ true) is a sentence because all its components are sentences, appropriately connected by logical operators.

iii) Various sentences in first order logic formed using connectives:
1) If S is a sentence, then so its negation, $\neg$ S.
2) If S_1, and S_2 are sentences, then so their conjunction, $S_1 \wedge S_2$.
3) If S_1, and S_2 are sentences, then so their disjunction, $S_1 \vee S_2$.
4) If S_1, and S_2 are sentences, then so their implication, $S_1 \rightarrow S_2$.
5) If S_1, and S_2 are sentences, then so their equivalence, $S_1 \equiv S_2$.

9. What is Unification? **Dec - 14**
1) It is the process of finding substitutions for lifted inference rules, which can make different logical expression to look similar (identical).
2) Unification is a procedure for determining substitutions needed to make two first order logic expressions match.
3) Unification is important component of all first order logic inference algorithms.
4) The unification algorithm takes two sentences and returns a unifier for them, if one exists.

10. Differentiate forward chaining and backward chaining? **May – 15**

Forward chaining is data driven

It is automatic unconscious processing.

Ex. – Object reorganization, routine decision.

It may do lots of work that is irrelevant to the goal.

Backward chaining is goal driven.

It is appropriate for the problem solving.

Ex. Where are my keys?, How do I get into a Ph.D programme?

Complexity of backward chaining can be much less than linear in size of knowledge base.

11. Define metarules? **May - 17**

The rules that determine the conflict resolution strategy are called meta rules. Meta rules define knowledge about how the system will work. For example, meta rules may define that

knowledge from expert1 is to be trusted more than knowledge from expert 2. Meta rules are treated by the system like normal rules but are they are given higher priority.

Convert the following into Horn clauses. **Dec -17**

$$\forall x: \forall y: cat(x) \vee fish(y) \xrightarrow{eat} likes \ x, y$$

Horn clauses are as follows,

$$\neg cat(x) \vee \neg fish(y) \vee likes \xrightarrow{eat} x, y$$

12. Explain following term with reference to prolog programming language :clauses

Clauses: clauses are the structure elements of the program. A prolog programmer develops a program by writing a collection of clauses in a text file. The programmer the uses the consult command ,specifying the name of the text file, to load the process into the prolog environment.

The two types of clauses – facts and rules.

Facts – a fact is an atom or structure followed by fullstop .examples of valid prolog syntax defining facts are :cold , male(homer).and father(homer,bart)

Rules: a rule consist of a head and body . the head and body are separated by a :- and followed by a fullstop. If the body of a clause id true then the head of the clause is true. Examples of valid prolog syntax for defining rules are: bigger(X,Y):-X>Y.and parents(F,M,C):-father (F,C),mother(M,C).

13. explain following term with refernce to prolog programming language : predicates

Each predicate has a name and zero or more arguments .the predicate name is a prolog atom . each argument is an arbitrary prolog term. A predicate with pred and n arguments is denoted by pred/N, which is called a predicate indicator. A predicate is defined by a collection of clauses.

A clause is either a rule or fact . A clauses that constitute a predicate denote logical alternative: if any clause is true, then the whole predicate is true.

14. explain the following term with reference to prolog programming language : domains

Domains : the argument to the predicates must belong to know prolog domains. A domain can be a standard domain, or it can be one you declare in the domain section. The two types of process- facts and rules. Examples :

If you declare a predicate my_predicate(symbol,integer) in the predicate section, like this: predicates:

My_predicate(symbol,integer)

You don't need to declare its arguments domains in the domain section, because symbol and integer are standard domains. But if you declare a predicate my_predicate(name,number) in the predicates section, like this

Predicates:

my_predicate(name,number) you will need to declare suitable domains for name and number.

Assuming you want these to be symbol and integer respectively, the domain declaration looks like this.

Domains:

Name=symbol

Number = integer

Predicates:

my_predicate(name,number)

15. explain the following term with reference to prolog programming language :goal

a goal is a statement starting with a predicate and probably followed by its arguments. In a valid goal,the predicate must have appeared in atleast one fact or a rule in the consulted program, and a number of arguments in the goal must be the same as that appears in the consulted program . also, al the arguments (if any) or constants.

The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database(i.e. the facts and rules in the consulted program). This is similar to proving a hypothesis – the goal being the hypothesis, the facts being the axioms and the rules being the theorem.

16. explain the following term with reference to prolog programming language : cut

The cut, in prolog , is a goal, return us !,which always succeeds, but cannot be backtracked past. The prolog cut predicate, or '!',eliminates choices is a prolog derivation tree it is used to prevent unwanted backtracking, for example, to prevent extra solutions being found by prolog.

The cut should be used sparingly. There is a temptation to insert cuts experimentally into code that is not working correctly.

17. explain the following term with reference to prolog programming language :fail

It is the built-in prolog predicate with no arguments, which, as the name suggest, always fails, it is useful for forcing backtracking and various other contexts.

The two types of process- facts and rules.

18. explain the following term with reference to prolog programming language : Inference engine

Inference engine: prolog built-in backward chaining inference engine which can be used to partially implement some expert system. Prolog rules are used for the knowledge representation, and the prolog inference engine is used to derive conclusions. Other portions of the system, such as the user interface, must be coded using prolog as a programming language. The prolog inference engine thus simple backward chaining. Each rule has a goal and a number of each sub-goals. The prolog inference engine either proves or disproves each goal. There is no uncertainty associated with the results.

This rule structure and inference strategy is adequate for many expert system applications. Only the dialogue with the user needs to be improved to create a simple expert system. These feature are used in he chapter to build a sample application called, "birds,"which identifies birds.

19. define ontological engineering?

It is a process of representing the abstract concepts like actions,time which are related to the real world domains. This process is complex and lengthy because in real world objects have many different characteristics with various values which can differ over time. In such cases ontological engineering generalizes the objects having similar characteristics.

20. Differentiate general purpose ontology from special purpose ontology.

Special purpose ontology considers some basics facts about the world in such a way tat they may notbe represented in generalized manner. It provides domain specific axioms.

Whereas the general purpose ontology is applicable to any special purpose domain with the addiction of domain specific axioms, it tries to represent real world abstract concepts in more generic manner, so as to cover larger domains.

A general purpose ontology unifies and do reasoning for sufficiently large domains and different areas, where as special purpose ontology is restricted to specific problem domain.

16 MARKS QUESTIONS:

1. **Write the algorithm for deciding entailment in propositional logic.**   **May 13 Dec 14**
   REFER Qno 7

2. **What is conductive normal form of a rule? What is solemnizations?(4)**   **Dec -10**
   REFER Qno 7

3. **Describe the detail the steps involved in the knowledge engineering process?May – 10,May – 11,May-13,Dec-13**
   Knowledge Engineering is the process of imitating how a human expert in a specific domain would act and take decisions. It looks at the **metadata** (information about a data object that describes characteristics such as content, quality, and format), structure and processes that are the basis of how a decision is made or conclusion reached. Knowledge engineering attempts to take

on challenges and solve problems that would usually require a high level of human expertise to solve. Figure 1 illustrates the knowledge engineering pipeline.
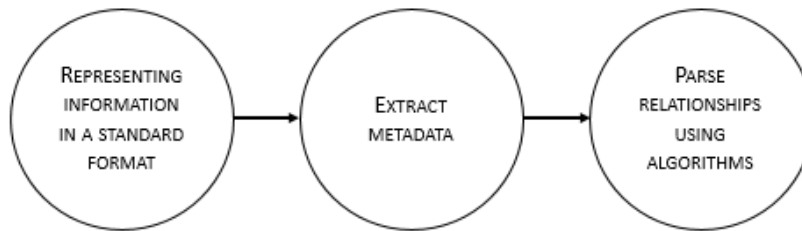
Figure 1: Knowledge engineering pipeline

Knowledge Engineering Processes

In terms of its role in **artificial intelligence (AI)**, knowledge engineering is the process of understanding and then representing human knowledge in **data structures**, **semantic models** (conceptual diagram of the data as it relates to the real world) and **heuristics** (rules that lead to solution to every problem taken in AI). **Expert systems**, and **algorithms** are examples that form the basis of the representation and application of this knowledge.
The knowledge engineering process includes:

- Knowledge acquisition
- Knowledge representation
- Knowledge validation
- Inferencing
- Explanation and justification

The interaction between these stages and sources of knowledge is shown in Figure 2.
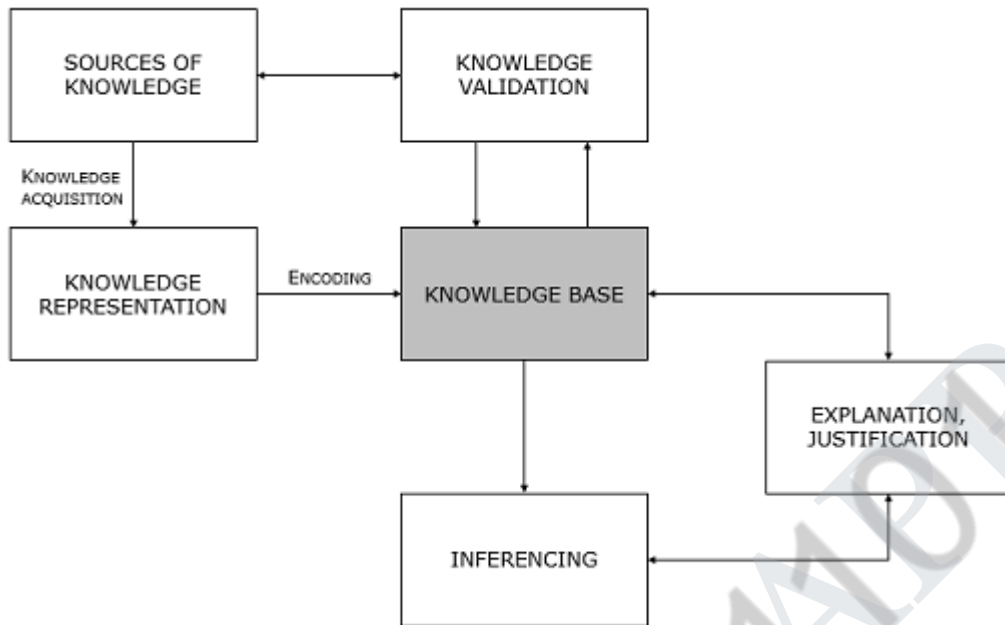
Figure 2: Knowledge engineering processes

The amount of collateral knowledge can be very large depending on the task. A number of advances in technology and technology standards have assisted in integrating data and making it accessible. These include the **semantic web** (an extension of the current web in which information is given a well-defined meaning), **cloud computing** (enables access to large amounts of computational resources), and **open datasets** (freely available datasets for anyone to use and republish). These advances are crucial to knowledge engineering as they expedite data integration and evaluation.

4.Explain unification algorithm used for reasoning under predicate logic (first order logic) with an example? **Dec-09,May-10,May – 11**

    a. **Knowledge can be represented** as *"symbol structures"* that characterize bits of knowledge about objects, concepts, facts, rules, strategies.

Examples: **"red"** represents *colour red*, **"car1"** represents *my car* , **"red(car1)"** represents fact that *my car is red*.

Assumptions about KR :

    4. *Intelligent Behavior* can be achieved by manipulation of symbol structures.

    5. *KR languages* are designed to facilitate operations over symbol structures, have precise syntax and semantics; *Syntax* tells which expression is legal? **e.g., red1 (car1), red1 car1, car1(red1), red1(car1 & car2) ?;** and *Semantic* tells what an expression means ?

e.g., property "dark red" applies to my car.

6. *Make Inferences, draw new conclusions* from existing facts.

To satisfy these assumptions about KR, we need formal notation that allows automated inference and problem solving. One popular choice is use of **logic**.

Logic

Logic is concerned with the truth of statements about the world. Generally each statement is either *TRUE* or *FALSE*. Logic includes: *Syntax, Semantics and Inference Procedure*.

1. Syntax:

Specifies the *symbols* in the language about how they can be combined to form sentences. The facts about the world are represented as sentences in logic.

2. Semantic:

Specifies how to assign a truth value to a sentence based on its *meaning* in the world. It Specifies what facts a sentence refers to. A fact is a claim about the world, and it may be *TRUE* or *FALSE*.

3. Inference Procedure:

Specifies *methods* for computing new sentences from the existing sentences. **Logic as a KR Language**

Logic is a language for reasoning, a collection of rules used while doing logical reasoning. Logic is studied as KR languages in artificial intelligence. Logic is a formal system in which the formulas or sentences have true or false values. Problem of designing KR language is a tradeoff between that which is

MM. Expressive enough to represent important objects and relations in a problem domain.

NN.   Efficient enough in reasoning and answering questions about implicit information in a reasonable amount of time.

Logics are of different types: Propositional logic, Predicate logic, temporal logic, Modal logic, Description logic etc;

They represent things and allow more or less efficient inference. Propositional logic and Predicate logic are fundamental to all logic. Propositional Logic is the study of statements and their connectivity. Predicate Logic is the study of individuals and their properties.

Logic Representation

Logic can be used to represent simple facts.

The *facts* are claims about the world that are *True* or *False*. To build a Logic-based representation:

  a. User defines a set of primitive *symbols* and the associated *semantics*.

  b. Logic defines ways of putting symbols together so that user can define

legal *sentences* in the   language that represent *TRUE* facts.

  OO.   Logic defines ways of inferring *new sentences* from existing ones.

  PP. Sentences - either *TRUE* or *false* but not both are called *propositions*.

  QQ.   A declarative sentence expresses a *statement* with a proposition as content; example:

the declarative **"snow is white"** expresses that **snow is white**; further, **"snow is white"** expresses that *snow is white* is **TRUE**.

Resolution and Unification algorithm

In propositional logic it is easy to determine that two literals cannot both be true at the same time. Simply look for L and ~L . In predicate logic, this matching process is more complicated, since bindings of variables must be considered.

For example man (john) and man(john) is a contradiction while man (john) and man(Himalayas) is not. Thus in order to determine contradictions we need a matching procedure that compares two literals and discovers whether there exist a set of substitutions that makes them identical. There is a recursive procedure that does this matching . It is called Unification algorithm.

In Unification algorithm each literal is represented as a list, where first element

is the name of a predicate and the remaining elements are arguments. The

argument may be a single element (atom) or may be another list. For example we

can have literals as

(tryassassinate Marcus Caesar)

(tryassassinate Marcus (ruler of Rome))

To unify two literals, first check if their first elements re same. If so proceed. Otherwise they cannot be unified. For example the literals (try assassinate Marcus Caesar)

(hate Marcus Caesar)

Cannot be unified. The unification algorithm recursively matches pairs of elements, one pair at a time. The matching rules are :

RR. Different constants, functions or predicates can not match, whereas identical ones can.

SS. A variable can match another variable, any constant or a function or predicate expression, subject to the condition that the function or

[predicate expression must not contain any instance of the variable being matched (otherwise it will lead to infinite recursion).

TT. The substitution must be consistent. Substituting y for x now and then z for x later is inconsistent. (a substitution y for x written as y/x)

The Unification algorithm is listed below as a procedure UNIFY (L1, L2). It

returns a list representing the composition of the substitutions that were performed

during the match. An empty list NIL indicates that a match was found without any

substitutions. If the list contains a single value F, it indicates that the unification

procedure failed.

UNIFY (L1, L2)

UU.   if L1 or L2 is an atom part of same thing do

a.   if L1 or L2 are identical then return NIL

b.   else if L1 is a variable then do

VV.   if L1 occurs in L2 then return F else return (L2/L1) © else if L2 is a variable then do

WW.   if L2 occurs in L1 then return F else return (L1/L2)

else return F.

XX.   If length (L!) is not equal to length (L2) then return F.

YY.   Set SUBST to NIL

(at the end of this procedure , SUBST will contain all the substitutions used to unify L1 and L2).

4. For I = 1 to number of elements in L1 do

ZZ. call UNIFY with the i th element of L1 and I'th element of L2, putting the result in S

AAA.   if S = F then return F

BBB.   if S is not equal to NIL then do

CCC.   apply S to the remainder of both L1 and L2

DDD.   SUBST := APPEND (S, SUBST) return SUBST.

Resolution yields a complete inference algorithm when coupled with any complete search algorithm. Resolution makes use of the inference rules. Resolution performs deductive inference. Resolution uses proof by contradiction. One can perform Resolution from a Knowledge Base. A Knowledge Base is a collection of facts or one can even call it a database with all facts.

Resolution basically works by using the principle of proof by contradiction. To find the conclusion we should negate the conclusion. Then the resolution rule is applied to the resulting clauses.

Each clause that contains complementary literals is resolved to produce a 2new clause, which can be added to the set of facts (if it is not already present). This process continues until one of the two things happen. There are no new clauses that

can be added. An application of the resolution rule derives the empty clause An empty clause shows that the negation of the conclusion is a complete contradiction, hence the negation of the conclusion is invalid or false or the assertion is completely valid or true.

Steps for Resolution

EEE.   Convert the given statements in Predicate/Propositional Logic

FFF.   Convert these statements into Conjunctive Normal Form

GGG.   Negate the Conclusion (Proof by Contradiction)

HHH.   Resolve using a Resolution Tree (Unification)

Steps to Convert to CNF (Conjunctive Normal Form)

III. Every sentence in Propositional Logic is logically equivalent to a conjunction of disjunctions of literals.

A sentence expressed as a conjunction of disjunctions of literals is said to be in Conjunctive normal Form or CNF.

JJJ. Eliminate implication _→'

KKK.   a → b = ~a v b

LLL.   ~ (a ^ b) = ~ a v ~ b …………DeMorgan'sLaw

MMM.   ~ (a v b) = ~ a ^ ~ b ………... DeMorgan'sLaw

NNN.   ~ (~a) = a

Eliminate Existential Quantifier _∃'

To eliminate an independent Existential Quantifier, replace the variable by a Skolemconstant. This process is called as Skolemization.

Example: ∃y: President (y)

Here _y' is an independent quantifier so we can replace _y' by any name (say – George Bush).

So, ∃y: President (y) becomes President (George Bush).

To eliminate a dependent Existential Quantifier we replace its variable by SkolemFunction that accepts the value of _x' and returns the corresponding value of _y.'

Example: ∀x : ∃y : father_of(x, y)

Here _y' is dependent on _x', so we replace _y' by S(x).

To eliminate the Universal Quantifier, drop the prefix in PRENEX NORMAL FORM i.e. just drop ∀and the sentence then becomes in PRENEX NORMAL FORM. Eliminate AND _^'

a ^ b splits the entire clause into two separate clauses i.e. a and b

(a v b) ^ c splits the entire clause into two separate clauses a v b and c (a ^ b) v c splits the clause into two clauses i.e. a v c and b v c

To eliminate _^' break the clause into two, if you cannot break the clause, distribute the OR _v' and then break the clause.

Problem Statement:

OOO.   Ravi likes all kind of food.

PPP.   Apples and chicken are food

QQQ.   Anything anyone eats and is not killed is food

RRR.   Ajay eats peanuts and is still alive

SSS.   Rita eats everything that Ajay eats

Prove by resolution that Ravi likes peanuts using resolution.

Step 1: Converting the given statements into Predicate/Propositional Logic

   i. ∀x : food(x) → likes (Ravi, x)

TTT.   food (Apple) ^ food (chicken)
UUU.   ∀a : ∀b: eats (a, b) ^ ~killed (a) → food (b)

VVV.   eats (Ajay, Peanuts) ^ alive (Ajay)
WWW.   ∀c : eats (Ajay, c) → eats (Rita, c)

XXX.   ∀d : alive(d) → ~killed (d)

YYY. ∀e: ~killed(e) → alive(e)

Conclusion: likes (Ravi, Peanuts)

Step 2: Convert into CNF

i. ~food(x) v likes (Ravi, x)

ZZZ. Food (apple)

AAAA. Food (chicken)

BBBB. ~ eats (a, b) v killed (a) v food (b)

CCCC. Eats (Ajay, Peanuts)

DDDD. Alive (Ajay)

EEEE. ~eats (Ajay, c) V eats (Rita, c)

FFFF. ~alive (d) v ~ killed (d)

GGGG. Killed (e) v alive (e)
Conclusion: likes (Ravi, Peanuts)
Negate the conclusion
~ likes (Ravi, Peanuts)

~ likes (Ravi, Peanuts)   ~food(x) v likes (Ravi, x)
                                    x | peanuts
    ~food (peanuts)   ~ eats (a, b) v killed (a) v food (b)
                                    b | peanuts
    ~eats (a, peanuts) v killed (a)   eats (Ajay, peanuts)
                                                    a | Ajay
            Killed (Ajay)   ~alive(d) v ~killed (d)
                                                    d | Ajay
                    ~alive (Ajay)   alive (Ajay)
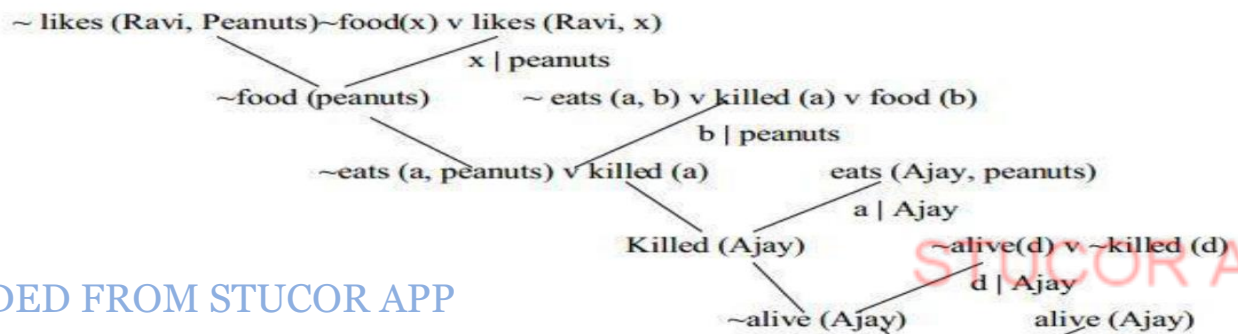
Uses of Resolution in Today's World

       a. Used widely in AI.

       b. Helps in the development of computer programs to automate reasoning and theorem proving

5. Explain the forward chaining process in detail with example? What is the need of incremental chaining?Dec-09,May-10,Dec-10,Dec–14,Dec–13,Dec-16,May–12,May-15,May-17,Dec-18,May-18

FORWARD CHAINING

Forward chaining working from the facts to a conclusion. Sometimes called the data driven approach. To chain forward, match data in working memory against conditions of rules in the rule-base. Starts with the facts, and sees what rules apply (and hence what should be done) given the facts.
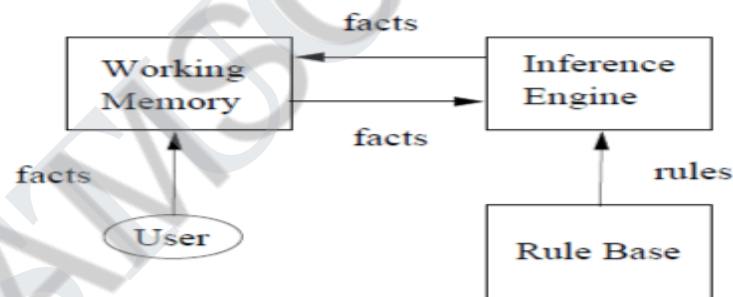
WORKING

Facts are held in a working memory. Condition-action rules represent actions

to take when specified facts occur in working memory. Typically the actions involve

adding or deleting facts from working memory.

STEPS

•To chain forward, match data in working memory against 'conditions' of rules in the rule base.

•When one of them fires, this is liable to produce more data.

•So the cycle continues up to conclusion.

Example

•Here are two rules:

•If corn is grown on poor soil, then it will get blackfly.

•If soil hasn't enough nitrogen, then it is poor soil.

Forward chaining algorithm

Repeat

Collect the rule whose condition matches a fact in WM.

Do actions indicated by the rule.

(add facts to WM or delete facts from WM)

Until problem is solved or no condition match

Apply on the Example 2 extended (adding 2 more rules and 1 fact)

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Rule R4 : IF dry THEN ADD switch_on_humidifier

Rule R5 : IF sprinklers_on THEN DELETE dry

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F2 : Dry [Given]

Now, two rules can fire (R2 and R4)

Rule R4 ADD humidifier is on [from F2]

ADD smoky [from F1]

ADD fire [from F2 by R1]

ADD switch_on_sprinklers [by R3]

Rule R2

[followed by

sequence of

actions]

DELEATE dry, ie

humidifier is off a conflict ! [by R5]

6.Discuss backward chaining algorithm?May-10,Dec-10,May-15,May–13,Dec-16,May-17,Dec-18

BACKWARD CHAINING

Backward chaining: working from the conclusion to the facts. Sometimes

called the goal-driven approach. Starts with something to find out, and looks for rules

that will help in answering it goal driven.

Steps in BC

•To chain backward, match a goal in working memory against 'conclusions' of

rules in the rule-base.

•When one of them fires, this is liable to produce more goals.

•So the cycle continues

Example

•Same rules:

•If corn is grown on poor soil, then it will get blackfly.

•If soil hasn't enough nitrogen, then it is poor soil.

■ Backward chaining algorithm

Prove goal G

If G is in the initial facts , it is proven.

Otherwise, find a rule which can be used to conclude G, and

try to prove each of that rule's conditions.

Encoding of rules

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

Example 1

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : IF fire THEN switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Example 2

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : IF fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

     Fact F2 : hot [Given]

Example 3 : A typical Forward Chaining

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F4 : smoky [from F1 by R2]

Fact F2 : fire [from F2, F4 by R1]

Fact F6 : switch_on_sprinklers [from F2 by R3]

Example 4 : A typical Backward Chaining
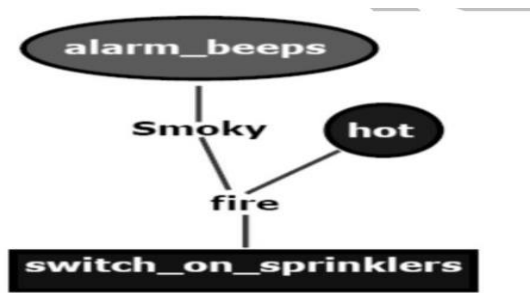
Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If _fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

7. Explain the algorithm for computing most general unifiers.   May - 13

   REFER Qno 5

8. Write short notes on unification? (6)   Dec-12,**Dec -15, Dec – 14,May-19,May-15**
   REFER Qno 5

9. Illustrate the use of first order logic to represent knowledge.   May-12,Dec - 14,May-14,May – 15, Dec-16,Dec-18

The best way to find usage of First order logic is through examples. The examples can be taken from some simple domains. In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge. Assertions and queries in first-order logic Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions. For example, we can assert that John is a king and that kings are persons: Where KB is knowledge base. TELL(KB, $\forall x$ King(x) => Person(x)). We can ask questions of the knowledge base using AS K. For example,

$$ASK(KB, King(John))$$

returns true. Questions asked using ASK are called queries or goals ASK(KB,Person(John)) Will return true. (ASK KBto find whther Jon is a king) ASK(KB, $\exists x$ person(x)) The kinship domain The first example we consider is the domain of family relationships, or kinship. This domain includes facts such as "Elizabeth is the mother of Charles" and "Charles is the father of William7' and rules such as "One's grandmother is the mother of one's parent." Clearly, the objects in our domain are people. We will have two unary predicates, Male and Female. Kinship relations- parenthood, brotherhood, marriage, and so on-will be represented by binary predicates: Parent, Sibling, Brother, Sister, Child, Daughter,Son, Spouse, Husband, Grandparent, Grandchild, Cousin, Aunt, and Uncle. We will use functions for Mother and Father.

10. Explain dempster shafer theory with an example.May-19

   Ruled based approach

   Rule-Based system architecture consists *a set of rules*, *a set of facts*, and an *inference engine*.

Types of Rules

Three types of rules are mostly used in the Rule-based production systems.

HHHH. **Knowledge Declarative Rules :**

These rules state all the facts and relationships about a problem. Example :

IF inflation rate declines

THEN the price of gold goes down.

These rules are a part of the knowledge base.

IIII. **Inference Procedural Rules**

These rules advise on how to solve a problem, while certain facts are known. Example :

IF the data needed is not in the system
THEN request it from the user.

These rules are part of the inference engine.

JJJJ. **Meta rules**

These are rules for making rules. Meta-rules reason about which rules should be considered for firing.

Example :

IF the rules which do not mention the current goal in their premise, AND there are rules which do mention the current goal in their premise, THEN the former rule should be used in preference to the latter. Meta-rules direct reasoning rather than actually performing reasoning.

Meta-rules specify which rules should be considered and in which order they

should be invoked. FACTS :They represent the real world information

Inference Engine

The inference engine uses one of several available forms of inferencing. By inferencing means the method used in a knowledge-based system to process the stored knowledge and supplied data to produce correct conclusions.

Example

How old are you?

Subtract the year you were born in from 2014.

The answer will either be exactly right,

Or one year short.

Dempster/Shafer theory

KKKK. The Dempster-Shafer theory, also known as the theory of belief functions, is a generalization of the Bayesian theory of subjective probability.

LLLL. The Bayesian theory requires probabilities for each question of interest, belief functions allow us to base degrees of belief for one question on probabilities for a related question.

MMMM. These degrees of belief may or may not have the mathematical properties of probabilities; how much they differ from probabilities will depend on how closely the two questions are related.

NNNN. The Dempster-Shafer theory owes its name to work by A. P. Dempster (1968) and Glenn Shafer (1976), but the kind of reasoning the theory uses can be found as far back as the seventeenth century.

OOOO. The theory came to the attention of AI researchers in the early 1980s, when they were trying to adapt probability theory to expert systems.

PPPP. Dempster-Shafer degrees of belief resemble the certainty factors in MYCIN, and this resemblance suggested that they might combine the rigor of probability theory with the flexibility of rule-based systems.

QQQQ. Subsequent work has made clear that the management of uncertainty inherently requires more structure than is available in simple rule-based systems, but the Dempster-Shafer theory remains attractive because of its relative flexibility.

RRRR. The Dempster-Shafer theory is based on two ideas: the idea of obtaining degrees of belief for one question from subjective probabilities for a related question, and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence.

11. What are fuzzy membership functions? Explain them with examples. **May-19**

Fuzzy Set

a. The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut.

b. Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.

c. Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the

set.

SSSS. Fuzzy set theory permits the gradual assessment of the membership of

elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].

Fuzzy Set Theory

Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, True and False, is sometimes

inadequate when describing human reasoning. Fuzzy logic uses the whole interval between 0 (false) and **1** (true) to describe human reasoning.

TTTT.   **Fuzzy logic** is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic.

UUUU.   Fuzzy logic is capable of handling inherently imprecise concepts.

VVVV.   Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets.

Crisp and Non-Crisp Set

WWWW.   The **characteristic function** μ**A(x)** whichhas values **0** ('false') and **1** ('true") only are **crisp sets.**

XXXX.   For Non-crisp sets the characteristic function μ**A(x)** can be defined.

The characteristic function μ**A(x)** for the crisp set is generalized for the Non-crisp sets.

a. This generalized characteristic function μ**A(x)** is called **membership function**.

b. Such Non-crisp sets are called **Fuzzy Sets**.

c. Crisp set theory is not capable of representing descriptions and classifications in many cases, In fact, Crisp set does not provide adequate

representation for most cases.

Representation of Crisp and Non-Crisp Set Example:

Classify students for a basketball team

This example explains the grade of truth value.

YYYY.   **tall students** qualify and **not tall students** do not qualify

ZZZZ.   if students 1.8 m tall are to be qualified, then should we exclude a student who is 1/10" less? Or should we exclude a student who is 1" shorter?

AAAAA.   Non-Crisp Representation to represent the notion of a tall person
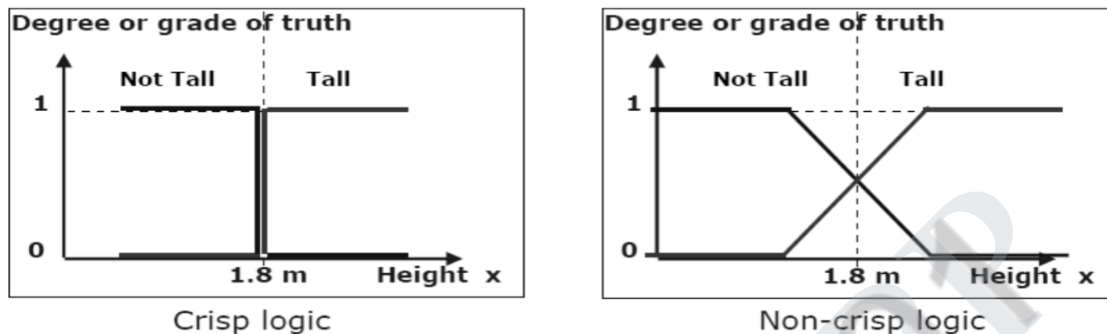
Fig. 1 Set Representation – Degree or grade of truth

BBBBB. A student of height 1.79m would belong to both tall and not tall sets with a particular degree of membership.

CCCCC. As the height increases the membership grade within the tall set would increase whilst the membership grade within the not-tall set would decrease.

Capturing Uncertainty

Instead of avoiding or ignoring uncertainty, Lotfi Zadeh introduced Fuzzy Set theory that captures uncertainty.

DDDDD. A fuzzy set is described by a **membership function μA (x)** of **A**. This membership function associates to each element $x\sigma \in$ **X** a number as μA **($x\sigma$)** in the closed unit interval **[0, 1]**.

EEEEE. The number μA **($x\sigma$)** represents the **degree of membership** of $x\sigma$ in **A**.

FFFFF. The notation used for membership function μA **(x)** of a fuzzy set **A** is

    a. **:X$\in$[0,1]**

GGGGG. Each membership function maps elements of a given universal base set **X** ,

which is itself a crisp set, into real numbers in **[0, 1] .**

Example:
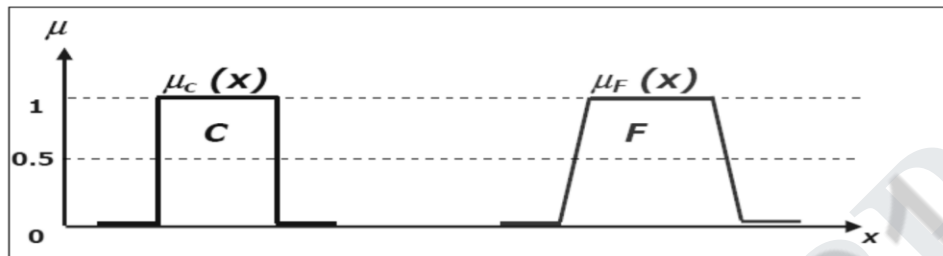
Fig. 2 Membership function of a Crisp set C and Fuzzy set F

HHHHH. In the case of Crisp Sets the members of a set are : either out of the set, with membership of degree " **0** ", or in the set, with membership of degree " **1** ",

a. Therefore, **Crisp Sets ЄFuzzy Sets**

In other words, Crisp Sets are Special cases of Fuzzy Sets.

Examples of Crisp and Non-Crisp Set Example 1: Set of prime numbers (a crisp set)

IIIII. If we consider space **X** consisting of natural numbers ≤ **12** ie **X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}**

JJJJJ. Then, the set of prime numbers could be described as follows.

PRIME = {x contained in X | x is a prime number} = {2, 3, 5, 6, 7, 11} <u>Fuzzy Set</u>

A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, in the interval **[0 , 1]**.

Definition of Fuzzy set

A **fuzzy set A**, defined in the universal space **X**, is a function defined in **X** which assumes values in the range **[0, 1]**.

A fuzzy set **A** is written as a set of pairs **{x, A(x)}** as

A = {{x , A(x)}} , x in the set X

where **x** is an element of the universal space **X**, and **A(x)** is the value of the function **A** for this element.

The value **A(x)** is the **membership grade** of the element **x** in a fuzzy set **A**.

Example : Set **SMALL** in set **X** consisting of natural numbers $\leq$ **to 12**.

Assume: SMALL(1) = 1, SMALL(2) = 1, SMALL(3) = 0.9, SMALL(4) = 0.6, SMALL(5) = 0.4, SMALL(6) = 0.3, SMALL(7) = 0.2, SMALL(8) = 0.1, SMALL(u) = 0 for u >= 9.

Then, following the notations described in the definition above :

Set SMALL = {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Note that a fuzzy set can be defined precisely by associating with each **x** , its grade of membership in **SMALL**.

• **Definition of Universal Space**

Originally the universal space for fuzzy sets in fuzzy logic was defined only on the integers. Now, the universal space for fuzzy sets and fuzzy relations is defined with three numbers.

The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements.

This gives the user more flexibility in choosing the universal space. **Example :** The fuzzy set of numbers, defined in the universal space

X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as **SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]**

Fuzzy Membership

KKKKK.   A fuzzy set **A** defined in the universal space **X** is a function defined in **X** which assumes values in the range **[0, 1]**.

LLLLL.   A fuzzy set **A** is written as a set of pairs **{x, A(x)}**.

A = {{x , A(x)}} , x in the set X

where **x** is an element of the universal space **X**, and

A(x) is the value of the function **A** for this element.

The value **A(x)** is the **degree of membership** of the element **x** in a fuzzy set

A. The Graphic Interpretation of fuzzy membership for the fuzzy sets :

• **Graphic Interpretation of Fuzzy Sets SMALL**

The fuzzy set SMALL of small numbers, defined in the universal space **X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as **SetOption [Fuzzy Set, Universal Space → {1, 12, 1}]**

The Set **SMALL** in set **X** is : **SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3},{7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}**

Therefore **SetSmall** is represented as

SetSmall = FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2}, {8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}] Fuzzy Plot [SMALL, AxesLable → {"X", "SMALL"}]
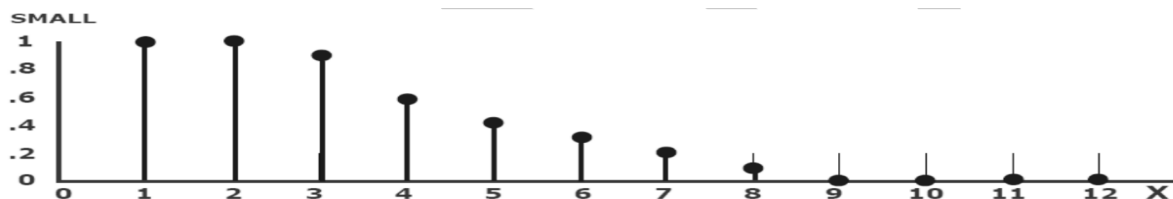
Fig Graphic Interpretation of Fuzzy Sets SMALL

Graphic Interpretation of Fuzzy Sets EMPTY

MMMMM. An empty set is a set that contains only elements with a grade of membership equal to **0**.

Example: Let EMPTY be a set of people, in Minnesota, older than 120.

NNNNN. The Empty set is also called the Null set.

OOOOO. The fuzzy set **EMPTY** , defined in the universal space **X = { xi }** = **{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as **SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]**

PPPPP. The Set **EMPTY** in set **X** is : **EMPTY = FuzzySet {{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0},{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}**

QQQQQ. Therefore **SetEmpty** is represented as **SetEmpty = FuzzySet [{{1,0},{2,0}, {3,0}, {4,0}, {5,0},{6,0}, {7,0},{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**

RRRRR. **FuzzyPlot [EMPTY, AxesLable ={"X", " UNIVERSAL SPACE "}]**

Unit-4:

Two marks question:-

1. define intelligent AI agent?

An AI agent is a computer system that is situated in some environment, and that is capable of anonymous action in this environment in order to meet its design objectives. An intelligent agent that is capable of flexible autonomous action in order to meet its design objectives, where flexibility comprises of three aspects namely,

1.Reactivity-intelligent agents are able to perceive their environment and respond in a timely fashion to changes that occur in it in order to satisfy their prior designed objectives.

2.pro-activeness – intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their priority designed objectives.

3.social ability – intelligent agents are capable of increasing with other agents (and possibly humans) in order to satisfy their prior design objectives.

2. what is a purely reactive agent?

There can agents who decide their action without referring to earlier history. Their decision making is purely based on current situation that has nothing to do with the past situations. Such agents are are termed as purely reactive since they just respond directly to their environment. Formally the behavior of a purely reactive agent can be represent by a function action :S->A

3.what are the characteristics of multi agent systems?

1.each agent has just incomplete information and is restricted in is capabilities.

2. the system control is distributed .

3.data is decentralized.

4.computation is asynchronous

5.multi agent environments are typically open ad have no centralized designer.

6.multi agent environments provide an infrastructure specifying communication and interaction protocols.

7. multi agent environments have agents that are autonomous and distributed, and may be self – interested or cooperative.

4. what is KQML

KQML:

1.the knowledge query and manipulation language (KQML) is a protocol for exchanging information and knowledge among the agents in a multiagent system. It is the best known ACL developed by the ARPA for knowledge sharing initiative

2.it is a high-level, message-oriented communication language and protocol for information exchange, independent of content syntax(KIF,SQL,Prolog…,) and application ontology.

5. what is cooperation?

Cooperation is the practice of working in common with mutually agreed-upon goals and possibly methods, instead of working separately in competition, and in which the success of one is dependent and contingent upon the success of another.

For example, I cant play football alone!

Cooperation requires coordination. Cooperative agent uses various protocols for communication like CONTRACT NET.

<u>16 mark questions</u>

1.Explain briefly the Architecture of Intelligent Agents.

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior.

2.Explain in detail about engineering with agent communication.

Protocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents implementing the roProtocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. This is so because any message sent by one role is received by the other. Thus it is easy to ensure their joint computations generate correct outcomes. But when three or more roles are involved, because any message exchange involves two agents (neglecting multi-cast across roles for now) leaves one or more roles unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this nonexistence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role—in this metaphor , by fleshing out its skeleton—provides methods to process each incoming mes sage and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's specification is very much focused on the p erspective of the role whereas the machine of a protocol describes the progress of a protocol enactment from a neutral perspective.

Protocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. This is so because any message sent by one role is received by the other. Thus it is easy to ensure their joint computations generate correct outcomes. But when three or more roles are involved, because any message exchange involves two agents (neglecting multi-cast across roles for now) leaves one or more roles unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this nonexistence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role—in this metaphor , by fleshing out its skeleton—provides methods to process each incoming mes sage and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's

Programming with Communications

The Java Agent Development Framework (JADE) is a popular platform for de-veloping and running agent-based applications. It implements the FIPA protocols discussed earlier. JADE provides support for the notion of what it terms *behaviors*. A behavior is an abstract specification of an agent that chara cterizes important events such as the receipt of specified messages and the occur rence of timeouts. To implement an agent according to a behavior involves defini ng the methods it specifies as callbacks. In particular, a role skeleton can be implemented by defining the handlers for any incoming methods. The JADE tuto rial online offers comprehensive instructions for building JADE applications.

Modeling Communications

It is not trivial to specify the *right* commitments for particular applications. For instance, Desai et al. [19] show how a scenario dealing with foreign exchange transactions may be formalized in multiple ways using commitments, each with different ramifications on the outcomes. The challenge of sp ecifying the right commitments leads us to the question: *How can we guide software engineers in creating appropriate commitment-based specifications?*

Such guidance is often available for operational approaches such as state ma-chines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 3.7 shows two common patterns expressed as (partial) state machines, which can aid software engineers in specifying opera-tional interactions. Here, *b* and *s* are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.

Figure 3.7: Example operational pattern

By contrast, commitment protocols abstract away from operational details,

focusing on the meanings of messages, not their flow. Clearly , operational patterns such as the above would not apply to the design of commitment protocols. What kinds of patterns would help in the design of commitment protocols? By and large, they would need to be *business patterns*—characterizing requirements, not operations—that emphasize meanings in terms of commitment s. In contrast with Figure 3.7, these patterns describe what it *means* to make, accept, reject, or update an offer, not when to

send specific messages.Business patterns support specifying business protocols. These patterns are motivated by the following principles.

Autonomy compatibility Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed would cause hidden cou-pling.

Explicit meanings The meaning ought to be made public, not hidden within agent implementations.

Business Patterns

Business patternspattern encode the common ways in which businesses engage each other. Below is an example of the *compensation* pattern.

• COMPENSATION

Intent To compensate the creditor in case of commitment cancellation or viola-tion by the debtor.

Motivation It is not known in advance whether a party will fulfill its comm it-ments; compensation commitments provide some assurance to the creditor in case of violations.

Implementation $Compensate(x, y, r, u, p)$ means $Create(x, y, violated(x, y, r, u), p)$.

Example $Compensate(mer, cus, price, item, discount)$; it means that the merchant will offer the customer a discount on the next purchase if the item is paid for but not delivered.

Consequences A commitment (even a compensation commitment) should ide-ally be supported by compensation; however, at some level, the only re-course is escalation to the surrounding business *context*—for example, the local jurisdiction [51].

Enactment Patterns

Whereas a business pattern describes the meaning of communication, an enact-ment pattern describes the conditions under which an agent should enact a busi-ness pattern, that is, *when* to undertake the corresponding communication. A locus of such enactments may serve as the basic agent skeleton.

- COUNTER OFFER

Intent One party makes an offer to another, who responds with a modifi ed offer of its own.

Motivation Essential for negotiation.

When Let $C(x, y, r, u)$ be the commitment corresponding to the original of-fer. Making a counteroffer would amount to creating the commitment $C(y, x, u', r')$ such that $u' \vdash u$ and $r \vdash r'$, in other words, if the consequent is strengthened and the antecedent is weakened. An alternative implemen-tation includes doing *Release*(*x*, *y*, *r*, *u*) in addition.

Example Let's say $C(EBook, Alice, \$12, BNW)$ holds. Alice can make the counter offer $C(Alice, EBook, BNW \wedge Dune, \$12)$ meaning that she wants *Dune* in addition to *BNW* for the same price.

Consequences When $u \equiv u'$ and $r \equiv r'$, the counter offer amounts to a mutual commitment.

Semantic Antipatterns

Semantic antipatterns identify forms of representation and reasoning to be avoided because they conflict with the autonomy of the participants o r with a logical basis for commitments.

- COMMIT ANOTHER AS DEBTOR

Intent An agent creates a commitment in which the debtor is another agent.

Motivation To capture delegation, especially in situations where the delegator is in a position of power of over the delegatee.

Example Consider two sellers EBook and BookWorld. EBook sends

$Create(BookWorld, Alice, \$12, BNW)$ to Alice, which violated BookWorld's autonomy.

Consequences A commitment represents a public undertaking by the debtor. A special case is when $x = z$. That is, x unilaterally makes itself the creditor.

Criteria Failed y's autonomy is not respected.

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments. In the above example, BookWorld could have a standing commitment with EBook to accept delegations. EBook can then send a delegate "instruction" to BookWorld upon which BookW orld com-mits to Alice.

The above are some examples of patterns. For a more exhaustive list of pat-terns, see [16].

Communication-Based Methodologies

Because of the centrality of agent communication to multiagent systems, a number of methodologies for designing and implementing multiagent systems are based on communications. We point out a few such methodologies in the further read-ings section.

The common idea behind these methodologies is to identify the communica-tions involved in the system being specified and to state the m eanings of such communications. The main protocol concepts are roles, messages, and message meanings. Below we briefly outline the high-level considera tions involved in de-signing a protocol.

- Identify stakeholder requirements.

- Identify the roles involved in the interaction. Let's say the roles identified are customer, merchant, shipper, and banker.

- If a suitable protocol is available from a repository, then choose it and we're done. After all, one of key benefits of protocols is reusabili ty. For instance, suppose the stakeholders wanted to design a purchase protocol. If the pro-tocol of Table 3.1 fits their requirements, we're done.

- Often the required protocol may be obtained by *composing* existing proto-cols. For example, the desired protocol could potentially be obtained by combining *Ordering*, *Payment*, and *Shipping* protocols.

- Sometimes the protocol or parts of its may need to be written up from scratch. Identify the communications among the roles. For example, there would be messages between the customer and the merchant that would per-tain to ordering items. The messages between the customer and bank would pertain to payment, and so on.

- Identify how the messages would affect their commitments. For example, the *Offer* message could be given a meaning similar to the one in Table 3.1. The customer's payment to the bank would effectively discharge his com-mitment to pay the merchant. Similarly, the delivery of the goods by the shipper would effectively discharge the merchant's commitment to pay, and so on.

3.Describe the traditional software Engineering approaches.

Traditional Software Engineering Approaches

We referred above to low-level distributed computing protocols as a way to ex-plain architectures in general. We argued that we need to consider multiagent systems and high-level protocols as a way to specify architectures that yield in-teroperability at a level closer to application needs. However, traditional software engineering arguably addresses the challenges of interoperability too. Would it be possible to adopt software engineering techniques as a basis for dealing with agent communication?

The above view has received a significant amount of attention in the litera-ture. Partly because of the apparent simplicity of traditional techniques and largely because of their familiarity to researchers and practitioners alike, the traditional techniques continue to garner much interest in the agents community.

> The traditional techniques leave the formulation of the message syntax open—a message could be any document and in common practice is an XML document. And, they disregard the application meaning of the messages involved. Instead, these techniques focus on the operational details of communication, mostly con-centrating on the occurrence and ordering of messages.

Thus a protocol may be specified in terms of a finite state machi ne that de-scribes its states and legal transitions from a centralized perspective. Formally, this may be done in a variety of ways, including state machines [58, 8], Petri Nets [18], statecharts [24], UML sequence diagrams [35], process algebras such as the pi-calculus [9], and logic-based or declarative approaches [47, 54]. All of these approaches specify a set of message occurrences and orderings that are deemed to capture the protocol being specified. We discuss a few of thes e below.

The above-mentioned traditional representations have the advantage of there

> being a number of formal tools for verifying and even validating specifications written in those representations. Thus a protocol designer would be able to deter-mine if a protocol in question would satisfy useful properties such as termination. Implementing the endpoints or agents to satisfy such specifi cations is generally quite straightforward. Checking compliance with the specification is also concep-tually straightforward. As long as the messages observed respect the ordering and occurrence constraints given by a protocol, the enactment is correct with respect to the protocol; otherwise, an enactment is not correct.

> However, the value of such tools is diminished by the fact that in the traditional representations there is no clear way to describe the meanings of the interactions. In other words, these approaches lack an independent application-centric standard of correctness. For example, let us suppose that a protocol happens to specify that a merchant ships the goods to the customer and then the customer pays. Here, if the customer happens to pay first, that would be a violation of the protocol. In informal terms, we should not care. It should be the customer's internal decision whether to pay first. If the customer does (taking the risk of p aying first or losing bank interest on the money paid), that is the customer's prerogative. However, given the traditional, operational specification, any such deviation from the stated protocol is equally unacceptable. Notice that it may in fact be in the customer's interest to pay first, for example, to include the expense in t he current year's tax deductions. But we have no way of knowing that.

> Instead, if the protocol could be specified in terms of the mea nings of the communications involved, we would naturally express the intuition that all we expect is that the customer eventually pays or that the customer pays no later than some other crucial event. If the customer fails to pay, that would be a violation. But if the customer pays early, so much the better.

Choreographies

The service-oriented computing literature includes studies of the notion of a *choreography*. A choreography is a specification of the message flow among th e participants. Typically, a choreography is specified in ter ms of *roles* rather than the participants themselves. Involving roles promotes reusability of the chore-ography specification. Participants *adopt* roles, that is, bind to the roles, in the choreography.

A choreography is a description of an interaction from a shared or, more prop-erly, a *neutral* perspective. In this manner, a choreography is distinguished from a specification of a *workflow* , wherein one party drives all of the other parties. The

latter approach is called an *orchestration* in the services literature.

An advantage of adopting a neutral perspective, as in a choreography, is that it better applies in settings where the participants retain their autonomy: thus it is important to state what each might expect from the others and what each might offer to the others. Doing so promotes loose coupling of the components: centralized approaches could in principle be equally loosely coupled but there is a tendency associated with the power wielded by the central party to make the other partners fit its mold. Also, the existence of the cen tral party and the resulting regimentation of interactions leads to implicit dependencies and thus tight coupling among the parties.

A neutral perspective yields a further advantage that the overall computation becomes naturally distributed and a single party is not involved in mediating all information flows. A choreography is thus a way of specifying and building dis-tributed systems that among the conventional approaches most closely agrees with the multiagent systems way of thinking. But important distinctions remain, which we discuss below.

WS-CDL [57] and ebBP [25] are the leading industry supported choreography standardization efforts. WS-CDL specifies choreographies as message exchanges among partners. WS-CDL is based on the pi-calculus, so it has a formal oper-ational semantics. However, WS-CDL does not satisfy important criteria for an agent communication formalism. First, WS-CDL lacks a theory of the mean-ings of the message exchanges. Second, when two or more messages are per-formed within a given WS-CDL choreography, they are handled sequentially by default, as in an MSC. Third, WS-CDL places into a choreography actions that would be private to an agent, such as what it should do upon receiving a message. Fourth, for nested choreographies, WS-CDL relies upon local decision-making by an agent, such as whether to forward a request received in one choreography to another [50].

Sequence Diagrams

The most natural way to specify a protocol is through a message sequence chart (MSC), formalized as part of UML as Sequence Diagrams [28]. The roles of a protocol correspond to the lifelines of an MSC; each edge connecting two life-lines indicates a message from a sender to a receiver. Time flo ws downward by convention and the ordering of the messages is apparent from the chart. MSCs support primitives for grouping messages into blocks. Additional primitives in-clude alternatives, parallel blocks, or iterative blocks. Although we do not use

MSCs extensively, they provide a simple way to specify agent communication protocols.

FIPA (the Foundation of Intelligent Physical Agents) is a standards body, now part of the IEEE Computer Society, that has formulated agent communication standards. FIPA defines a number of interaction protocols. T hese protocols in-volve messages of the standard types in FIPA. Each FIPA protocol specifies the possible ordering and occurrence constraints on messages as a UML Sequence Diagram supplemented with some informal documentation.

Figure 3.2 shows the FIPA Request Interaction Protocol in FIPA's variant of the UML Sequence Diagram notation [26]. This protocol involves two roles, an

INITIATOR and a PARTICIPANT.The INITIATOR sends a *request* to the PARTIC-

IPANT, who either responds with a *refuse* or an *agree*. In the latter case, it fol-lows up with a detailed response, which could be a *failure*, an *inform-done*, or an *inform-result*. The PARTICIPANT may omit the *agree* message unless the INITIA-TOR asked for a notification.

The FIPA Request protocol deals with the operational details of when certain messages may or must be sent. It does not address the meanings of the messages themselves. Thus it is perfectly conventional in this regard. Where it deviates from traditional distributed computing is in the semantics it assigns to the mes-sages themselves, which we return to below. However, the benefit of having a protocol is apparent even in this simple example: it identifi es the roles and their mutual expectations and thus decouples the implementations of the associated agents from one another.

State Machines

Figure 3.3 shows a state machine between two roles, merchant (mer) and customer (cus) as a state machine. The transitions are labeled with messages; the prefix mer, cus indicates a message from the merchant to the customer, and cus, mer indicates a message from the customer to the merchant. This state machine supports two executions. One execution represents the scenario where the customer rejects the merchant's offer. The other execution represents the scenario where the customer accepts the offer, following which the merchant

and the customer exchange the item and the payment for the item. In the spirit of a state machine, Figure 3.3 does not reflect the internal policies based upon which the custom er accepts an offer.

Consider the state machine in Figure 3.4. The dotted paths indicate two addi-tional executions that are not supported by the state machine in Figure 3.3. The executions depict the scenarios where the customer sends the payment upon re-

Figure 3.2: FIPA Request Interaction Protocol, from the FIPA specification [26], expressed as a UML Sequence Diagram.

ceiving an offer and after sending an accept, respectively. These additional execu-tions are just as sensible as the original ones. However, in the context of the state machine in Figure 3.3, these executions are trivially *noncompliant*. The reason is that checking compliance with choreographies is purely syntactical—the mes-sages have to flow between the participants exactly as prescr ibed. Clearly, this curbs the participants' autonomy and flexibility.

We can attempt to ameliorate the situation by producing ever larger FSMs that include more and more paths. However, doing so complicates the implementation
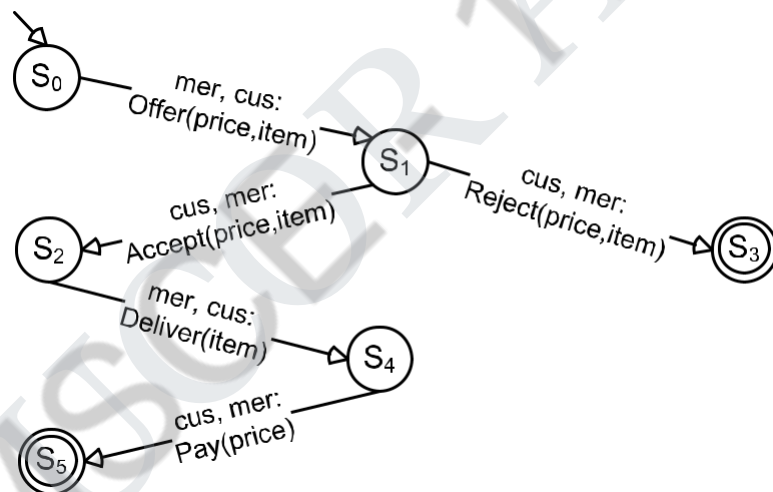

Figure 3.3: A protocol specified as a state machine.

of agents and the task of comprehending and maintaining protocols, while not supporting any real runtime flexibility. Further, any selec tion of paths will remain arbitrary.
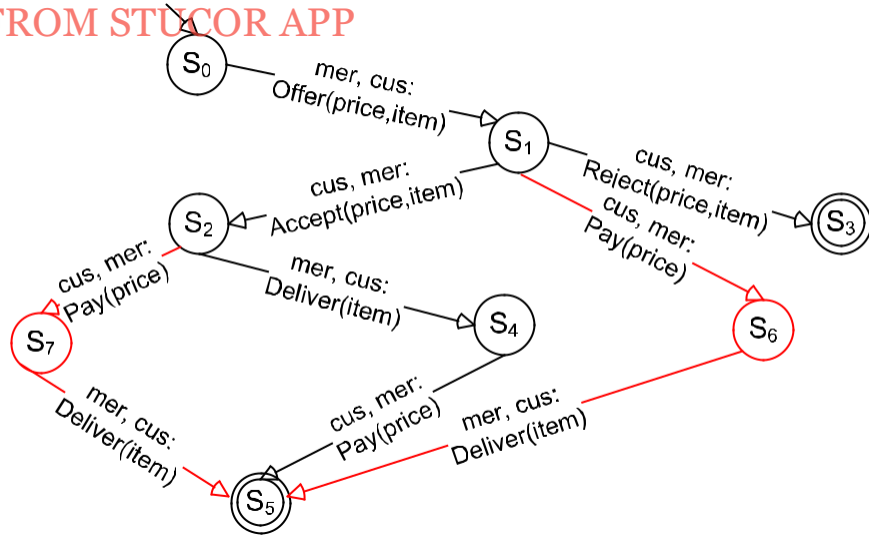
Figure 3.4: An alternative, more flexible state machine.

Evaluation with Respect to MAS

Traditional software engineering approaches for specifying protocols are opera-tional in nature. Instead of specifying the meaning of a communication, they spec-ify the flow of information among agents. The lack of meaning l eads to the fol-lowing observations about protocols produced following traditional approaches.

Software engineering Because the protocols specify the set of possible enact-ments at a low level of abstraction, any but the most trivial are difficult to design and maintain. It is difficult to map the business req uirements of stakeholders to the protocols produced.

Flexibility Agents have little flexibility at runtime; the protocols ess entially dic-tate agent skeletons. Any deviation from a protocol by an agent, no matter how sensible from a business perspective, is a violation. Further, to enable interoperation, the protocols are specified so that they pro duce lock-step synchronization among agents, which also limits flexibilit y.

Compliance Checking an agent's compliance with the protocol is easy: compu-tationally, it is akin to verifying whether a string is accepted by an FSM. However, that ease comes at the expense of flexibility.

4. Explain the Following.

a) Traditional AI Approaches

The traditional AI approaches to agent communication begin from the opposite extreme. These approaches presume that the agents are constructed based on cog-nitive concepts, especially, beliefs, goals, and intentions. Then they specify the communication of such agents in terms of how the communication relates to their cognitive representations.

The AI approaches came from two related starting points, which has greatly affected how they were shaped. The first starting point was of human-computer interaction broadly and natural language understanding specifically. The latter

includes the themes of discourse understanding from text or speech, and speech understanding. What these approaches had in common was that they were geared toward developing a tool that would assist a user in obtaining information from a database or performing simple transactions such as booking a train ticket. A key functionality of such tools was to infer what task the user needed to perform and to help the user accordingly. These tools maintained a user model and were

configured with a domain model upon which they reasoned via he uristics to de-termine how best to respond to their user's request, and potentially to anticipate the user's request.

Such a tool was obviously cooperative: its raison d'ˆetre was to assist its user and failure to be cooperative would be simply unacceptable. Further, it was an appropriate engineering assumption that the user was cooperative as well. That is, the tool could be based on the idea that the user was not purposefully misleading it, because a user would gain nothing in normal circumstances by lying about his needs and obtaining useless responses in return.

As the tools became more proactive they began to be thought of as agents. Further, in some cases the agents of different users could communicate with one another, not only with their users. The agents would maintain their models of their users and others based on the communications exchanged. They could make strong inferences regarding the beliefs and intentions of one another, and act and communicate accordingly. These approaches worked for their target setting. To AI researchers, the approaches these agents used for communicating with users and other agents appeared to be applicable for agent communication in general.

The second body of work in AI that related to agent communication came from the idea of building distributed knowledge-based systems (really just ex-pert systems with an ability to communicate with each other). The idea was that each agent would include a reasoner and a knowledge representation and com-munication was merely a means to share such knowledge. Here, too, we see the same two assumptions as for the human interaction work. First, that the member agents were constructed with the same knowledge representations. Second, that the agents were largely cooperative with each other.

KQML

Agent communication languages began to emerge in the 1980s. These were usu-ally specific to the projects in which they arose, and typical ly relied on the specific internal representations used within the agents in those projects.

Somewhat along the same lines, but with some improved generality, arose the Knowledge Query and Manipulation Language or KQML. KQML was created by the DARPA Knowledge Sharing Effort, and was meant to be an adjunct to the other work on knowledge representation technologies, such as ontologies. KQML sought to take advantage of a knowledge representation based on the construct of a knowledge base, such as had become prevalent in the 1980s. Instead of a specific internal representation, KQML assumes that each agent maintains a knowledge

base described in terms of knowledge (more accurately, belief) assertions. KQML proposed a small number of important primitives, such as *query* and

tell. The idea was that each primitive could be given a semantics based on the effect it had on the knowledge bases of the communicating agents. Specifically, an agent would send a *tell* for some content only if it believed the content, that is, the content belonged in its knowledge base. And, an agent who received a *tell* for some content would insert that content into its knowledge base, that is, it would begin believing what it was told.

Even though KQML uses knowledge as a layer of abstraction over the detailed data structures of the internal implementation of agent, it turns out to be overly restricted in several ways. The main assumption of KQML is that the commu-nicating agents are cooperative and designed by the same designers. Thus the designers would make sure that an agent sent a message, such as a *tell*, only under the correct circumstances and an agent who received such a message could imme-diately accept its contents. When the agents are autonomous, they may generate spurious messages—and not necessarily due to malice.

KQML did not provide a clear basis for agent designers to choose which of the message types to use and how to specify their contents. As a result, designers all too often resolved to using a single message type, typically *tell*, with all meanings encoded (usually in some ad hoc manner) in the contents of the messages. That is, the approach is to use different *tell* messages with arbitrary expressions placed within the contents of the messages.

The above challenges complicated interoperability so that it was in general difficult if not impossible for agents developed by differen t teams to be able to successfully communicate with one another.

FIPA ACL

We discussed the FIPA interaction protocols in Section 3.2. FIPA has also pro-duced the FIPA ACL, one of the motivations behind which was to address the challenges with KQML. A goal for the FIPA ACL or Agent Communication Lan-guage was to specify a definitive syntax through which intero perability among agents created by different developers could be facilitated. In addition, to ensure interoperability, the FIPA ACL also specified the semantics of the primitives. Like KQML's, the FIPA ACL semantics is mentalist, although it has a stronger basis in logic. The FIPA ACL semantics is based on a formalization of the cognitive concepts such as the beliefs and intentions of agents.

Beliefs and intentions are suitable abstractions for designing and implement-ing agents. However, they are highly unsuitable as a basis for an agent communi-cation language. A communication language supports the interoperation of two or more

agents. Thus it must provide a basis for one agent to compute an abstraction of the local state of another agent. The cognitive concepts provide no such basis in a general way. They lead to the internal implementations of the interacting agents to be coupled with each other. The main reason for this is that the cognitive con-cepts are definitionally internal to an agent. For example, c onsider the case where a merchant tells a customer that a shipment will arrive on Wednesday. When the shipment fails to arrive on Wednesday, would it be any consolation to the cus-tomer that the merchant sincerely believed that it was going to? The merchant could equally well have been lying. The customer would never know without an audit of the merchant's databases. In certain legal situations, such audits can be performed but they are far from the norm in business encounters.

One might hope that it would be possible to infer the beliefs and intentions of another party, but it is easy to see with some additional refle ction that no unique characterization of the beliefs and intentions of an agent is possible. In the above example, maybe the merchant had a sincere but false belief; or, maybe the mer-chant did not have the belief it reported; or, maybe the merchant was simply un-sure but decided to report a belief because the merchant also had an intention to consummate a deal with the customer.

It is true that if one developer implements all the interacting agents correctly, the developer can be assured that an agent would send a particular message only in a particular internal state (set of beliefs and intentions). However such a mul-tiagent system would be logically centralized and would be of severely limited value.

It is worth pointing out that the FIPA specifications have end ed up with a split personality. FIPA provides the semiformal specification of an agent management system, which underlies the well-regarded JADE system [7]. FIPA also provides definitions for several interaction protocols (discussed i n Section 3.2), which are also useful and used in practice, despite their limitations. FIPA provides a formal semantics for agent communication primitives based on cognitive concepts, which gives a veneer of rigor, but is never used in multiagent systems.

Evaluation with Respect to MAS

The traditional AI approaches are mentalist, which render them of limited value for multiagent systems.

Software engineering The AI approaches offer high-level abstractions, which is a positive. However, because the abstractions are mentalist, the approaches cannot be applied to the design of multiagent systems except in the restricted case where one developer designs all the agents (as explained above). Since there is no interaction in the sense of interaction Further, recall the discus-sion from Section 2.2 regarding the unsuitability of a small set of primitives. Both KQML and FIPA suffer from this problem.

Flexibility The flexibility of agents is severely curtailed because of re strictions on when agents can send particular communications.

Compliance It is impossible for an observer to verify the cognitive state of an agent. Hence verifying agent compliance (for example, if the agent has the requisite cognitive state for sending a particular message) is impossible.

b) Commitment-Based Multiagent Approach

Commitment-Based Multiagent Approaches

In contrast with the operational approaches, commitment protocols give primacy to the *business meanings* of service engagements, which are captured through the participants' *commitments* to one another [60], [11, 52, 22, 56], [20]. Computa-tionally, each participant is modeled as an *agent*; interacting agents carry out a service engagement by creating and manipulating commitments to one another.

Commitments

A commitment is an expression of the form C(*debtor*, *creditor*, *antecedent*, *consequent*), where *debtor* and *creditor* are agents, and *antecedent* and *consequent* are propo-sitions. A commitment C(x, y, r, u) means that x is committed to y that if r holds, then it will bring about u. If r holds, then C(x, y, r, u) is *detached*, and the commitment C(x, y, $\top$, u) holds ($\top$ being the constant for truth). If u holds, then

the commitment is *discharged* and does not hold any longer. All commitments are *conditional*; an unconditional commitment is merely a special case where the antecedent equals $\top$. Examples 1–3 illustrate these concepts. In the examples below, EBook is a bookseller, and Alice is a customer.)

Example 1 (Commitment) C(*EBook*, *Alice*, $12, *BNW*) means that EBook com-mits to Alice that if she pays $12, then EBook will sends her the book *Brave New World*.

Example 2 (Detach) If Alice makes the payment, that

holds, then C(*EBook*, *Alice*, $12, *BNW*) is detached. In

C(*EBook*, *Alice*, $12, *BNW*) $\wedge$ $12 \Rightarrow$ C(*EBook*, *Alice*, $\top$, *BNW*).

is, if $12
words,

other

Example 3 (Discharge) If EBook sends the book, that is, if *BNW* holds, then both C(*EBook*, *Alice*, $12, *BNW*) and C(*EBook*, *Alice*, $\top$, *BNW*) are discharged. In other words, *BNW* $\Rightarrow \neg$C(*EBook*, *Alice*, $12, *BNW*)$\wedge \neg$C(*EBook*, *Alice*, $\top$, *BNW*).

Importantly, commitments can be manipulated, which supports flexibility. The commitment operations [45] are listed below; CREATE, CANCEL, and RELEASE are two-party operations, whereas DELEGATE and ASSIGN are three-party opera-tions.

- CREATE (x, y, r, u) is performed by x, and it causes C(x, y, r, u) to hold.

- CANCEL (x, y, r, u) is performed by x, and it causes C(x, y, r, u) to not hold.

- RELEASE (x, y, r, u) is performed by y, and it causes C(x, y, r, u) to not hold.

- DELEGATE (x, y, z, r, u) is performed by x, and it causes C(z, y, r, u) to hold.

- ASSIGN (x, y, z, r, u) is performed by y, and it causes C(x, z, r, u) to hold.

- DECLARE (x, y, r) is performed by x to inform y that the r holds.

DECLARE is not a commitment operation, but may indirectly affect commit-ments by causing detaches and discharges. In relation to Example 2, when Alice informs EBook of the payment by performing DECLARE (*Alice*, *EBook*, \$12), then the proposition \$12 holds, and causes a detach of C(*EBook*, *Alice*, \$12, *BNW*).

Further, a commitment arises in a social or legal context. The context defines the rules of encounter among the interacting parties, and often serves as an arbiter in disputes and imposes penalties on parties that violate their commitments. For example, eBay is the context of all auctions that take place through the eBay marketplace; if a bidder does not honor a payment commitment for an auction that it has won, eBay may suspend the bidder's account.

A formal treatment of commitments and communication based on commit-ments is available in the literature [48, 15].

Table 3.1: A commitment protocol.

Offer(*mer*, *cus*, *price*, *item*) means CREATE (*mer*, *cus*, *price*, *item*)

Accept(*cus*, *mer*, *price*, *item*) means CREATE (*cus*, *mer*, *item*, *price*)

Reject(*cus*, *mer*, *price*, *item*) means RELEASE (*mer*, *cus*, *price*, *item*)

Deliver(*mer*, *cus*, *item*) means DECLARE (*mer*, *cus*, *item*)

Pay(*cus*, *mer*, *price*) means DECLARE (*cus*, *mer*, *price*)

Commitment Protocol Specification

Table 3.1 shows the specification of a commitment protocol be tween a merchant and a customer (omitting sort and variable declarations). It simply states the meanings of the messages in terms of the commitments arising between the merchant and customer. For instance, the message *Offer*(*mer*, *cus*, *price*, *item*) means the creation of the commitment C(*mer*, *cus*, *price*, *item*), meaning the merchant commits to delivering the item if the customer pays the price; *Reject*(*cus*, *mer*, *price*, *item*) means a release of the commitment; *Deliver*(*mer*, *cus*, *item*) means that the proposition *item* holds.

Figure 3.5(A) shows an execution of the protocol and Figure 3.5(B) its mean-ing in terms of commitments. (The figures depicting executio ns use a notation similar to UML interaction diagrams. The vertical lines are agent lifelines; time flows downward along the lifelines; the arrows depict messag es between the agents; and any point where an agent sends or receives a message is annotated with the commitments that hold at that point. In the figures, i nstead of writing CREATE, we write *Creat e*. We say that the *Creat e* message realizes the CREATE operation. Likewise, for other operations and DECLARE.) In the figure, the mer-chant and customer role are played by EBook and Alice, respectively; c_B and $c_{U\,B}$ are the commitments C(*EBook*, *Alice*, $12, BNW) and C(*EBook*, *Alice*, ⊤, *BNW*) respectively.

Evaluation with Respect to MAS

Compliance Protocol enactments can be judged correct as long as the parties in-volved do not violate their commitments. A customer would be in violation if he keeps the goods but fails to pay. In this manner, commitments support business-level compliance and do not dictate specific opera tionalizations [22].

> Figure 3.5: Distinguishing message syntax and meaning: two views of the same enactment.

Flexibility The above formulation of correctness enhances flexibility o ver tradi-tional approaches by expanding the operational choices for each party [13]. For example, if the customer substitutes a new way to make a payment or elects to

pay first, no harm is done, because the behavior is co rrect at the business level. And, the merchant may employ a new shipper; the customer may return damaged goods for credit; and so on. By contrast, without busi-ness meaning, exercising any such flexibility would result i n noncompliant executions.

Software Engineering Commitments offer a high-level abstraction for capturing business interactions. Further, a commitment-based approach accommo-dates the autonomy of the participants in the natural manner: socially, an agent is expected to achieve no more than his commitments. Commitments thus also support loose coupling among agents. Commitment-based ap-proaches offer a compelling alternative to the traditional SE approaches de-scribed in Section 3 for building systems comprised of autonomous agents.

Figure 3.6 shows some of the possible enactments based on the proto-col in Table 3.1. The labels c_A and c_{UA} are C(*Alice*, *EBook*, *BNW*, $*12*$) and C(*Alice*, *EBook*, $\top$, $*12*$), respectively. Figure 3.6(B) shows the enactment where the book and payment are exchanged in Figure 3.3. Figures 3.6(A) and (C) show the additional executions supported in Figure 3.4; Figure 3.6(D) reflects a new execution that we had not considered before, one where Alice sends an *Accept* even before receiving an offer. All these executions are compliant executions in terms of commitments, and are thus supported by the protocol in Table 3.1.
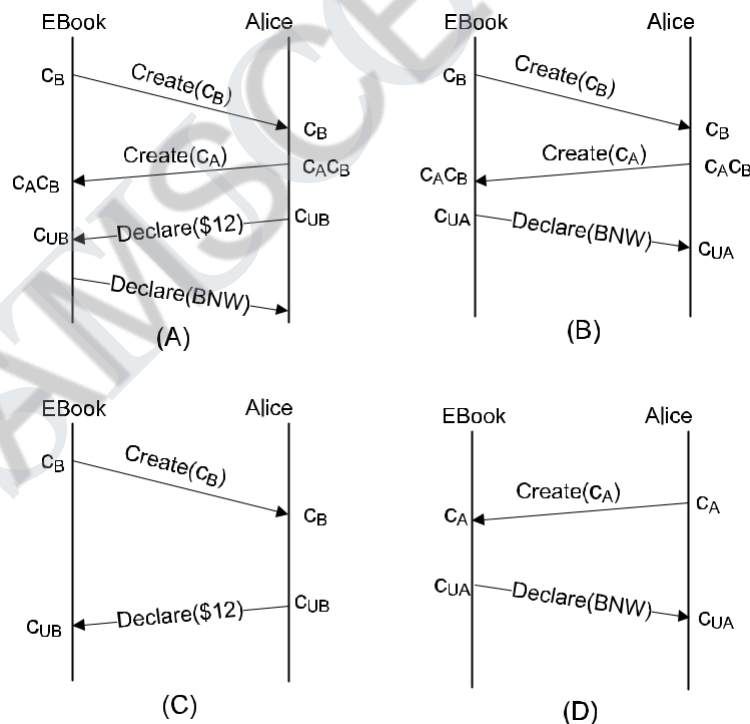

Figure 3.6: Flexible enactment.

Table 3.2: Comparison of agent communication approaches.

	Traditional SE	Traditional AI	Commitment Protocols
Abstraction	control flow	mentalist	business relationship
Compliance	lexical basis	unverifiable	semantic basis
Flexibility	low	low	high
Interoperability	message-level	integration	business-level

Table 3.2 summarizes the three approaches.

5. Explain in detail about the Agent Communication.

Protocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents implementing the roProtocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents

implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. This is so because any message sent by one role is received by the other. Thus it is easy to ensure their joint computations generate correct outcomes. But when three or more roles are involved, because any message exchange involves two agents (neglecting multi-cast across roles for now) leaves one or more roles unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this nonexistence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role—in this metaphor , by fleshing out its skeleton—provides methods to process each incoming mes sage and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's specification is very much focused on the p erspective of the role whereas the machine of a protocol describes the progress of a protocol enactment from a neutral perspective.

Protocols support the development of distributed systems. A natural way to ap-ply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role spec-ifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles sug-gests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those inc luded in the proto-col and which feature in the individual role specifications. In other words, we would like the agents implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. This is so because any message sent by one role is received by the other. Thus it is easy to ensure their joint computations generate correct outcomes. But when three or more roles are involved, because any message exchange involves two agents (neglecting multi-cast across roles for now) leaves one or more roles unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this nonexistence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role—in this metaphor , by fleshing out its skeleton—provides methods to process each incoming mes sage and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's

Programming with Communications

The Java Agent Development Framework (JADE) is a popular platform for developing and running agent-based applications. It implements the FIPA protocols discussed earlier. JADE provides support for the notion of what it terms *behaviors*. A behavior is an abstract specification of an agent that chara cterizes important events such as the receipt of specified messages and the occur rence of timeouts. To implement an agent according to a behavior involves defini ng the methods it specifies as callbacks. In particular, a role skeleton can be implemented by defining the handlers for any incoming methods. The JADE tuto rial online offers comprehensive instructions for building JADE applications.

Modeling Communications

It is not trivial to specify the *right* commitments for particular applications. For instance, Desai et al. [19] show how a scenario dealing with foreign exchange transactions may be formalized in multiple ways using commitments, each with different ramifications on the outcomes. The challenge of sp ecifying the right commitments leads us to the question: *How can we guide software engineers in creating appropriate commitment-based specifications?*

Such guidance is often available for operational approaches such as state machines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 3.7 shows two common patterns expressed as (partial) state machines, which can aid software engineers in specifying opera-tional interactions. Here, *b* and *s* are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.
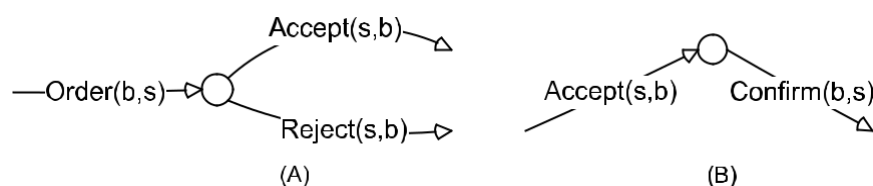

Figure 3.7: Example operational patterns

By contrast, commitment protocols abstract away from operational details,

focusing on the meanings of messages, not their flow. Clearly , operational patterns such as the above would not apply to the design of commitment protocols. What kinds of patterns would help in the design of commitment protocols? By and large, they would need to be *business patterns*—characterizing requirements, not operations— that emphasize meanings in terms of commitment s. In contrast with Figure 3.7, these patterns describe what it *means* to make, accept, reject, or update an offer, not when to send specific messages.

Business patterns support specifying business protocols. These patterns are motivated by the following principles.

Autonomy compatibility Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed would cause hidden cou-pling.

Explicit meanings The meaning ought to be made public, not hidden within agent implementations.

Business Patterns

Business patternspattern encode the common ways in which businesses engage each other. Below is an example of the *compensation* pattern.

- COMPENSATION

Intent To compensate the creditor in case of commitment cancellation or viola-tion by the debtor.

Motivation It is not known in advance whether a party will fulfill its comm it-ments; compensation commitments provide some assurance to the creditor in case of violations.

Implementation $Compensate(x, y, r, u, p)$ means $Create(x, y, violated(x, y, r, u), p)$.

Example $Compensate(mer, cus, price, item, discount)$; it means that the merchant will offer the customer a discount on the next purchase if the item is paid for but not delivered.

Consequences A commitment (even a compensation commitment) should ide-ally be supported by compensation; however, at some level, the only re-course is escalation to the surrounding business *context*—for example, the local jurisdiction [51].

Enactment Patterns

Whereas a business pattern describes the meaning of communication, an enact-ment pattern describes the conditions under which an agent should enact a busi-ness pattern, that is, *when* to undertake the corresponding communication. A locus of such enactments may serve as the basic agent skeleton.

- COUNTER OFFER

Intent One party makes an offer to another, who responds with a modifi ed offer of its own.

Motivation Essential for negotiation.

When Let C(x, y, r, u) be the commitment corresponding to the original of-fer. Making a counteroffer would amount to creating the commitment C(y, x, u', r') such that $u' \vdash u$ and $r \vdash r'$, in other words, if the consequent is strengthened and the antecedent is weakened. An alternative implemen-tation includes doing *Release*(x, y, r, u) in addition.

Example Let's say C(*EBook*, *Alice*, \$12, *BNW*) holds. Alice can make the counter offer C(*Alice*, *EBook*, *BNW* ∧ *Dune*, \$12) meaning that she wants *Dune* in addition to *BNW* for the same price.

Consequences When $u \equiv u'$ and $r \equiv r'$, the counter offer amounts to a mutual commitment.

Semantic Antipatterns

Semantic antipatterns identify forms of representation and reasoning to be avoided because they conflict with the autonomy of the participants o r with a logical basis for commitments.

- COMMIT ANOTHER AS DEBTOR

Intent An agent creates a commitment in which the debtor is another agent.

Motivation To capture delegation, especially in situations where the delegator is in a position of power of over the delegatee.

Implementation The sender of $Create(y, z, p, q)$ is x (x and y are different agents), thus contravening the autonomy of the y.

Example Consider two sellers EBook and BookWorld. EBook sends $Create(BookWorld, Alice, \$12, BNW)$ to Alice, which violated BookWorld's autonomy.

Consequences A commitment represents a public undertaking by the debtor. A special case is when $x = z$. That is, x unilaterally makes itself the creditor.

Criteria Failed y's autonomy is not respected.

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments. In the above example, BookWorld could have a standing commitment with EBook to accept delegations. EBook can then send a delegate "instruction" to BookWorld upon which BookWorld com-mits to Alice.

The above are some examples of patterns. For a more exhaustive list of pat-terns, see [16].

Communication-Based Methodologies

Because of the centrality of agent communication to multiagent systems, a number of methodologies for designing and implementing multiagent systems are based on communications. We point out a few such methodologies in the further read-ings section.

The common idea behind these methodologies is to identify the communica-tions involved in the system being specified and to state the m eanings of such communications. The main protocol concepts are roles, messages, and message

meanings. Below we briefly outline the high-level considera tions involved in designing a protocol.

- Identify stakeholder requirements.

- Identify the roles involved in the interaction. Let's say the roles identified are customer, merchant, shipper, and banker.

- If a suitable protocol is available from a repository, then choose it and we're done. After all, one of key benefits of protocols is reusabili ty. For instance, suppose the stakeholders wanted to design a purchase protocol. If the pro-tocol of Table 3.1 fits their requirements, we're done.

- Often the required protocol may be obtained by *composing* existing proto-cols. For example, the desired protocol could potentially be obtained by combining *Ordering*, *Payment*, and *Shipping* protocols.

- Sometimes the protocol or parts of its may need to be written up from scratch. Identify the communications among the roles. For example, there would be messages between the customer and the merchant that would per-tain to ordering items. The messages between the customer and bank would pertain to payment, and so on.

- Identify how the messages would affect their commitments. For example, the *Offer* message could be given a meaning similar to the one in Table 3.1. The customer's payment to the bank would effectively discharge his com-mitment to pay merchant. Similarly, the delivery of the goods by the shipper would effectively discharge the merchant's commitment to pay, and so on.

Unit-5:

Two marks

1.what is understanding?

By understanding something we mean that, to transform one representation into another which indicate the interpretation of 'something' which is being precieved.The newly transformed representation is chosen such that it corresponds to a set of actions that are available and can be performed with respect to event.

For examples,when at canteen counter you ask for coffee,the system first understands that you want coffee ,and then it finds all available types of coffee and present you the menu where as

if you ask for hot coffee then system should understand and present you with only hot coffee menu.

2.what are terms used in communication?

Termed used in communication:

Speech act- this is the action by which agent produces language.The speech here,need not always be the general talking:But the speech can be emailing,skywriting or any type of communication established through signs.

Speaker-An agent that is capable of producing language is called as speaker or hearer or utterance.

Word-It is any kind of conventional communicative sign.

3.Define formal language string grammar.

Formal language is defined as set of strings having well defined rules for string formation.

Grammar-Is a finite set of rules that specifies a language.

4.What are over generating and under generating grammar.

The over generating grammar;The grammar that generates sentences that are not grammatically valid in English language is called as overgeneraating grammar.

The undergenerating grammar:The grammar that do not generate sentences that are grammatically valid in English language is called as undergenerating grammar.

5.Define definite clause grammar.

1.The notation X->YZ…translates as

$Y(S_1) \wedge Z(S_2) … => X(S_1+S_2+…)$

2.The notation X ->Y|Z| … translates as

$Y(S) \wedge Z(S) … X(S)$

3.In either of the preceeding rules, any non-terminal symbol ¥ can be augmented with one or more arguments.The variable can be a constant , or a function of arguments.In the translation , these arguments precede the string argument.

For example:

NP (case) is translated as NP (case ,S_1).

4.The notation {P (…)} can appear on the right -hand side of a rule and translates verbatim into P(…).This allows the grammar writer to insert a test for P(…).without having the automatic string argumented added.

5.The notation X-> word,is translated as X([word]).

5.What is robotics?

The robot Institute of America (1979) a robot is :

" A reprogrammable asa well as multifunctional manipulator designed to move material,parts,tools,or specialized devices through various programmed motions for the performance of a variety of tasks".

According to Webster a robot is:

" An automatic device that performs functions normally ascribed to humans or machine in the form of a human."Another simpler and general definition of robot is : A robot is a programmable electromechanical system that can gather information from its environment (sense) , use this information and make decisions (plan) and follow instructions to do work (act) From this definition we can determine what is not a robot.A robot must have a programme and sensors to interact with its environment and to decide its actions.However,it is unfortunate that the market for robots is crowded with toy models with no robotic sometimes function residing with real robots in the same manufacture's line.

"A robot is re-programmable, multi-functional manipulator designed to move material ,parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks."

6.What are characteristics of robot?

Machines – mechanical devices designed for doing work.

Automatic-operations which are executed without external help.

Reprogrammable -multifunctional and flexible: not restricted to one job but can be programmed to perform many jobs (nearly all robot systems contain a reprogrammable computer).

7.What are advantages of robot?

Advantages:

1.Greater flexibility, re-programmability, kinematics dexterity

2.Greater response time to inputs than humans

3.Improved product quality

4.Maximize capital intensive equipment in multiple work shifts

5.Accidentreduction

6.Reduction of hazardous exposure for human workers

7.Automatic less susceptible to work stoppages

8.What are disadvantages of robot?

Disadvantages:

1.Replacement of human labor

2.Greater unemployment

3.Significant retraining costs for both unemployed and users of new technology

4.Advertised technology does not always disclose some of the hidden disadvantages.

5.Hidden costs because of the associated technology that must be purchase and intergrated into a functioning cell.Typically ,a functioning cell will cost 3-10 times the cost of the robot.

5.What are the limitations of robotics?

Assembly dexterity does not match that of human beings,particularly where eye-hand coordination required.

1.Payload to robot weight ratio is poor ,often less than 5%.

2.Robot structural configuration may limit joint movement.

3.Work volumes can be constrained by parts by parts or tooling /sensors added to thr robot .

4.Robot repeatability/accuracy can contain the range of potential applications.

5,Closed architectures of modern robot systems make it difficult to automat.

16 mark questions:

1.Explain in detail about the moving?
MOVING

In the real world, of course, this is not the case. Robots have inertia and cannot execute arbitrary paths except at arbitrarily slow speeds. In most cases, the robot gets to exert forces rather than specify positions. This section discusses methods for calculating these forces.

Dynamics and control

Here introduced the notion of dynamic state, which extends the kinematic state of a robot by its velocity. For example, in addition to the angle of a robot joint, the dynamic state also captures the rate of change of the angle, and possibly even its momentary acceleration. The transition model for a dynamic state representation includes the effect of forces on this rate of change. Such models are typically expressed via differential equations, which are equations that relate a quantity (e.g., a kinematic state) to the change of the quantity over time (e.g., velocity). In principle, we could have chosen to plan robot motion using dynamic models, instead of our kinematic models. Such a methodology would lead to superior robot performance, if we could generate the plans. However, the dynamic state has higher dimension than the kinematic space, and the curse of dimensionality would render many motion planning algorithms inapplicable for all but the most simple robots. For this reason, practical robot system often rely on simpler kinematic path planners. A common technique to compensate for the limitations of kinematic plans is to use a separate mechanism, a controller, for keeping the robot on track. Controllers are techniques for generating robot controls in real time using feedback from the environment, so as to achieve a control objective. If the objective is to keep the robot on a preplanned path, it is often referred to as a reference controller and the path is called a reference path. Controllers that optimize a global cost function are known as optimal controllers. Optimal policies for continuous MDPs are, in effect, optimal controllers. On the surface, the problem of keeping a robot on a prespecified path appears to be relatively straightforward. In practice, however, even this seemingly simple problem has its pitfalls.
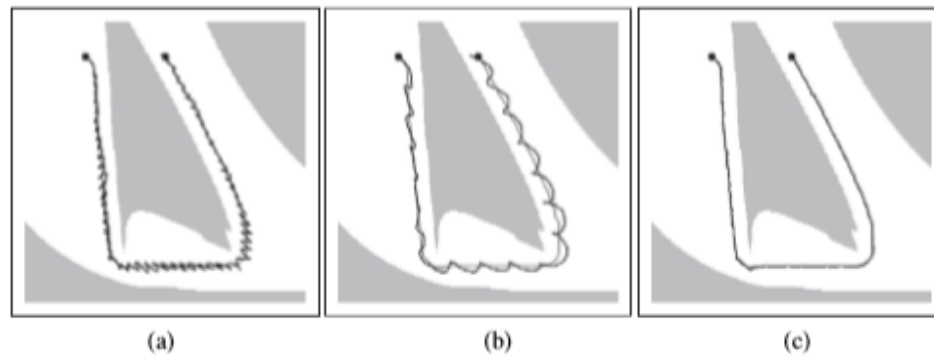
Figure 25.22 Robot arm control using (a) proportional control with gain factor 1.0, (b) proportional control with gain factor 0.1, and (c) PD (proportional derivative) control with gain factors 0.3 for the proportional component and 0.8 for the differential component. In all cases the robot arm tries to follow the path shown in gray.

Figure 25.22(a) illustrates what can go wrong; it shows the path of a robot that attempts to follow a kinematic path. Whenever a deviation occurs—whether due to noise or to constraints on the forces the robot can apply—the robot provides an opposing force whose magnitude is proportional to this deviation. Intuitively, this might appear plausible, since deviations should be compensated by a counterforce to keep the robot on track. However, as Figure 25.22(a) illustrates, our controller causes the robot to vibrate rather violently. The vibration is the result of a natural inertia of the robot arm: once driven back to its reference

position the robot then overshoots, which induces a symmetric error with opposite sign. Such vershooting may continue along an entire trajectory, and the resulting robot motion is far from desirable. Before we can define a better controller, let us formally describe what went wrong. Controllers that provide force in negative proportion to the observed error are known as P controllers. The letter 'P' stands for proportional, indicating that the actual control is proportional to the error of the robot manipulator. More formally, let y(t) be the reference path, parameterized by time index t. The control at generated by a P controller has the form:

at = KP (y(t) − xt) .

Here xt is the state of the robot at time t and KP GAIN PARAMETER is a constant known as the gain parameter of the controller and its value is called the gain factor); Kp regulates how strongly the controller

corrects for deviations between the actual state xt and the desired one y(t). In our example,KP = 1. At first glance, one might think that choosing a smaller value for KP would remedy the problem. Unfortunately, this is not the case. Figure 25.22(b) shows a trajectory for KP = .1, still exhibiting oscillatory behavior. Lower values of the gain parameter may simply slow down the oscillation, but do not solve the problem. In fact, in the absence of friction, the P controller is essentially a spring law; so it will oscillate indefinitely around a fixed target location. Traditionally, problems of this type fall into the realm of control theory, a field of increasing importance to researchers in AI. Decades of research in this field have led to a large number of controllers that are superior to the simple control law given above. In particular, a STABLE reference controller is said to be stable if small perturbations lead to a bounded error between **STRICTLY STABLE the robot and the reference signal. It is said to be strictly stable if it is able to return to and**then stay on its reference path upon such perturbations. Our P controller appears to be stable but not strictly stable, since it fails to stay anywhere near its reference trajectory. The simplest controller that achieves strict stability PD

CONTROLLER in our domain is a PD controller. The letter 'P' stands again for proportional, and 'D' stands for derivative. PD controllers are described by the following equation:

$$a_t = K_P (y(t) - x_t) + K_D \frac{\partial (y(t) - x_t)}{\partial t}. \quad (25.2)$$

As this equation suggests, PD controllers extend P controllers by a differential component, which adds to the value of at a term that is proportional to the first derivative of the error $y(t) - x_t$ over time. What is the effect of such a term? In general, a derivative term dampens the system that is being controlled. To see this, consider a situation where the error $(y(t)-x_t)$ is changing rapidly over time, as is the case for our P controller above. The derivative of this error will then counteract the proportional term, which will reduce the overall response to the perturbation. However, if the same error persists and does not change, the derivative will

vanish and the proportional term dominates the choice of control. Figure 25.22(c) shows the result of applying this PD controller to our robot arm, using as gain parameters $K_P = .3$ and $K_D = .8$. Clearly, the resulting path is much smoother, and does not exhibit any obvious oscillations. PD controllers do have failure modes, however. In particular, PD controllers may fail to regulate an error down to zero, even in the absence of external perturbations. Often such a situation is the result of a systematic external force that is not part of the model. An autonomous car driving on a banked surface, for example, may find itself systematically pulled to one side. Wear and tear in robot arms cause similar systematic errors. In such situations, an over-proportional feedback is required to drive the error closer to zero. The solution to this

problem lies in adding a third term to the control law, based on the integrated error over time:

$$a_t = K_P (y(t) - x_t) + K_I \int (y(t) - x_t)dt + K_D \frac{\partial (y(t) - x_t)}{\partial t}. \quad (25.3)$$

Here K_I is yet another gain parameter. The term $+ \int (y(t)-x_t)dt$ calculates the integral of the error over time. The effect of this term is that long-lasting deviations between the reference signal and the actual state are corrected. If, for example, x_t is smaller than $y(t)$ for a long period of time, this integral will grow until the resulting control at forces this error to shrink. Integral terms, then, ensure that a controller does not exhibit systematic error, at the expense of increased danger of oscillatory behavior. A controller with all three terms is called a PID controller (for proportional integral derivative). PID controllers are widely used in industry, for a variety of control problems.

Potential-field control

We introduced potential fields as an additional cost function in robot motion planning, but they can also be used for generating robot motion directly, dispensing with the path planning phase altogether. To achieve this, we have to define an attractive force that pulls the robot towards its goal configuration and a repellent potential field that pushes the robot away from obstacles. Such a potential field is shown in Figure 25.23. Its single global minimum is

Figure 25.23 Potential field control. The robot ascends a potential field composed of repelling forces asserted from the obstacles and an attracting force that corresponds to the goal configuration. (a) Successful path. (b) Local optimum.
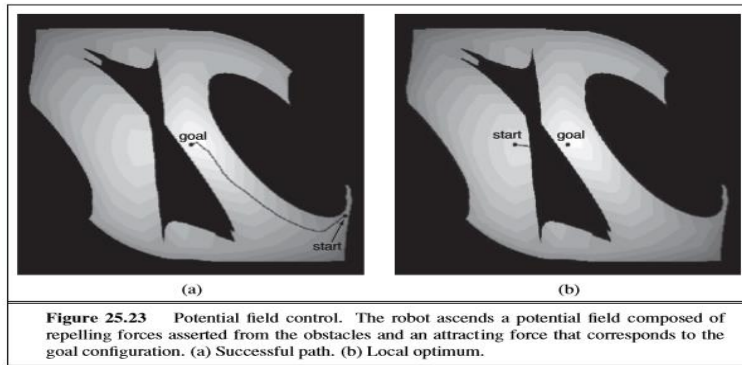
the goal configuration, and the value is the sum of the distance to this goal configuration and the proximity to obstacles. No planning was involved in generating the potential field shown in the figure. Because of this, potential fields are well suited to real-time control. Figure 25.23(a) shows a trajectory of a robot that performs hill climbing in the potential field. In many applications, the potential field can be calculated efficiently for any given configuration. Moreover, optimizing the potential amounts to calculating the gradient of the potential 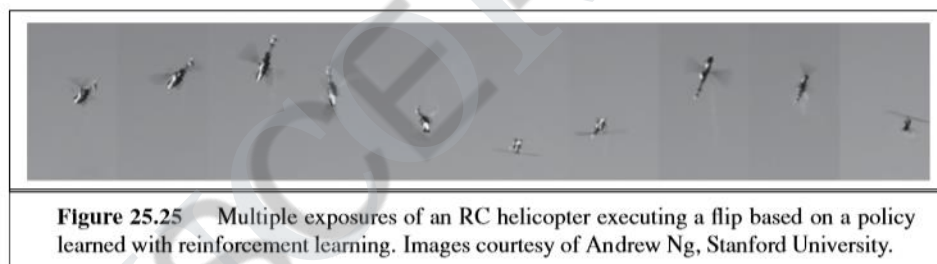for the present robot configuration. These calculations can be extremely efficient, especially when compared to path-planning algorithms, all of which are exponential in the dimensionality of the configuration space (the DOFs) in the worst case. The fact that the potential field approach manages to find a path to the goal in such an efficient manner, even over long distances in configuration space, raises the question as to whether there is a need for planning in robotics at all. Are potential field techniques sufficient, or were we just lucky in our example? The answer is that we were indeed lucky. Potential fields have many local minima that can trap the robot. In Figure 25.23(b), the robot approaches the obstacle by simply rotating its shoulder joint, until it gets stuck on the wrong side of the obstacle. The potential field is not rich enough to make the robot bend its elbow so that the arm fits under the obstacle. In other words, potential field control is great for local robot motion but sometimes we still need global planning. Another important drawback with potential fields is that the forces they generate dependonly on the obstacle and robot positions, not on the robot's velocity. Thus, potential field control is really a kinematic method and may fail if the robot is moving quickly.

Figure 25.24 (a) Genghis, a hexapod robot. (b) An augmented finite state machine (AFSM) for the control of a single leg. Notice that this AFSM reacts to sensor feedback: if a leg is stuck during the forward swinging phase, it will be lifted increasingly higher.
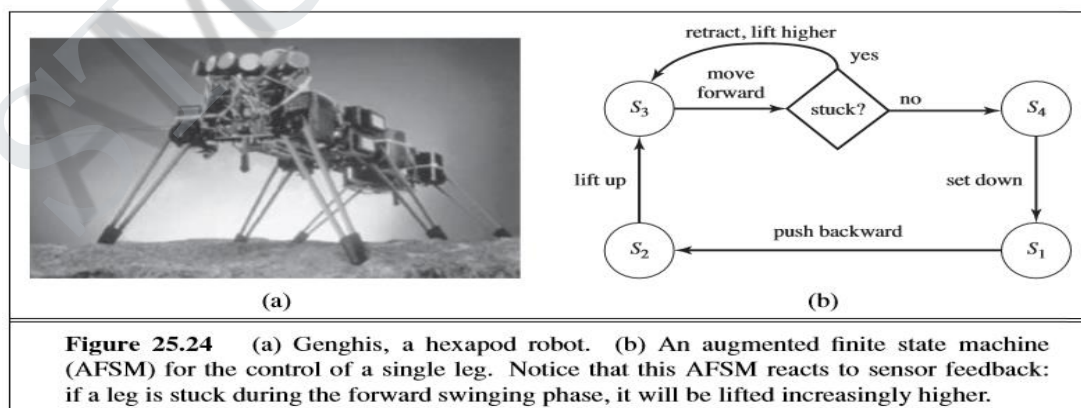
25.6.3 Reactive control

So far we have considered control decisions that require some model of the environment for constructing either a reference path or a potential field. There are some difficulties with this approach. First, models that are sufficiently accurate are often difficult to obtain, especially in complex or

remote environments, such as the surface of Mars, or for robots that have few sensors. Second, even in cases where we can devise a model with sufficient accuracy, computational difficulties and localization error might render these techniques impractical. In some cases, a reflex agent architecture using reactive control is more appropriate. REACTIVECONTROL Forexample, picture alegged robot that attempts tolift alegoveran obstacle. Wecould give this robot a rule that says lift the leg a small height h and move it forward, and if the leg encounters an obstacle, move it back and start again at a higher height. You could say that h is modeling an aspect of the world, but we can also think of h as an auxiliary variable of the robot controller, devoid of direct physical meaning. One such example is the six-legged (hexapod) robot, shown in Figure 25.24(a), designed for walking through rough terrain. The robot's sensors are inadequate to obtain models of the terrain for path planning. Moreover, even if we added sufficiently accurate sensors, the twelve degrees of freedom (two for each leg) would render the resulting path planning problem computationally intractable. It is possible, nonetheless, to specify a controller directly without an explicit environmental model. (We have already seen this with the PD controller, which was able to keep a complex robot arm ontarget without an explicit model of the robot dynamics; itdid, however, require a reference path generated from a kinematic model.) For the hexapod robot we first choose a gait, or pattern of movement of the limbs. One statically stable gait is to first moveGAIT the right front, right rear, and left center legs forward (keeping the other three fixed), and then move the other three. This gait works well on flat terrain. On rugged terrain, obstacles may prevent a leg from swinging forward. This problem can be overcome by a remarkably simple control rule: when a leg's forward motion is blocked, simply retract it, lift it higher

Figure 25.25   Multiple exposures of an RC helicopter executing a flip based on a policy learned with reinforcement learning. Images courtesy of Andrew Ng, Stanford University.

and try again. The resulting controller is shown in Figure 25.24(b) as a finite state machine; it constitutes a reflex agent with state, where the internal state is represented by the index of the current machine state (s1 through s4). Variants of this simple feedback-driven controller have been found to generate remarkably robust walking patterns, capable of maneuvering the robot over rugged terrain. Clearly, such a controller is model-free, and it does not deliberate or use search for generating controls. Environmental feedback plays a crucial role in the controller's execution. The software alone does not specify what will actually happen when the robot is placed in an environment. Behavior that emerges through the interplay of a (simple) controller and a (complex) environment is often referred to as emergent behavior. Strictly speaking, all robots discussedEMERGENT BEHAVIOR in this chapter exhibit emergent behavior, due to the fact that no model is perfect. Historically, however, the term has been reserved for control techniques that do not utilize explicit environmental models. Emergent behavior is also characteristic of biological organisms.

25.6.4 Reinforcement learning control

Oneparticularly exciting formof control isbased on the policysearch formof reinforcement learning (see Section 21.5). This work has been enormously influential in recent years, at is has solved challenging robotics problems for which previously no solution existed. An example is acrobatic autonomous helicopter flight. Figure 25.25 shows an autonomous flip of a small RC (radio-controlled) helicopter. This maneuver is challenging due to the highly nonlinear nature of the aerodynamics

involved. Only the most experienced of human pilots are able to perform it. Yet a policy search method (as described in Chapter 21), using only a few minutes of computation, learned a policy that can safely execute a flip every time. Policy search needs an accurate model of the domain before it can find a policy. The input to this model is the state of the helicopter at time t, the controls at time t, and the resulting state attime t+Δt. Thestate of ahelicopter canbe described bythe 3D coordinates of the vehicle, its yaw, pitch, and roll angles, and the rate of change of these six variables. The controls are the manual controls of of the helicopter: throttle, pitch, elevator, aileron, and rudder. All that remains is the resulting state—how are we going to define a model that accurately says how the helicopter responds to each control? The answer is simple: Let an expert human pilot fly the helicopter, and record the controls that the expert transmits over the radio and the state variables of the helicopter. About four minutes of human-controlled flight suffices to build a predictive model that is sufficiently accurate to simulate the vehicle.

What is remarkable about this example is the ease with which this learning approach solves achallenging robotics problem. Thisis one of the manysuccesses of machine learning in scientific fields previously dominated by careful mathematical analysis and modeling.

2. Describe in detail about planning?
PLANNING TO MOVE

All of a robot's deliberations ultimately come down to deciding how to move effectors. The point-to-point motion problem is to deliver the robot or its end effector to a designated target location. A greater challenge is the compliant motion problem, in which a robot moves while being in physical contact with an obstacle. An example of compliant motion is a robot manipulator that screws in a light bulb, or a robot that pushes a box across a table top.

We begin by finding a suitable representation in which motion-planning problems can be described and solved. It turns out that the configuration space—the space of robot states defined by location, orientation, and joint angles—is a better place to work than the original 3D space. The path planning problem is to find a path from one configuration to another in configuration space. We have already encountered various versions of the path-planning problem throughout this book; the complication added by robotics is that path planning involves continuous spaces. There are two main approaches: cell decomposition and skeletonization.

Each reduces the continuous path-planning problem to a discrete graph-search problem. In this section, we assume that motion is deterministic and that localization of the robot is exact.Subsequent sections will relax these assumptions.

Configuration space

Figure 25.14 (a) Workspace representation of a robot arm with 2 DOFs. The workspace is a box with a flat obstacle hanging from the ceiling. (b) Configuration space of the same robot. Only white regions in the space are configurations that are free of collisions. The dot in this diagram corresponds to the configuration of the robot shown on the left.
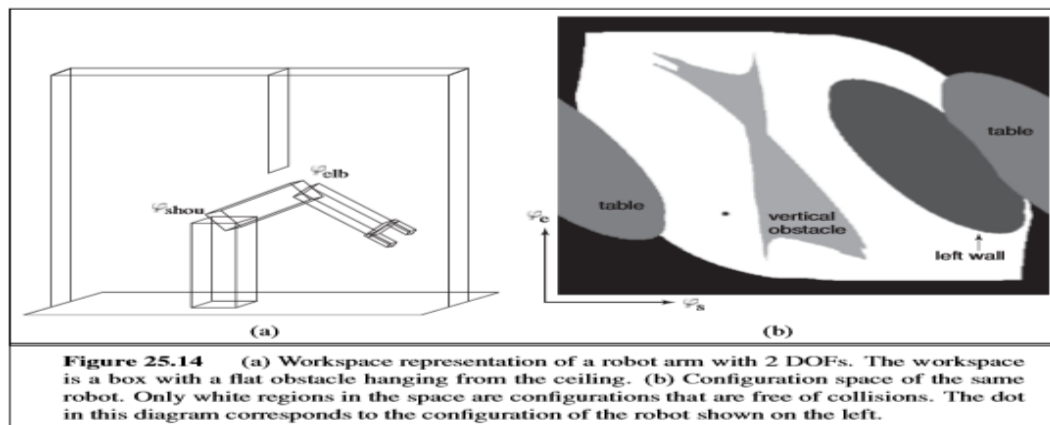
We will start with a simple representation for a simple robot motion problem. Consider the robot arm shown in Figure 25.14(a). It has two joints that move independently. Moving the joints alters the (x, y) coordinates of the elbow and the gripper. (The arm cannot move in the z direction.) This suggests that the robot's configuration can be described by a fourdimensional coordinate: (x_e, y_e) for the location of the elbow relative to the environment and (x_g, y_g) for the location of the gripper. Clearly, these four oordinates characterize the full state of the robot. They constitute what is known as workspace representation, since the coordinates of the robot are specified in the same coordinate system as the objects it seeks to manipulate (or to avoid). Workspace representations are well-suited for collision checking, especially if the robot and all objects are represented by simple polygonal models.

The problem with the workspace representation is that not all workspace coordinates are actually attainable, even in the absence of obstacles. This is because of the linkage conLINKAGE straints on the space of attainable workspace coordinates. For example, the elbow position (x_e, y_e) and the gripper position (x_g, y_g) are always a fixed distance apart, because they are joined by a rigid forearm. A robot motion planner defined over workspace coordinates faces **the challenge of generating paths that adhere to these constraints. This is particularly tricky** because the state space is continuous and the constraints are nonlinear. It turns out to be easier to plan with a configuration space CONFIGURATION representation. Instead of representing the state of the SPACE robot by the Cartesian coordinates of its elements, we represent the state by a configuration of the robot's joints. Our example robot possesses two joints. Hence, we can represent its state with the two angles ϕ_s and ϕ_e for the shoulder joint and elbow joint, respectively. In the absence of any obstacles, a robot could freely take on any value in configuration space. In particular, when planning a path one could simply connect the present configuration and the target configuration by a straight line. In following this path, the robot would then move its joints at a constant velocity, until a target location is reached. Unfortunately, configuration spaces have their own problems. The task of a robot is usually expressed in workspace coordinates, not in configuration space coordinates. This raises the question of how to map between workspace coordinates and configuration space. Transforming configuration space coordinates into workspace coordinates is simple: it involves a series of straightforward coordinate transformations. These transformations are linear for prismatic joints and trigonometric for revolute joints. This chain of coordinate transformation KINEMATICS is known as kinematics. The inverse problem of calculating the configuration of a robot whose effector location INVERSE is specified in workspace coordinates is known as inverse kinematics. Calculating the inverse kinematics is hard, especially for robots with many DOFs. In particular, the solution is seldom unique.
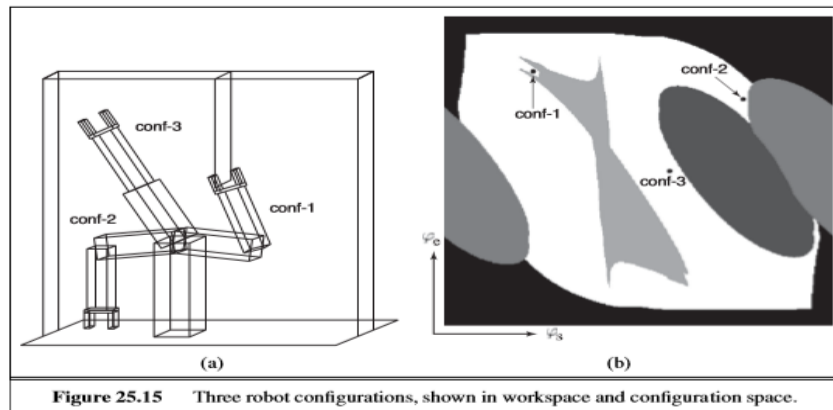
Figure 25.15 Three robot configurations, shown in workspace and configuration space.

Figure 25.14(a) shows one of two possible configurations that put the gripper in the

same location. (The other configuration would has the elbow below the shoulder In general, this two-link robot arm has between zero and two inverse kinematic solutions for any set of workspace coordinates. Most industrial robots have sufficient degrees of freedom to find infinitely many solutions to motion problems. To see how this is possible, simply imagine that we added a third revolute joint to our example robot, one whose rotational axis is parallel to the ones of the existing joints. In such a case, we can keep the

location (but not the orientation!) of the gripper fixed and still freely rotate its internal joints, for most configurations of the robot. With a few more joints (how many?) we can achieve the same effect while keeping the orientation of the gripper constant as well. We have already seen an example of this in the "experiment" of placing your hand on the desk and moving your elbow. The kinematic constraint of your hand position is insufficient to determine the configuration of your elbow. In other words, the inverse kinematics of your shoulder–arm assembly possesses an infinite number of solutions.The second problem with configuration space representations arises from the obstacles that may exist in the robot's workspace. Our example in Figure 25.14(a) shows several such obstacles, including a free-hanging obstacle that protrudes into the center of the robot'sworkspace. In workspace, such obstacles take on simple geometric forms—especially in most robotics textbooks, which tend to focus on polygonal obstacles. But how do they look configurations that a robot may attain, commonly called free space, and the FREE SPACE space of unattainable configurations, called occupied space. The white OCCUPIED SPACE area in Figure 25.14(b) corresponds to the free space. All other regions correspond to occupied space. The different shadings of the occupied space corresponds to the different objects in the robot's workspace; the black region surrounding the entire free space corresponds to configurations in which the robot collides with itself. It is easy to see that extreme values of the shoulder or elbow angles cause such a violation. The two oval-shaped regions on both sides of robot correspond to the table on which the robot is mounted. The third oval region corresponds to the left wall. Finally, the most interesting object in configuration space is the vertical obstacle that hangs from the ceiling and impedes the robot's motions. This object has a funny shape in configuration space: it is highly nonlinear and at places even concave. With a little bit of imagination the reader will recognize the shape of the gripper at the upper left end. We encourage the reader to pause for a moment and study this diagram. The shape of this obstacle is not at all obvious! The dot inside Figure 25.14(b) marks the configuration of the robot, as shown in Figure 25.14(a). Figure 25.15 depicts three additional configurations, both in workspace and in configuration space. In configuration conf-1, the gripper encloses

the vertical obstacle.

Even if the robot's workspace is represented by flat polygons, the shape of the free space can be very complicated. In practice, therefore, one usually probes a configuration space instead of constructing it

explicitly. A planner may generate a configuration and then test to see if it is in free space by applying the robot kinematics and then checking for collisions in workspace coordinates.

25.4.2 Cell decomposition methods

The first CELL approach to path planning uses cell decomposition—that is, it decomposes the free space into a finite number of contiguous regions, called cells. These regions have the important property that the path-planning problem within a single region can be solved by simple means (e.g., moving along a straight line). The path-planning problem then becomes a discrete graph-search problem, very much like the search problems introduced in Chapter 3.

The simplest cell decomposition consists of a regularly spaced grid. Figure 25.16(a) shows a square grid decomposition of the space and a solution path that is optimal for this grid size. Grayscale shading indicates the value of each free-space grid cell—i.e., the cost of the shortest path from that cell to the goal. (These values can be computed by a deterministic form of the VALUE-ITERATION algorithm given in Figure 17.4 on page 653.) Figure 25.16(b) shows the corresponding workspace trajectory for the arm. Of course, we can also use the A∗ algorithm to find a shortest path. Such a decomposition has the advantage that it is extremely simple to implement, but it also suffers from three limitations. First, it is workable only for low-dimensional configuration spaces, because the number of grid cells increases exponentially with d, the number of dimensions. Sounds familiar? This is the curse!dimensionality@of dimensionality. Second, there is the problem of what to do with cells that are "mixed"—that is, neither entirely within free space nor entirely within occupied space. A solution path that includes such a cell may not be a real solution, because there may be no way to cross the cell in the desired direction in a straight line. This would make the path planner unsound. On the other hand, if we insist **that only completely free cells may be used, the planner will be incomplete, because it might**be the case that the only paths to the goal go through mixed cells— especially if the cell size is comparable to that of the passageways and clearances in the space. And third, any path through a discretized state space will not be smooth. It is generally difficult to guarantee thatSKELETONIZATION The second major family of path-planning algorithms is based on the idea of skeletonization. These algorithms reduce the robot's free space to a one-dimensional representation, for which the planning problem is easier. This lower-dimensional representation is called a skeleton of the configuration space. Figure 25.18 shows an example skeletonization: it is a Voronoi graph of the free space—the set of all points that are equidistant to two or more obstacles. To do path planning with a Voronoi graph, the robot first changes its present configuration to a point on the Voronoi graph. It is easy to show that this can always be achieved by a straight-line motion in configuration space. Second, the robot follows the Voronoi graph until it reaches the point nearest to the target configuration. Finally, the robot leaves the Voronoi graph and moves to **the target. Again, this final step involves straight-line motion in configuration space. In this way, the original path-planning problem is reduced to finding a path on the** Voronoi graph, which is generally one-dimensional (except in certain nongeneric cases) and **has finitely many points where three or more one-dimensional curves intersect. Thus, finding** the shortest path along the Voronoi graph is a discrete graph-search problem of the kind discussed in Chapters 3 and 4. Following the Voronoi graph may not give us the shortest path, but the resulting paths tend to maximize clearance. Disadvantages of Voronoi graph techniques are that they are difficult to apply to higher-dimensional configuration spaces, and that they tend to induce unnecessarily large detours when the configuration space is wide open. Furthermore, computing the Voronoi graph can be difficult, especially in configuration space, where the shapes of obstacles can be complex. An alternative to the Voronoi PROBABILISTIC graphs is the probabilistic roadmap, a skeletonization approach that offers more possible routes, and thus deals better with wide-open spaces. Figure 25.18(b) shows an example of a probabilistic roadmap. The graph is created by randomly generating a large number of configurations, and discarding those that do not fall into free space. Two nodes are joined by an arc if it is "easy" to reach

one node from the other–for example, by a straight line in free space. The result of all this is a randomized graph in the

robot's free space. If we add the robot's start and goal configurations to this graph, path planning amounts to a discrete graph search. Theoretically, this approach is incomplete, because a bad choice of random points may leave us without any paths from start to goal. It is possible to bound the probability of failure in terms of the number of points generated and certain geometric properties of the configuration space. It is also possible to direct the generation of sample points towards the areas where a partial search suggests that a good path may be found, working bidirectionally from both the start and the goal positions. With these improvements, probabilistic roadmap planning tends to scale better to high—dimensional configuration spaces than most alternative path-planning techniques. a smooth solution exists near the discrete path. So a robot may not be able to execute the solution found through this decomposition. Cell decomposition methods can be improved in a number of ways, to alleviate some of these problems. The first approach allows further subdivision of the mixed cells— perhaps using cells of half the original size. This can be continued recursively until a path is found that lies entirely within free cells. (Of course, the method only works if there is a way to decide if a given cell is a mixed cell, which is easy only if the configuration space boundaries have relatively simple mathematical descriptions.) This method is complete provided there is

a bound on the smallest passageway through which a solution must pass. Although it focuses most of the computational effort on the tricky areas within the configuration space, it still fails to scale well to high-dimensional problems because each recursive splitting of a cell creates 2d smaller cells. A second way to obtain a complete algorithm is to insist on an exact cell decomposition of the free space. This EXACT CELL method must allow cells to be irregularly shaped where they meet the boundaries of free space, but the shapes must still be "simple" in the sense that it should be easy to compute a traversal of any free cell. This technique requires some quite advanced geometric ideas, so we shall not pursue it further here. Examining the solution path shown in Figure 25.16(a), we can see an additional difficulty that will have to be resolved. The path contains arbitrarily sharp corners; a robot moving at any finite speed could not execute such a path. This problem is solved by storing certain **continuous values for each grid cell. Consider an algorithm which stores, for each grid cell**the exact, continuous state that was attained with the cell was first expanded in the search. Assume further, that when propagating information to nearby grid cells, we use this continuous state as a basis, and apply the continuous robot motion model for jumping to nearby cells. In doing so, we can now guarantee that the resulting trajectory is smooth and can indeed be executed by the robot. One HYBRID A* algorithm that implements this is hybrid A*.

25.4.3 Modified cost functions

Notice that in Figure 25.16, the path goes very close to the obstacle. Anyone who has driven a car knows that a parking space with one millimeter of clearance on either side is not really a parking space at all; for the same reason, we would prefer solution paths that are robust with respect to small motion errors POTENTIAL FIELD This problem can be solved by introducing a potential field. A potential field is a function defined over state space, whose value grows with the distance to the closest obstacle. Figure 25.17(a) shows such a potential field—the darker a configuration state, the closer it is to an obstacle. The potential field can be used as an additional cost term in the shortest-path calculation. This induces an interesting tradeoff. On the one hand, the robot seeks to minimize path length to the goal. On the other hand, it tries to stay away from obstacles by virtue of minimizing the potential function. With the appropriate weight balancing the two objectives, a resulting path may look like the one shown in Figure 25.17(b). This figure also displays the value function derived from the combined cost function, again calculated by value iteration. Clearly, the resulting path is longer, but it is also safer. There exist many other ways to modify the cost function. For example, it may be desirable to smooth the control parameters over time. For example, when driving a car, a smooth path

is better than a jerky one. In general, such higher-order constraints are not easy to accommodate in the planning process, unless we make the most recent steering command   a part of the state. However, it is often easy to smooth the resulting trajectory after planning, using conjugate gradient methods. Such post-planning smoothing is essential in many realworld applications. 25.4.4 Skeletonization methods SKELETONIZATION The second major family of path-planning algorithms is based on the idea of skeletonization. These algorithms reduce the robot's free space to a one-dimensional representation, for which the planning problem is easier. This lower-dimensional representation is called a skeleton of the configuration space.Figure 25.18 shows an example skeletonization: it is a Voronoi graph of the free space—the set of all points that are equidistant to two or more obstacles. To do path planning with a Voronoi graph, the robot first changes its present configuration to a point on the Voronoi graph. It is easy to show that this can always be achieved by a straight-line motion in configuration space. Second, the robot follows the Voronoi graph until it reaches the point nearest to the target configuration. Finally, the robot leaves the Voronoi graph and moves to **the target. Again, this final step involves straight-line motion in configuration space. In this way, the original path-planning problem is reduced to finding a path on the** Voronoi graph, which is generally one-dimensional (except in certain nongeneric cases) and **has finitely many points where three or more one-dimensional curves intersect. Thus, finding** the shortest path along the Voronoi graph is a discrete graph-search problem of the kind discussed in Chapters 3 and 4. Following the Voronoi graph may not give us the shortest path, but the resulting paths tend to maximize clearance. Disadvantages of Voronoi graph techniques are that they are difficult to apply to higher-dimensional configuration spaces, and that they tend to induce unnecessarily large detours when the configuration space is wide open. Furthermore, computing the Voronoi graph can be difficult, especially in configuration space, where the shapes of obstacles can be complex.

An alternative to the Voronoi PROBABILISTIC graphs is the probabilistic roadmap, a skeletonization approach that offers more possible routes, and thus deals better with wide-open spaces. Figure 25.18(b) shows an example of a probabilistic roadmap. The graph is created by randomly generating a large number of configurations, and discarding those that do not fall into free space. Two nodes are joined by an arc if it is "easy" to reach one node from the other–for example, by a straight line in free space. The result of all this is a randomized graph in the robot's free space. If we add the robot's start and goal configurations to this graph, path planning amounts to a discrete graph search. Theoretically, this approach is incomplete, because a bad choice of random points may leave us without any paths from start to goal. It is possible to bound the probability of failure in terms of the number of points generated and certain geometric properties of the configuration space. It is also possible to direct the generation of sample points towards the areas where a partial search suggests that a good path may be found, working bidirectionally from both the start and the goal positions. With these improvements, probabilistic roadmap planning tends to scale better to high—dimensional configuration spaces than most alternative path-planning techniques.

3.Briefly detail the preception?

Perception provides agents with information PERCEPTION about the world they inhabit by interpreting the response of sensors. A sensor measures some aspect of the environment in a form that can be used as input by an agent program. The sensor could be as simple as a switch, which gives one bit telling whether it is on or off, or as complex as the eye. A variety of sensory modalities are available to artificial agents. Those they share with humans include vision, hearing, and touch. Modalities that are not available to the unaided human include radio, infrared, GPS, and wireless signals. Some robots do active sensing, meaning they send out a signal, such as radar or ultrasound, and sense the reflection of this signal off of the environment. Rather than trying to cover all of these, this chapter will cover one modality in depth: vision.

We saw in our description of POMDPs (Section 17.4, page 658) that a model-based decision-theoretic agent in a partially observable environment has a sensor model—a probability distribution P(E | S) over the evidence that its sensors provide, given a state of the world. Bayes' rule can then be used to update the estimation of the state. OBJECT MODEL For vision, the sensor model can be broken into two components: An object model describes the objects that inhabit the visual world—people, buildings, trees, cars, etc. The

object model could include a precise 3D geometric model taken from a computer-aided design (CAD) system, or it could be vague constraints, such as the fact that human eyes are usually 5 RENDERING MODEL to 7 cm apart. A rendering model describes the physical, geometric, and statistical processes that produce the stimulus from the world. Rendering models are quite accurate, but they are ambiguous. For example, a white object under low light may appear as the same color as a black object under intense light. A small nearby object may look the same as a large distant object. Without additional evidence, we cannot tell if the image that fills the frame is a toy Godzilla or a real monster. Ambiguity can be managed with prior knowledge—we know Godzilla is not real, so the image must be a toy—or by selectively choosing to ignore the ambiguity. For example, the vision system for an autonomous car may not be able to interpret objects that are far in the distance, but the agent can choose to ignore the problem, because it is unlikely to crash **into an object that is miles away. A decision-theoretic agent is not the only architecture that can make use of vision sensors.** For example, fruit flies (Drosophila) are in part reflex agents: they have cervical giant fibers that form a direct pathway from their visual system to the wing muscles that initiate an

escape response—an immediate reaction, without deliberation. Flies and many other flying animals make used of a closed-loop control architecture to land on an object. The visual system extracts an estimate of the distance to the object, and the control system adjusts the wing muscles accordingly, allowing very fast changes of direction, with no need for a detailed model of the object. Compared to the data from other sensors (such as the single bit that tells the vacuum robot that it has bumped into a wall), visual observations are extraordinarily rich, both in the detail they can reveal and in the sheer amount of data they produce. A video camera for robotic applications might produce a million 24-bit pixels at 60 Hz; a rate of 10 GB per

minute. The problem for a vision-capable agent then is: Which aspects of the rich visual stimulus should be considered to help the agent make good action choices, and which aspects should be ignored? Vision—and all perception—serves to further the agent's goals, not as an end to itself. We can characterize three FEATURE broad approaches to the problem. The feature extraction EXTRACTION approach, as exhibited by Drosophila, emphasizes simple computations applied directly to RECOGNITION the sensor observations. In the recognition approach an agent draws distinctions among the objects it encounters based on visual and other information. Recognition could mean labelling each image with a yes or no as to whether it contains food that we should forage, or contains RECONSTRUCTION Grandma's face. Finally, in the reconstruction approach an agent builds a geometric model of the world from an image or a set of images.

The last thirty years of research have produced powerful tools and methods for addressing these approaches. Understanding these methods requires an understanding of the processes by which images are formed. Therefore, we now cover the physical and statistical phenomena that occur in the production of an image.

24.1 IMAGE FORMATION

Imaging distorts the appearance of objects. For example, a picture taken looking down a long straight set of railway tracks will suggest that the rails converge and meet. As another example, if you hold your hand in front of your eye, you can block out the moon, which is not smaller than your hand. As you move your hand back and forth or tilt it, your hand will seem to shrink and grow in the image, but
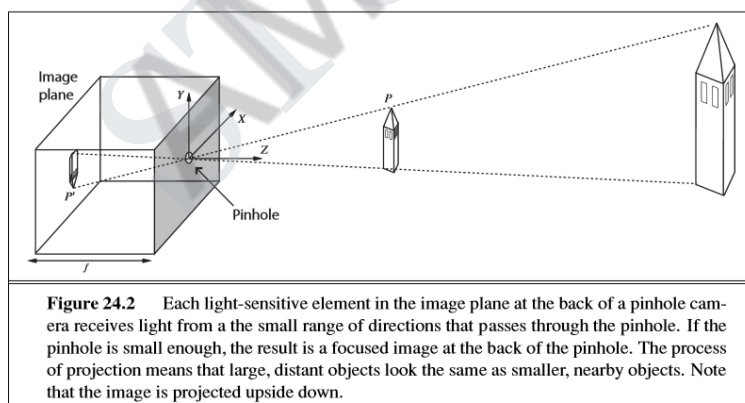
it is not doing so in reality (Figure 24.1). Models of these effects are essential for both recognition and reconstruction.

24.1.1 Images without lenses: The pinhole camera SCENE Image sensors gather light scattered from objects in a scene and create a two-dimensional IMAGE image. In the eye, the image is formed on the retina, which consists of two types of cells: **about 100 million rods, which are sensitive to light at a wide range of wavelengths, and 5** million cones. Cones, which are essential for color vision, are of three main types, each of which is sensitive to a different set of wavelengths. In cameras, the image is formed on an

image plane, which can be a piece of film coated with silver halides or a rectangular grid of a few m illion photosensitive PIXEL pixels, each a complementary metal-oxide semiconductor (CMOS) or charge-coupled device (CCD). Each photon arriving at the sensor produces an effect, whose strength depends on the wavelength of the photon. The output of the sensor is the sum of all effects due to photons observed in some time window, meaning that image sensors report a weighted average of the intensity of light arriving at the sensor. To see a focused image, we must ensure that all the photons from approximately the same spot in the scene arrive at approximately the same point in the image plane. The simplest PINHOLE CAMERA way to form a focused image is to view stationary objects with a pinhole camera, which consists of a pinhole opening, O, at the front of a box, and an image plane at the back of the box (Figure 24.2). Photons from the scene must pass through the pinhole, so if it is small enough then nearby photons in the scene will be nearby in the image plane, and the image will be in focus. The geometry of scene and image is easiest to understand with the pinhole camera. We use a three-dimensional coordinate system with the origin at the pinhole, and consider a point P in the scene, with coordinates (X, Y,Z). P gets projected to the point P in the image plane with coordinates (x, y, z). If f is the distance from the pinhole to the image plane, then by similar triangles, we can derive the following equations:

$$\frac{-x}{f} = \frac{X}{Z}, \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, y = \frac{-fY}{Z} .$$

These equations define an image-formation process known as perspective projection. Note **that the Z in the denominator means that the farther away an object is, the smaller its image** will be. Also, note that the minus signs mean that the image is inverted, both left–right and up–down, compared with the scene.

Figure 24.2   Each light-sensitive element in the image plane at the back of a pinhole cam-era receives light from a small range of directions that passes through the pinhole. If the pinhole is small enough, the result is a focused image at the back of the pinhole. The process of projection means that large, distant objects look the same as smaller, nearby objects. Note that the image is projected upside down.

Under perspective projection, distant objects look small. This is what allows you to cover the moon with your hand (Figure 24.1). An important result of this effect is that parallel lines converge to a point on the horizon. (Think of railway tracks, Figure 24.1.) A line in the scene in the direction (U,
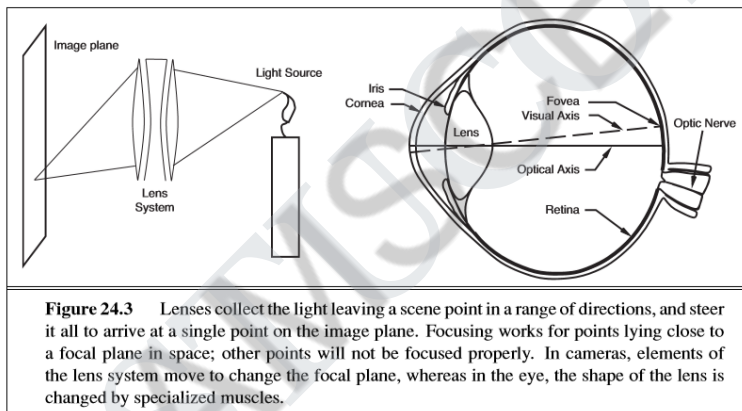
V,W) and passing through the point (X0, Y0, Z0) can be described as the set of points (X0 + λU, Y0 + λV,Z0 + λW), with λ varying between −∞ and +∞. Different choices of (X0, Y0, Z0) yield different lines parallel to one another. The projection of a point Pλ from this line onto the image plane is given by

$$\left(f \frac{X_0 + \lambda U}{Z_0 + \lambda W}, f \frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right) .$$

.

As λ→ ∞or λ → −∞, this becomes p ∞ = (fU/W, fV/W) if W two parallel lines leaving different points in space will converge in the image—for large λ, the image points are nearly the same, whatever the value of (X0, Y0, Z0) (again, think railway tracks, Figure 24.1). We call p ∞ the vanishing VANISHING POINT point associated with the family of straight lines with direction (U, V,W). Lines with the same direction share the same vanishing point.

24.1.2 Lens systems

The drawback of the pinhole camera is that we need a small pinhole to keep the image in focus. But the smaller the pinhole, the fewer photons get through, meaning the image will be dark. We can gather more photons by keeping the pinhole open longer, but then we will get MOTION BLUR motion blur—objects in the scene that move will appear blurred because they send photons to multiple locations on the image plane. If we can't keep the pinhole open longer, we can try to make it bigger. More light will enter, but light from a small patch of object in the scene **will now be spread over a patch on the image plane, causing a blurred image.**

Figure 24.3 Lenses collect the light leaving a scene point in a range of directions, and steer it all to arrive at a single point on the image plane. Focusing works for points lying close to a focal plane in space; other points will not be focused properly. In cameras, elements of the lens system move to change the focal plane, whereas in the eye, the shape of the lens is changed by specialized muscles.

Vertebrate eyes and LENS modern cameras use a lens system to gather sufficient light while keeping the image in focus. A large opening is covered with a lens that focuses light from nearby object locations down to nearby locations in the image plane. However, lens systems DEPTH OF FIELD have a limited depth of field: they can focus light only from points that lie within a range FOCAL PLANE of depths (centered around a focal plane). Objects outside this range will be out of focus in the image. To move the focal plane, the lens in the eye can change shape (Figure 24.3); in a camera, the lenses move back and forth.

24.1.3 Scaled orthographic projection

Perspective effects aren't always pronounced. For example, spots on a distant leopard may look small because the leopard is far away, but two spots that are next to each other will have about the same size. This is because the difference in distance to the spots is small compared to the distance to them, and so we can simplify the projection model. The appropriate model is scaled orthographic projection. The idea is as follows: If the depth Z of points on the SCALED ORTHOGRAPHIC

PROJECTION object varies within some range Z0 Â± ΔZ, with ΔZ * Z0, then the perspective scaling factor f/Z can be approximated by a constant s = f/Z0. The equations for projection from the scene coordinates (X, Y,Z) to the image plane become x = sX and y = sY . Scaled orthographic projection is an approximation that is valid only for those parts of the scene with not much internal depth variation. For example, scaled orthographic projection can be a good model for the features on the front of a distant building.

24.1.4 Light and shading

The brightness of a pixel in the image is a function of the brightness of the surface patch in the scene that projects to the pixel.
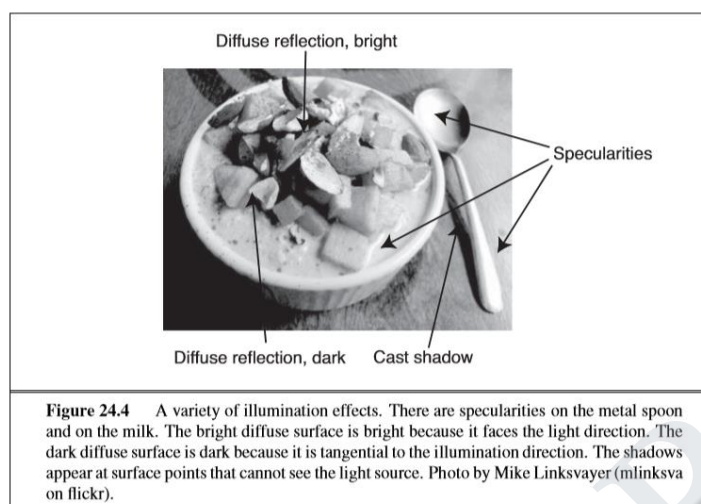

Figure 24.4   A variety of illumination effects. There are specularities on the metal spoon and on the milk. The bright diffuse surface is bright because it faces the light direction. The dark diffuse surface is dark because it is tangential to the illumination direction. The shadows appear at surface points that cannot see the light source. Photo by Mike Linksvayer (mlinksva on flickr).
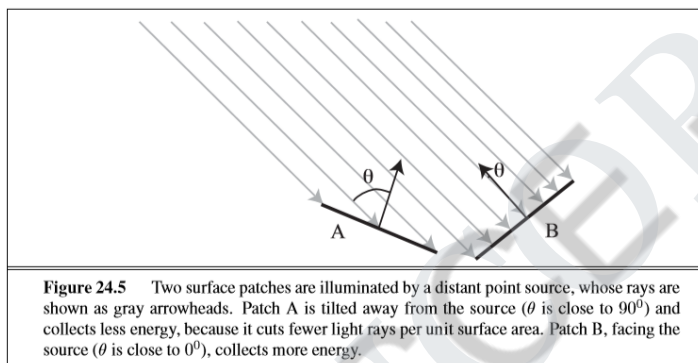
 We will assume a linear model (current cameras have nonlinearities **at the extremes of light and dark, but are linear in the middle). Image brightness is** a strong, if ambiguous, cue to the shape of an object, and from there to its identity. People are usually able to distinguish the three main causes of varying brightness and reverse-engineer the object's properties. The first cause OVERALL INTENSITY is overall intensity of the light. Even though a white object in shadow may be less bright than a black object in direct sunlight, the eye can distinguish relative brightness well, and perceive the white object as white. Second, different points REFLECT in the scene may reflect more or less of the light. Usually, the result is that people perceive these points as lighter or darker, and so see texture or markings on the object. Third, surface patches facing the light are brighter than surface patches tilted away from the light, an effect SHADING known as shading. Typically, people can tell that this shading comes from the geometry of the object, but sometimes get shading and markings mixed up. For example, a streak of dark makeup under a cheekbone will often look like a shading effect, making the face look thinner. DIFFUSE Most surfaces reflect light by a process of diffuse reflection. Diffuse reflection scat- REFLECTION ters light evenly across the directions leaving a surface, so the brightness of a diffuse surface doesn't depend on the viewing direction. Most cloth, paints, rough wooden surfaces, vegetation, and rough stone are diffuse. Mirrors are not diffuse, because what you see depends on the direction in which you look at the mirror. The behavior of a perfect mirror is known as

SPECULAR specular reflection. Some surfaces—such as brushed metal, plastic, or a wet floor— display EFLECTION SPECULARITIES small patches where specular reflection has occurred, called specularities. These are easy to identify, because they are small and bright (Figure 24.4). For almost all purposes, it is enough **to model all surfaces as being diffuse with specularities. The main source of illumination outside is the sun, whose rays all travel parallel to one** another. We model this behavior as a distant point light DISTANT POINT source. This is the most important LIGHT SOURCE model of lighting, and is quite effective for indoor scenes as well as outdoor scenes. The

amount of light collected by a surface patch in this model depends on the angle θ between the illumination direction and the normal to the surface. A diffuse surface patch illuminated by a distant point light source will reflect some DIFFUSE ALBEDO fraction of the light it collects; this fraction is called the diffuse albedo. White paper and snow have a high albedo, about 0.90, whereas flat black velvet and charcoal have a low albedo of about 0.05 (which means that 95% of the incoming light is absorbed within the fibers of LAMBERT'S COSINE the velvet or the pores of the charcoal). Lambert's cosine law states that the brightness of a LAW diffuse patch is given by

$I = \rho I0 \cos \theta$,

where ρ is the diffuse albedo, I0 is the intensity of the light source and θ is the angle between the light source direction and the surface normal (see Figure 24.5). Lampert's law predicts bright image pixels come from surface patches that face the light directly and dark pixels come from patches that see the light only tangentially, so that the shading on a surface provides some shape information. We explore this cue in Section 24.4.5. If the surface is not SHADOW reached by the light source, then it is in shadow. Shadows are very seldom a uniform black, because the shadowed surface receives some light from other sources. Outdoors, the most important such source is the sky, which is quite bright. Indoors, light reflected from other INTERREFLECTIONS surfaces illuminates shadowed patches. These interreflections can have a significant effect on the brightness of other surfaces, too. These effects are sometimes modeled by adding a AMBIENT constant ambient illumination term to the predicted intensity.

Figure 24.5 Two surface patches are illuminated by a distant point source, whose rays are shown as gray arrowheads. Patch A is tilted away from the source (θ is close to 90^0) and collects less energy, because it cuts fewer light rays per unit surface area. Patch B, facing the source (θ is close to 0^0), collects more energy.

ILLUMINATION
24.1.5 Color

Fruit is a bribe that a tree offers to animals to carry its seeds around. Trees have evolved to have fruit that turns red or yellow when ripe, and animals have evolved to detect these color changes. Light arriving at the eye has different amounts of energy at different wavelengths; this can be represented by a spectral energy density function. Human eyes respond to light in the 380–750nm wavelength region, with three different types of color receptor cells, which have peak receptiveness at 420mm (blue), 540nm (green), and 570nm (red). The human eye can capture only a small fraction of the full spectral energy density function—but it is enoughto tell when the fruit is ripe The principle of trichromacy states that for PRINCIPLE OF any spectral energy density, no matter how TRICHROMACY complicated, it is possible to construct another spectral energy density consisting of a mixture of just three colors—usually red, green, and blue—such that a human can't tell the difference between the two. That means that our TVs and computer displays can get by with just the three red/green/blue (or R/G/B) color elements. It makes our computer vision algorithms easier, too. Each surface can be modeled with three different albedos for R/G/B. Similarly each light source can be modeled with three R/G/B intensities. We then apply Lambert's cosine law to each to get three R/G/B pixel values. This model predicts, correctly, that the same surface will produce different colored image patches under different-colored lights. In fact, human observers are quite good at

ignoring the effects of different colored lights and are COLOR CONSTANCY able to estimate the color of the surface under white light, an effect known as color constancy. Quite accurate color constancy algorithms are now available; simple versions show up in the "auto white balance" function of your camera. Note that if we wanted to build a camera for mantis shrimp, we would need 12 different pixel colors, corresponding to the 12 types ofcolor receptors of the crustacean.

4.Briefly explain the Natural Language Processing.

There are over a trillion pages of information on the Web, almost all of it in natural language, An agent that wants to do knowledge acquisition needs to understand (at least partially) the ambiguous, messy languages that humans use. We examine the problem from the point of view of specific information-seeking tasks: text classification, information re- trieval, and information extraction. One common factor in addressing these tasks is the use of language models: models that predict the probability distribution of language expressions.

LANGUAGE MODELS:

Formal languages, such as the programming languages Java or Python, have precisely defined language models. A language can be defined as a set of strings; "print (2 + 2 ► " is a legal program in the language Python, whereas "2)+(2 print" is not. Since there are an infinite number of legal programs, they cannot be enumerated; instead they are specified by a set of rules called a grammar. Formal languages also have rules that define the meaning or semantics of a program; for example, the Riles say that the "meaning" of "2 + 2" is 4, and the meaning of "1 / Cl" is that an error is signaled. Natural languages, such as English or Spanish, cannot be characterized as a definitive act of sentences. Everyone agrees that "Not to be invited is sad" is a sentence of English, but people disagree on the grammaticality of "To be not invited is sad." Therefore, it is more fruitful to define a natural language model as a probability distribution over sentences rather than a definitive set. That is, rather than asking if a string of words is or is not a member of the set defining the language, we instead ask for $P(S = words)$—what is the probability that a random sentence would be words. Natural languages are also ambiguous. "He saw her duck" can mean either that he saw a waterfowl belonging to her, or that he saw her move to evade something. Thus, again, we cannot speak of a single meaning for a sentence, but rather of a probability distribution over possible meanings. Finally, natural languages are difficult to deal with because they are very large, and constantly changing. Thus, our language models are, at best, an approximation. We start with the simplest possible approximations and move up from there N gram character models ultimately, a written text is composed of characters—letters, digits, punctuation, and spaces in English (and more exotic characters in some other languages). Thus, one of the simplest language models is a probability distribution over sequences of characters. As in Chapter 15, we write $P(c_i, N)$ for the probability of a sequence of N characters, e_i through c_{isr}. In oneWeb collection, $P("the") = 0.027$ and $P("zgq") = 0.00UU00002$. A sequence of written sym-bols of length n is called an n-gram (from the Greek root for writing or letters), with special case "unigram" for l-gram, "bigram" for 2-gram, and "trigram" for 3-gram. A model of theprobability distribution of n-letter sequences is thus called an n-gram model. (But be care-ful: we can have n-gram models over sequences of words, syllables, or other units; not justover characters.)An n,-gram model is defined as a Markov chain of order n — 1. Recall from page 568that in a Markov chain the probability of character c_i depends only on the immediately pre-ceding characters, not on any other characters. So in a trigram model (Markov chain oforder 2) we have$P(c_i|c_{i-1}) = P(e_i | e_{i-2:i-1})$ •We can define the probability of a sequence of characters $P(c_i, /y)$ under the trigram modelby first factoring with the chain rule and then using the Markov assumption:$P(C_1:N) =\prod_{i=1} P(c_i | N = \prod_{i=1} Ph | e_{i-2:i-1})$ Pr, C, _2.,_ 1) has a millionentries, and can be accurately estimated by counting character sequences in a body of text of10 million characters or more. We call a body of text a corpus {plural corposu), from theLatin word for body.What can we do with n-gram character models? One task for which they are well suitedis language identification: given a text, determine

what natural language it is written in. Thisis a relatively easy task; even with short texts such as "Hello, world" or "Wie geht es dir," itis easy to identify the first as English and the second as German. Computer systems identify languages with greater than 99% accuracy; occasionally, closely related languages, such as Swedish and Norwegian, are confused.One approach to language identification is to first build a trigram character model ofeach candidate language, $P(c_i \mid 1)$, where the variable .E ranges over languages. Fme ach f the model is built by counting trigrams in a corpus of that language. (About 100,000characters of each language are needed.) That gives us a model of P, Text Language.), but we want to select the most probable language given the text, so we apply Bayes' rule followed by the Markov assumption to get the most probable language:$r = \text{argrnax } P(i \mid c_i, N)$

= argmax

$P(/) P(c_i:N = \text{argmax } P(1) \text{ I t)ft } P(c_i C_{i-2} N-1, =1$

The trigram model can be learned from a corpus, but what about the prior probability $P(P)$?We may have some estimate of these values; for example, if we are selecting a random Webpage we know that English is the most likely language and that the probability of Macedonianwill be less than 1%. The exact number we select for these priors is not critical because thetrigram model usually selects one language that is several orders of magnitude more probablethan any other.Other tasks for character models include spelling correction, genre classification, andnamed-entity recognition. Genre classification means deciding if a text is a news story, alegal document, a scientific article, etc. While many features help make this classification,counts of punctuation and other character n-gram features go a long way (Kessler et aL,1997). Named-entity recognition is the task of finding names of things in a document anddeciding what class they belong to. For example, in the text "Mr. Sopersteen was prescribedaciphex," we should recognize that "Mr. Sopersteen" is the name of a person and "aciphex" isthe name of a drug. Character-level models are good for this task because they can associatethe character sequence "ex_" ("ex" followed by a space) with a drug name and "steen_" witha person name, and thereby identify words that they have never seen before.

22.1.2 Smoothing n gram models

-

The major complication of n-gram models is that the training corpus provides only an esti-mate of the true probability distribution. For common character sequences such as "-th" anyEnglish corpus will give a good estimate: about 1.5% of all trigrams. On the other hand, ".ht"is very uncommon—no dictionary words start with ht. It is likely that the sequence wouldhave a count of zero in a training corpus of standard English. Does that mean we should as-sign $P(\text{"-th"}) = 0$? If we did, then the text "The program issues an http request" would have an English probability of zero, which seems wrong. We have a problem in generalization: wewant our language models to generalize well to texts thcy haven't seen yet. Just because wehave never seen "imp" before does not mean that our model should claim that it is impossi-ble. Thus, we will adjust our language model so that sequences that have a count of zero inthe training corpus will be assigned a small nonzero probability (and the other counts will beadjusted downward slightly so that the probability still sums to O. The process od adjustingthe probability of low-frequency counts is called smoothing.The simplest type of smoothing was suggested by Pierre-Simon Laplace in the 18th cen-tury: he said that, in the lack of further information, if a random Boolean variable X has beenfalse in all n observations so far then the estimate for $P(X = \text{true})$ should be $1 \ gn \pm 2)$. Thatis, he assumes that with two more trials, one might be true and one false. Laplace smoothing(also called add-one smoothing) is a step in the right direction, but performs relatively poorly.A better approach is a backoff model, in which we start by estimating n-gram counts, but furany particular sequence that has a low (or zero) count, we back off to $(n - 1)$-grams. Linearinterpolation smoothing is a backotf model that combines trgram, bigram, and unigrammodels by linear interpolation. It defines the probability estimate as$P(e_i = A.3 P(c_i \text{ I } e_{t-2}:z - + A_2 P(e_i e-1) + A_i P(C_2))12$where $A_3 + + = 1$. The parameter values A_i can be fixed, or they can be

trained withan expectation–maximization algorithm. It is also possible to have the values of Ai dependon the counts: if we have a high count of trigrams, then we weigh them relatively more; ifonly a low count, then we put more weight on the bigram and !migrant models. One camp ofresearchers has developed ever more sophisticated smoothing models, while the other campsuggests gathering a larger corpus so that even simple smoothing models work well. Both aregetting at the same goal: reducing the variance in the language model.One complication: note that the expression P(ciasks for P(ci c-1 : 0) whena. = 1, but there are no characters before Cl. We can introduce artificial characters, forexample, defining ro to be a space character or a special "begin text" character. Or we canfall back on lower-order Markov models, in effect defining C _1, O to be the empty sequenceand thus P(c i c_ iro) = P(c i).Model evaluation:With so many possible n-gram models—unigram, bigram, trigrann, interpolated smoothingwith different values of A, etc.—how do we know what model to choose? We can evaluate amodel with cross-validation. Split the corpus into a training corpus and a validation corpus.Determine the parameters of the model from the training data. Then evaluate the model onthe validation corpus.The evaluation can be a task-specific metric, such as measuring accuracy on languageidentification. Alternatively we can have a task-independent model of language quality: cal-culate the probability assigned to the validation corpus by the model; the higher the proba-bility the better. This metric is inconvenient because the probability of a large corpus willbe a very small number, and floating-point andel - Row becomes an issue. A different way ofdescribing the probability of a sequence is with a measure called perplexity, defined as   Perplexity(ei,N)= P(ci : N)Perplexity can be thought of as the reciprocal of probability, normalized by sequence length.It can also be thought of as the weighted average branching factor of a model. Suppose thereare 100 characters in our language, and our model says they are all equally likely. Then fora sequence of any length, the perplexity will be 100. If some characters are more likely thanothers, and the model reflects that, then the model will have a perplexity less than 100. N gram word models: Now   we turn to n-gram models over words rather than characters. All the same mechanismapplies equally to word and character models. The main difference is that the vocabulary—the set of symbols that make up the corpus and the model—is larger. There are only about100 characters in most languages, and sometimes we build character models that are evenmore restrictive, for example by treating "A" and "a" as the same symbol or by treating allpunctuation as the same symbol. But with word models we have at least tens of thousands ofsymbols, and sometimes millions. The wide range is because it is not clear what constitutes aword. In English a sequence of letters surrounded by spaces is a word, but in some languages,like Chinese, words are not separated by spaces, and even in English many decisions must bemade to have a clear policy on word boundaries: how many words are in "ne'er-do-well"? Or in "(Te1:1-800-960-5660x123)"?Word n-gram models need to deal with out of vocabulary words. With character mod-els, we didn't have to worry about someone inventing a new letter of the alphabet. 1 Butwith word models there is always the chance of a new word that was not seen in the trainingcorpus, so we need to model that explicitly in our language model. This can be done byadding just one new word to the vocabulary: <UNK>, standing for the unknown word. Wecan estimate n-gram counts for <UNK> by this trick: go through the training corpus, andthe first lime any individual word appears it is previously unknown, so replace it with thesymbol <UNK>. All subsequent appearances of the word remain unchanged. Then computen-gram counts for the corpus as usual, treating <UNK> just like any other word. Then whenan unknown word appears in a test set, we look up its probability under <UNK>. Sometimesmultiple unknown-word symbols are used, for different classes. For example, any string ofdigits might be replaced with <NUM>, or any email address with <EMAIL>.To get a feeling for what word models can do, we built unigram, bigram, and trigrarnmodels over the words in this book and then randomly sampled sequences of words from themodels. The results areUnigram: logical are as are contusion a may right tries agent goal the was . .Bigram: systems are very similar computational approach would be represented ..Trigram: planning

and scheduling are integrated the success of naive bayes model is ...Even with this small sample, it should be clear that the unigram model is a poor approximationof either English or the content of an AI textbook, and that the bigram and trigrarn models much better. The models agree with this assessment: the perplexity was 891 for the unigrammodel, 142 for the bigram model and 91 for thc trigram model.With the basics of n-gram models—both character- and word-based—established, wecan turn now to some language tasks. TEXT CLASSIFICATION: We now consider in depth the task of text classification, also known as categorization: givena text of some kind, decide which of a predefined set of classes it belongs to. Language iden-tification and genre classification are examples of text classification, as is sentiment analysis(classifying a movie or product review as positive or negative) and spam detection (classify-ing an email message as spam or not-spam). Since "not-spam" is awkward, researchers havecoined the term ham for not-spam. We can treat spam detection as a problem in supervisedlearning. A training set is readily available: the positive (spam) examples are in my spunfolder, the negative (ham) examples are in my inbox_ Here is an excerpt:Spam: Wholesale Fashion Watches -57% today. Designer watches for cheap ...

Spam: You can buy ViagraFt$1.85 All Medications at unbeatable prices! ...

Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...

Spam: Start eam*ing the salary yo,u d-eserve by obtaining the prope,r erede'ntials!

Ham: The practical significance of hypertree width in identifying more

Ham: Abstract: We will motivate the problem of social identity clustering: ...

Ham: Good to see you my friend. Hey Peter, It was good to hear from you....

Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

From this excerpt we can start to get an idea of what might be good features to include inthe supervised learning model. Word n-grams such as "for cheap" and "You can buy" seemto be indicators of spam (although they would have a nonzero probability in ham as well).Character-level features also seem important: spam is more likely to be all uppercase and tohave punctuation embedded in words. Apparently the spammers thought that the word bigram"you deserve" would be too indicative of spam, and thus wrote "yo,u d-eserve" instead. Acharacter model should detect this. We could either create a full character n-gram modelof spam and ham, or we could handcraft features such as "number of punctuation marksembedded in words "Note that we have two complementary ways of talking about classification. In thelanguage-modeling approach, we define one n-gram language model for P(Message spam)by training on the spam folder, and one model for P(Message hare) by training on the inbox.Then we can classify a new message with an application of Hayes' rule:arginaxcE(spam.harrilP(c I message) =argrnaxP(message c) P(c) .cEisparn,ham}where. P(c) is estimated just by counting the total number of Spam and ham messages. Thisapproach works well for spam detection, just as it did for language identification. In the machine-learning approach we represent the message as a set of feature/valuepairs and apply a classification algorithm h to thc feature vector X. We can make thclanguage-modeling and machine-learning approaches compatible by thinking of the n-gramsas features. This is easiest to see with a unigram model. The features are the words in thevocabulary: "a," "aardvark," ..., and the values are the number of times each word appearsin the message. That makes the feature vector large and sparse. If there are 100,000 words inthe language model, then the feature vector has length 100,000, but for a short email messagealmost all the features will have count zero. This unigram representation has been called thebag of words model. You can think of the model as putting the words of the training corpusin a bag and then selecting words one at a time. The notion of order of the words is lost; aunigram model gives the same probability to any permutation of a text. Higher-order n-grammodels maintain some local notion of word order.With bigrams and !lignin's the number of features is squared or cubed, and we can addin other, non-'n-gram features: the time the message was sent, whether a URL or an imageis part of the message, an ID number for the sender of the message, the sender's number ofprevious spam and ham

messages, and so on. The choice of features is the most important partof creating a good scam detector—more important than the choice of algorithm for processingthe features. In part this is because there is a lot of training data, so if we can propose afeature, the data can accurately determine if it is good or not. It is necessary to constantlyupdate features, because spam detection is an adversarial task; the spammers modify theirspam in response to the span] detector's changes.It can be expensive to run algorithms on a very large feature vector, so often a processof feature selection is used to keep only the features that best discriminate between spam andham. For example, the bigram "of the" is frequent in English, and may be equally frequent inspam and ham, so there is no sense in counting it. Often the top hundred or so features do agood job of discriminating between classes.Once we have chosen a set of features, we can apply any of the supervised learningtechniques we have seen; popular ones for text categorization include k-nearest-neighbors,support vector machines, decision trees, naive Hayes, and logistic regression_ All of thesehave been applied to spam detection, usually with accuracy in the 98%-99% range. With acarefully designed feature set, accuracy can exceed 99.9%. Classification by data compression: Another way to think about classification is as a problem in data compression. A losslesscompression algorithm takes a sequence of symbols, detects repeated patterns in it, and writesa description of the sequence that is more compact than the original. For example, the text"0.142857142857142857" might be compressed to "0.[142857]*3." Compression algorithmswork by building dictionaries of subsequences of the text, and then referring to entries in thedictionary. The example here had only one dictionary entry, "142857."In effect, compression algorithms are creating a language model. The LZW algorithmin particular directly models a maximum-entropy probability distribution. To do classificationby compression, we first lump together all the spam training messages and compress them asa unit. We do the same for the ham. Then when given a new message to classify, we appendit to the spam messages and compress the result. We also append it to the ham and compressthat. Whichever class compresses better—adds the fewer number of additional bytes for thenew message—is the predicted class. The idea is that a spam message will tend to sharedictionary entries with other spam messages and thus will compress better when appended toa collection that already contains the spam dictionary.Experiments with compression-based classification on some of the standard corpora fortext classification—the 20-Newsgroups data set, the Reuters-10 Corpora, the Industry Sectorcorpora—indicate that whereas running off-the-shelf compression algorithms like gzip, RAR,and LZW can be quite slow, their accuracy is comparable to traditional classification algo-rithms. This is interesting in its own right, and also serves to point out that there is promisefor algorithms that use character n-grams directly with no preprocessing of the text or featureselection: they seem to be captiring some real patterns. INFORMATION RETRIEVAL:Information retrieval is the task of finding documents that are relevant to a user's need forinformation. The best-known examples of information retrieval systems are search engineson the World Wide Web. A Web user can type a query such as [AI book] Z into a search engineand see a list of relevant pages. In this section, we will see how such systems are built. Aninformation retrieval (henceforth IR) system can be characterized byA corpus of documents. Each systcm must dccidc what it wants to treat as a document:a paragraph, a page, or a multipage text.2. Queries posed in a query language. A query specifies what the user wants to know.The query language can be just a list of words, such as [AI book]; or it can specifya phrase of words that must be adjacent, as in ("Al book"]; it can contain Booleanoperators as in [AI AND book]; it can include non-Boolean operators such as [AI NEARbook] or [AI book site:www.aaai.org].3. A result set. This is the subset of documents that the IR system judges to be relevant tothe query. By relevant, we mean likely to be of use to the person who posed the query,for the particular information need expressed in the query.4. A presentation of the result set. This can be as simple as a ranked list of documenttitles or as complex as a rotating color map of the result set projected onto a three-dimensional space, rendered as a two-dimensional display.The earliest IR systems worked on a

Boolean keyword model. Each word in the documentcollection is treated as a Boolean feature that is true of a document if the word occurs in thedocument and false if it does not. So the feature "retrieval" is true for the current chapterbut false for Chapter 15. The query language is the language of Boolean expressions overfeatures. A document is relevant only if the expression evaluates to true. For example, thequay [information AND retrieval] is true for the current chapter and false for Chapter 15.This model has the advantage of being simple to explain and implement. However.it has some disadvantages. First, the degree of relevance of a document is a single bit, sothere is no guidance as to how to order the relevant documents for presentation. Second,Boolean expressions are unfamiliar to users who are not programmers or logicians. Usersfind it unintuitive that when they want to know about farming in the states of Kansas andNebraska they need to issue the query [farming (Kansas OR Nebraska)]. Third, it can behard to formulate an appropriate query, even for a skilled user. Suppose we try [informationAND retrieval AND models AND optimization] and get an empty result set. We could try[information OR retrieval OR models OR optimization], but if that returns too many results,it is difficult to know what to try next.IR scoring functionsMost IR systems have abandoned the Boolean model and use models based on the statistics ofword counts. We describe the BM25 scoring function, which comes from the Okapi projectof Stephen Robertson andKaren Sparck Jonesat London's City College, and has been usedin search engines such as the open-source Lucene project.A scoring function takes a document and a query and returns a numeric score; the mostrelevant documents have the highest scores. In the BM25 function, the score is a linearweighted combination of scores for each of the words that make up the query. Three Factorsaffect the weight of a query term: First, the frequency with which a query term appears ina document (also known as IF for term frequency). For the query [farming in Kansas],documents that mention "farming" frequently will have higher scores. Second, the inverseIDF. The word "in" appears in almost every document,so it has a high document frequency, and thus a low inverse document frequency, and thus itis not as important to the query as "farming" or "Kansas." Third, the length of the document.A million-word document will probably mention all the query words, but may not actually beabout the query. A short document that mentions all the words is a much better candidate.The BM25 function takes all three of these into account. We assume we have createdan index of the N documents in the corpus so that we can look up TF(ai. di), the count ofthe number of times word q, appears in document di. We also assume a table of documentfrequency counts, DP (q,), that gives the number of documents that contain the word rliThen, given a document dj and a query consisting of the wordswe havedocument frequency of the term, orBlvI25(dj: qi,N) =L/DF(qi)i=1TF(qi, di) k— b b)where 1(1 3 1 is the length of document d i in words, and L is the average document lengthin the corpus: L =I d i I N. We have two parameters, Jr and b, that can be tuned bycross-validation; typical values are k = 2.0 and b = 0.75. IDF (qi) is the inverse documentfrequency of word qi, given byIDF(qi) = logN — DE(%)0.5DF(qi) ± 0.5Of course, it would be impractical to apply the BM25 scoring function to every documentin the corpus. Instead, systems create an index ahead of time that lists, for each vocabularyword, the documents that contain the word. This is called the hit list for the word. Then whengiven a query, we intersect the hit lists of the query words and only score the documents inthe intersection. IR system evaluationHow do we know whether an IR system is performing well? We undertake an experiment inwhich the system is given a set of queries and the result sets are scored with respect to humanrelevance judgments. Traditionally, there have been two measures used in the scoring: recalland precision. We explain them with the help of an example. Imagine that an IR system hasreturned a result set for a single query, for which we know which documents are and are notrelevant, out of a corpus of 100 documents. The document counts in each category are givenin the following table:RelevantNot relevantPR=rIRICINRECALLIn result set Not in result set30201040Precision measures the proportion of documents in the result set that are actually relevantIn our example, the precision is 30/(30 + 10) = .75. The false positive rate is 1 — .75 =

.25.Retail measures the proportion of all the relevant documents in the collection that arc inthe result set. In our example, recall is $30/(30 + 20) = .60$. The false negative rate is $1 — .60 = .40$. In a very large document collection, such as the World Wide Web, recall is difficultto compute, because there is no easy way to examine every page on the Web for relevance.All we can do is either estimate recall by sampling or ignore recall completely and just judgeprecision. In the case of a Web search engine, there may be thousands of documents in theresult set, so it makes more sense to measure precision for several different sizes, such as"P@10" (precision in the top 10 results) or "P@50," rather than to estimate precision in theentire result set.It is possible to trade off precision against recall by varying the size of the result setreturned. In the extreme, a system that returns every document in the document collection isguaranteed a recall of 100%, but will have low precision. Alternately, a system could returna single document and have low recall, but a decent chance at 100% precision. A summaryof both measures is the Ft score, a single number that is the harmonic mean of precision andrecall, 2P RI (P R).

22.3.3 IR refinements

There are many possible refinements to the system described here, and indeed Web search
engines arc continually updating their algorithms as they discover new approaches and as the
Web grows and changes.One common refinement is a better model of the effect of document length on relevance.Singhal et al. (1996) observed that simple document length normalization schemes tend tofavor short documents too much and long documents not enough. They propose a pivoteddocument length normalization scheme; the idea is that the pivot is the document length atwhich the old-style normalization is correct; documents shorter than that get a boast andlonger ones get a penalty.The BM25 scoring function uses a word model that treats all words as completely in-dependent, but we know that some words are correlated: "couch" is closely related to both"couches" and "sofa." Many IR systems attempt to account for these correlations.For example, if the query is [couch], it would he a shame to exclude from the result setthose documents that mention "COUCH" or "couches" but not "couch." Most IR systemsdo case folding of "COUCH" to "couch." and some use a stemming algorithm to reduce"couches" to the stern form "couch," both in the query and the documents. This typicallyyields a small increase in recall (on the order of 2% for English). However, it can harmprecision. For example, stemming "stocking" to "stock" will tend to decrease precision forqueries about either foot coverings or financial instruments, although it could improve recallfor queries about warehousing. Stemming algorithms based on rules (e.g., remove "-ing")cannot avoid this problem, but algorithms based on dictionaries (don't remove "-ing" if theword is already listed in the dictionary) can. While stemming has a small effect in English,it is more important in other languages. In German, for example, it is not uncommon tosee words like "Lebensversicherungsgesellschaftsangestellter" (life insurance company em-ployee). Languages such as Finnish, Turkish, Inuit, and Yupik have recursive morphologicalrules that in principle generate words of unbounded length.The next step is to recognize synonyms, such as "sofa" for "couch." As with stemming_this has the potential for small gains in recall, but can hurt precision. A user who gives thequery [Tim Couch] wants to see results about the football player, not sofas. The problem isthat "languages abhor absolute synonyms just as nature abhors a vacuum" (Cruse, 1986). Thatis, anytime there are two words that mean the same thing, speakers of the language conspireto evolve the meanings to remove the confusion_ Related words that are not synonyms alsoplay an important role in ranking—terms like "leather", "wooden," or "modem" can serveto confirm that the document really is about "couch." Synonyms and related words can befound in dictionaries or by looking for correlations in documents or in queries—if we findthat many users who ask the query [new sofa] follow it up with the query [new couch], wecan in the future alter [new sofa] to be [new sofa OR new couch].As a final refinement, IR can be improved by considering metadata—data outside ofthe text of the document. Examples include human-supplied keywords and publication data.On the Web,

hypertext links between documents are a crucial source of information.The PageRank algorithmPageRank 3 was one of the two original ideas that set Google's search apart from other Websearch engines when it was introduced in 1997. (The other innovation was the use of anchorfunction HITS(query) returns pages with hub and authority numberspages 4- EXPAND-PAGES(RELEVANT-PAGESS query))for each p in pages dop.AUTHORITYp.HUB 0— 11repeat until convergence dofor each p in pages dop.AurnonITY 4—E, INLINK,(p).HUBp.HUB 4- 1 . :,, OUTLINK.(p).ALITUORITYNORMALIZE(pages)return pagesFigure 221 The HITS algorithm for computing hubs and authorities with respect to aquery. RFLEVANT-PAGES fetches the pages that match the query, and F. XPA NM- PA GF S addsin every page that links to or is linked from one of the relevant pages. NORMALIZE divideseach page's score by the sum of the squares of all pages' scores (separately for both theauthority and hubs scores).text—the underlined text in a hyperlink—to index a page, even though the anchor text was ona dfferetn page than the one being indexed.) PageRank was invented to solve the problem ofthe tyranny of TF scores: if the query is [IBM], how do we make sure that IBM's home page,ibm. corn, is the first result, even if another page mentions the term "IBM" more frequently?The idea is that ibm. comhas many in-links (links to the page), so it should he ranked higher:each in-link is a vote for the quality of the linked-to page. But if we only counted in-links,then it would be possible for a Web spanimer to create a network of pages and have them allpoint to a page of his choosing, increasing the score of that page. Therefore, the PageRankalgorithm is designed to weight links from high-quality sites more heavily. What is a high-quality site? One that is linked to by other high-quality sites. The definition is recursive, butwe will see that the recursion bottoms out properly. The PageRank for a page p is defined as:PR(p) =1 — dN+ d E c(ini)where PR(p) is the PageRank of page p, N is the total number of pages in the corpus, inare the pages that link in to p ; and C(in t) is the count of the total number of out-links onpage in,. The constant d is a damping factor. It can be understood through the randomsurfer model: imagine a Web surfer who starts at some random page and begins exploring.With probability d (we'll assume d= 0.85) the surfer clicks on one of the links on the page(choosing uniformly among them), and with probability 1 — d she gets bored with the pageand restarts on a random page anywhere on the Web. The PageRank of page p is then theprobability that the random surfer will be at page p at any point in time. PageRank can becomputed by an iterative procedure: start with all pages having EMI)) = 1, and iterate thealgorithm, updating ranks until they converge.The Hyperlink-Induced Topic Search algorithm, also known as "Hubs and Authorities" orHITS, is another influential link-analysis algorithm (see Figure 22.1). HITS differs fromPageRank in several ways. First, it is a query-dependent measure: it rates pages with respectto a query. That means that it must be computed anew for each query—a computationalburden that most search engines have elected not to take on. Given a query, HITS first findsa set of pages that are relevant to the query. It does that by intersecting hit lists of querywords, and then adding pages in the link neighborhood of these pages—pages that link to orare linked from one of the pages in the original relevant set.Each page in this set is considered an authority on the query to the degree that otherpages in the relevant set point to it. A page is considered a hub to the degree that it pointsto other authoritative pages in the relevant set. Just as with PageRank, we don't want tomerely count the number of links; we want to give more value to the high-quality hubs andauthorities. Thus, as with PageRank, we iterate a process that updates the authority score ofa page to be the sum of the hub scores of the pages that point to it, and the hub score to bethe sum of the authority scores of the pages it points to. If we then normalize the scores andrepeat k times, the process will converge.Both PageRank and HITS played important roles in developing our understanding ofWeb information retrievaL These algorithms and their extensions are used in ranking billionsof queries daily as search engines steadily develop better ways of extracting yet finer signalsof search relevance.

22.3.6 Question answering

Information retrieval is the task of finding documents that are relevant to a query, where thequery may be a question, or just a topic area or concept. Question answering is a somewhatdifferent task, in which the query really is a question, and the answer is not a ranked listof documents but rather a short response—a sentence, or even just a phrase. There havebeen question-answering NLP (natural language processing) systems since the 1960s, butonly since 2001 have such systems used Web information retrieval to radically increase theirbreadth of coverage.The AsKMSR system (Banko el al., 2002) is a typical Web-based question-answeringsystem. It is based on the intuition that most questions will be answered many limes on theWeb, so question answering should be thought of as a problem in precision, not recall. Wedon't have to deal with all the different ways that an answer might be phrased—we onlyhave to find one of them. For example, consider the query [Who killed Abraham Lincoln?]Suppose a system had to answer that question with access only to a single encyclopedia,whose entry on Lincoln saidJohn Wilkes Booth altered history with a bullet. He will forever be known as the manwho ended Abraham Lincoln's life.To use this passage to answer the question, the system would have to know that ending a lifecan be a killing, that "He" refers to Booth, and several other linguistic and semantic facts.ASKMSR does not attempt this kind of sophistication—it knows nothing about pronounreference, or about killing, or any other verb. It does know 15 different kinds of questions, andhow they can be rewritten as queries to a search engine. It knows that [Who killed AbrahamLincoln] can be rewritten as the query [4.- killed Abraham Lincoln] and as [Abraham Lincolnwas killed by 1. It issues these rewritten queries and examines the results that come back—not the full Web pages, just the short summaries of text that appear near the query terms.The results are broken into 1-, 2-, and 3-grams and tallied for frequency in the result sets andfor weight: an is-gram that came back from a very specific query rewrite (such as the exactphrase match query [`Abraham Lincoln was killed by *"]l would get more weight than onefrom a general query rewrite, such as [Abraham OR Lincoln OR killed]. We would expectthat "John Wilkes Booth" would be among the highly ranked n-grams retrieved, but so would"Abraham Lincoln" and "the assassination of and "Ford's Theatre."Once the n-grams are scored, they are filtered by expected type. If the original querystarts with "who," then we filter on names of people; for "how many" we filter on numbers, for"when," on a date or time. There is also a filter that says the answer should not be part of thequestion: together these should allow us to return "John Wilkes Booth" (and not "AbrahamLincoln') as the highest-scoring response.In some cases the answer will be longer than three words: since the components re-sponses only go up to 3-grams. a longer response would have to be pieced together fromshorter pieces. For example, in a system that used only bigrams, the answer "John WilkesBooth" could be pieced together from high-scoring pieces "John Wilkes" and "Wilkes Booth."At the Text Retrieval Evaluation Conference (TREC), ASKMSR was rated as one ofthe top systems, beating out competitors with the ability to do far more complex languageunderstanding. ASKMSR relies upon the breadth of the content on the Web rather than onits own depth of understanding. It won't be able to handle complex inference patterns likeassociating "who killed" with "ended the life of." But it knows that the Web is so vast that it can afford to ignore passages like that and wait for a simple passage it can handle.

22,4 INFORMATION EXTRACTION

Information extraction is the process of acquiring knowledge by skimming a text and look-ing for occurrences of a particular class of object and for relationships among objects. Atypical task is to extract instances of addresses from Web pages, with database fields forstreet, city, state, and zip code; or instances of storms from weather reports, with fields fortemperature, wind speed, and precipitation. In a limited domain, this can be done with highaccuracy. As the domain gets more general, more complex linguistic models and more com-plex learning techniques are necessary. We will see in Chapter 23 how to define complexlanguage models of the phrase structure (noun phrases and verb

phrases) of English. But anfar there are no complete models of this kind, so for the limited needs of information ex-traction, we define limited models that approximate the full English model, and concentrateon just the parts that are needed for the task at hand. The models we describe in this sec-Lion are approximations in the same way that the simple 1-CNF logical model in Figure 7.21(page 271) is an approximations of the full, wiggly, logical model.In this section we describe six different approaches to information extraction, in orderof increasing complexity on several dimensions: deterministic to stochastic, domain-specificto general, hand-crafted to learned, and small-scale to large-scale.

22.4.1 Finite state automata for information extraction

The simplest type of information extraction system is an attribute-based extractionsystemthat assumes that the entire text refers to a single object and the task is to extract attributes ofthat object. For example, we mentioned in Section 12.7 the problem of extracting from thetext "IBM ThinkBook 970. Our price: $399.00" the set of attributes { Manufactmer=IBM,Model=ThinkI3ook970, Price=$399.00}. We can address this problem by defining a tem-plate (also known as a pattern) for each attribute we would like to extract The template isdefined by a finite state automaton, the simplest example of which is the regular expression_or regex. Regular expressions are used in Unix commands such as grep, in programminglanguages such as Peri, and in word processors such as Microsoft Word. The details varyslightly from one tool to another and so are best learned from the appropriate manual, buthere we show how to build up a regular expression template for prices in dollars;matches any digit from 0 to 9r 0-9 1matches one or more digits[0-9]+0-9-9[• 1 [] [G]I • 1 [0 -9] [0-9])?[0-9] + [.] [0-9] [0-9[matches a period followed by two digitsmatches a period followed by two digits, or nothing) ? matches $249.99 or $1.23 or $1000000 or ...Templates are often defined with three parts: a prefix regex, a target regex, and a postfix regex.For prices, the target regex is as above, the prefix would look for strings such as "price:" andthe postfix could be empty. The idea is that some clues about an attribute come from theattribute value itself and some come from the surrounding text.If a regular expression for an attribute matches the text exactly once, then we can pullout the portion of the text that is the value of the attribute. If there is no match, all we can dois give a default value or leave the attribute missing; but if there are several matches, we needa process to choose among them. One strategy is to have several templates for each attribute,ordered by priority. So, for example, the top-priority template for price might look for theprefix "our price:"; if that is not found, we look for the prefix "price:" and if that is not found,the empty prefix. Another strategy is to take all the matches and find some way to chooseamong them. For example, we could take the lowest price that is within 50% of the highestprice. That will select $78.00 as the target from the text "List price $99.00, special sale price$78.00, shipping $3.00."One step up from attribute-based extraction systems are relational extraction systems,which deal with multiple objects and the relations among them. Thus, when these systemssee the text "$249.99," they need to determine not just that it is a price, but also which objecthas that price. A typical relational-based extraction system is FASTUS, which handles newsstories about corporate mergers and acquisitions. It can read the story Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a localconcern and a Japanese trading house to produce golf clubs to be shipped to Japan.and extract the relations:e E Joint Ventures A Product(e,"golf clubs") A Date(e."Friday")A Member(6,"Bridgestone Sports Co") A Mernber(e,"a local concern")A Mentber(e,"a Japanese trading house") .CAnGASEDFINITE STATETRANSDUCERSA relational extraction system can be built as a series of cascaded finite-state transducers.That is, the system consists of a series of small, efficient finite-state automata (FSAs), whereeach automaton receives text as input, transduces the text into a different format, and passesit along to the next automaton. FASTUS consists of five stages:1.2.3.4.5.

Tokenization

Complex-word handling

Basic-group handling

Complex-phrase handling

Structure merging

FASTUS 's first stage is tokcniztition, which segments the stream of characters into tokens(words, numbers, and punctuation). For English, tokenization can be fairly simple; just sep-arating characters at white space or punctuation does a fairly good job. Some tokenizers alsodeal with markup languages such as HTML, SGML, and XML.The second stage handles complex words, including collocations such as "set up" and"joint venture," as well as proper names such as "Bridgestone Sports Co." These are rec-ognized by a combination of lexical entries and finite-state grammar rules. For example, acompany name might be recognized by the ruleCapitalized Word+ ("Company" "Co" "Inc" "Ltd")The third stage handles basic groups, meaning noun groups and verb groups. The idea isto chunk these into units that will be managed by the later stages. We will see how to writea complex description of noun and verb phrases in Chapter 23, but here we have simplerules that only approximate the complexity of English, but have the advantage of being rep-resentable by finite state automata. The example sentence would emerge from this stage asthe following sequence of tagged groups:123456789NG: Bridgestone Sports Co.VC: saidNG: FridayNG! itVG: had set upNG: a joint venturePR: inNG: TaiwanPR: with1011121314151617NG:CJ:NG:VG:NG:VG:PR:NG:a local concernanda Japanese trading houseto producegolf clubsto be shippedtoJapanHere NG means noun group, VG is verb group, PR is preposition, and Cl is conjunction.The fourth stage combines the basic groups into complex phrases. Again, the aimis to have rules that arc finite-state and thus can be processed quickly, and that result inunambiguous for nearly unambiguous) output phrases. One type of combination rule dealswith domain-specific events. For example, the ruleCompany+ SetUp JointVenture ("with" Company+)?captures one way to describe the formation of a joint venture. This stage is the first one inthe cascade where the output is placed into a database template as well as being placed in theoutput stream. The final stage merges structures that were built up in the previous step. Ifthe next sentence says "The joint venture will start production in January," then this step willnotice that there are two references to a joint venture, and that they should be merged intoone This is an instance of the identity, uncertainty problem discussed in Section 14.6.3.In general, finite-state template-based information extraction works well for a restricteddomain in which it is possible to predetermine what subjects will be discussed, and how theywill be mentioned. The cascaded transducer model helps modularize the necessary knowl-edge, easing construction of the system. These systems work especially well when they arereverse-engineering text that has been generated by a program. For example, a shopping siteon the Web is generated by a program that takes database entries and formats them into Webpages; a template-based extractor then recovers the original database. Finite-state informa-tion extraction is less successful at recovering information in highly variable format, such astext written by humans on a variety of subjects.22.4.2 Probabilistic models for information extractionWhen information extraction must be attempted from noisy or varied input, simple finite-stateapproaches fare poorly. It is too hard to get all the rules and their priorities right; it is betterto use a probabilistic model rather than a rule-based model. The simplest probabilistic modelfor sequences with hidden state is the hidden Marlow model, or HMM.Recall from Section 15.3 that an HMM models a progression through a sequence ofhidden states, x_t, with an observation e_t at each step. To apply HMMs to information ex-traction, we can either build one big HMM for all the attributes or build a separate HMMfor each attribute. We'll do the second. The observations are the words of the text, and thehidden states are whether we are in the target, prefix, or postfix part of the attribute template,or in the background (not part of a template). For example, here is a brief text and the mostprobable (Viterbi) path for that text for two HMMs, one trained to recognize the speaker in atalk announcement, and one trained to recognize dates. The "-" indicates a background
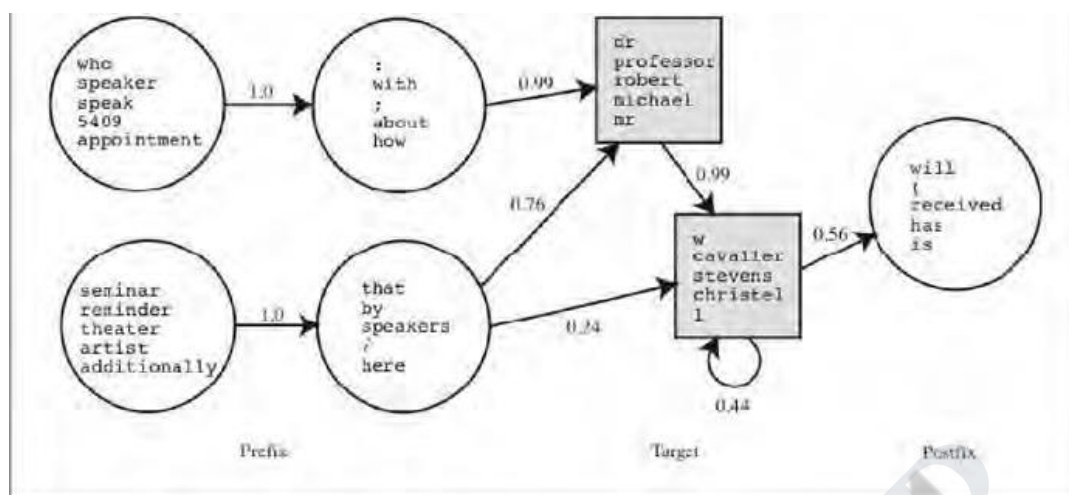
state:Text:There will be aSpeaker: -Date:-seminar by Dr.Andrew McCallum onFriday- - PREPRE TARGET TARGET TARGET POST -PRE TARGETHMMs have two big advantages over FSAs for extraction. First, HMMs are probabilistic, andthus tolerant to noise. In a regular expression, if a single expected character is missing, theregex fails to match; with HMMs there is graceful degradation with missing characters/words,and we get a probability indicating the degree of match, not just a Boolean match/fail. Second.HMV's can be trained from data; they don't require laborious engineering of templates, andthus they can more easily be kept up to date as text changes over time.Note that we have assumed a certain level of structure in our HMM templates: they allconsist of one or more target states. and any prefix states must precede the targets, postfixstates most follow the targets, and other states must be background. This structure makesit easier to learn HMMs from examples. With a partially specified structure, the forward–backward algorithm can be used to learn both the transition probabilities $P(X_t | X_{5-1})$ be-tween states and the observation model, $P(E_t | X_t)$, which says how likely each word is ineach state. For example, the word "Friday" would have high probability in one or more ofthe target states of the date IIMM, and lower probability elsewhere.With sufficient training data, the HMM automatically learns a structure of dates that wefind intuitive: the date HMM might have one target state in which the high-probability wordsare "Monday," "Tuesday," etc., and which has a high-probability transition to a target statewith words "Jan", "January," "Feb," etc. Figure 22.2 shows the HMM for the speaker of atalk announcement, as learned from data. The prefix covers expressions such as "Speaker:"and "seminar by," and the target has one state that covers titles and first names and anotherstate that covers initials and last names.Once the HMMs have been learned, we can apply them to a text, using the Viterbialgorithm to find the most likely path through the HMM states. One approach is to applyeach attribute HMM separately; in this case you would expect most of the IIMMs to spendmost of their time in background states. This is appropriate when the extraction is sparse—when the number of extracted words is small compared to the length of the text.The other approach is to combine all the individual attributes into one big HMM, whichwould then find a path that wanders through different target attributes, first finding a speakertarget, then a date target, etc. Separate HMMs are better when we expect just one of eachattribute in a text arid one big HMM is better when the texts are more free-form and densewith attributes. With either approach, in the end we have a collection of target attributeobservations, and have to decide what to do with them. If every expected attribute has onetarget filler then the decision is easy: we have an instance of the desired relation. If thereare multiple fillers, we need to decide which to choose, as we discussed with template-basedsystems. HMMs have the advantage of supplying probability numbers that can help makethe choice. If some targets are missing, we need to decide if this is an instance of the desiredrelation at all, or if the targets found are false positives. A machine learning algorithm can betrained to make this choice.22.4.3 Conditional random fields for information extraction

One issue with HMMs for the information extraction task is that they model a lot of prob-abilities that we don't really need. An HMM is a generative model; it models the full joint probability of observations and hidden states, and thus can be used to generate samples. That is, we can use the HMM model not only to parse a text and recover the speaker and date, but also to generate a random instance of a text containing a speaker and a date. Since we're not interested in that task, it is natural to ask whether we might be better off with a model that doesn't bother modeling that possibility. All we need in order to understand a text is a discriminative model, one that models the conditional probability of the hidden attributes given the observations (the text). Given a text $e_{i,N}$, the conditional model finds the hidden state sequence X_i, N that maximizes $P(X LN e_i, N)$.

COND1110NAL
19APDOM FIELD
LINEAR-CHAIV
CORDITIONAL
RAKDOM FIELD

Modeling this directly gives us some freedom. We don't need the independence as-sumptions of the Markov model—we can have an x_t that is dependent on x_1. A framework for this type of model is the conditional random field, or CRF, which models a conditional probability distribution of a set of target variables given a set of observed variables. Like Bayesian networks, CRFs can represent many different structures of dependencies among the variables. One common structure is the linean chain conditional random field for repre-senting Markov dependencies among variables in a temporal sequence. Thus, HMMs are the temporal version of naive Bayes models, and linear-chain CRFs are the temporal version of logistic regression, where the predicted target is an entire state sequence rather than a single binary variable. Let $e_{i,N}$ be the observations (e.g., words in a document), and x_i, N be the sequence of hidden states (e_g., the prefix, target. and pnstfix states), A linear-chain conditional random field defines a conditional probability distribution !N F x,x,c,i)](-nP(xl:Nleniv) = e [E where a is a normalization factor (to make sure the probabilities sum to 1), and F is a feature function defined as the weighted sum of a collection of k component feature functions: $F(xj\_i,xi, e,i) =k(.,\_,,, xi, e, i)$ .The Ak parameter values are learned with a MAP (maximum a posteriori) estimation proce-dure that maximizes thc conditional likelihood of thc training data. The feature functions arc the key components of a CRF. The function fa has access to a pair of adjacent states, xi_i and xi, but also the entire observation (word) sequence e, and the current position in the temporal sequence, i. This gives us a lot of flexibility in defining features. We can define a simple feature function, for example one that produces a value of 1 if the current word is ANDREW and the current state is SPEAKER: I l if xi = SPEAKER and ei = ANDREW h(x i _ l , xi, e, = I) otherwise How are features like these used? It depends on their corresponding weights. If $Ai > 0$, then whenever fi is tore, it increases the probability of the hidden state sequence $x l , Ar$. This is another way of saying "the CRF model should prefer the target state SPEAKER for the word ANDREW." If on the other hand $At < O$. the CRF model will try to avoid this association, and if $AI = 0$, this feature is ignored. Parameter values can be set manually or can be learned from data. Now consider a second feature function: { 1 if = SPEAKER and $e i+i$ = SAIDxi. e.0 otherwise This feature is true if the current state is SPEAKER and the next word is "said." One would therefore expect a positive $)12$ value to go with the feature. More interestingly, note that both f i and h can hold at the same time for a sentence like "Andrew said ...." In this case, the two features overlap each other and both boost the belief in $x i$ = SPEAKER. Because of the independence assumption, HMMs cannot use overlapping features; CRFs can. Furthermore, a feature in a CRF can use any part of the sequence $e i , N$. Features can also be defined over transitions between states. The features we defined here were binary, but in general, a feature function can be any real-valued function. For domains where we have some knowledge about the types of features we would like to

include, the CRF formalism gives us a great deal offlexibility in defining them. This flexibility can lead to accuracies that are higher than withless flexible models such as HMMs.

22.4.4 Ontology extraction from large corpora

So far we have thought of information extraction as finding a specific set of relations (e.g.,speaker, time, location) in a specific text (e.g., a talk announcement). A different applica-tion of extraction technology is building a large knowledge base or ontology of facts froma corpus. This is different in three ways: First it is open-ended—we want to acquire factsabout all types of domains, not just one specific domain. Second, with a large corpus, thistask is dominated by precision, not recall—just as with question answering on the Web (Sec-tion 22.3.6). Third, the results can be statistical aggregates gathered from multiple sources,rather than being extracted from one specific text.For example, Hearst (1992) looked at the problem of learning an ontology of conceptcategories and subcategories from a large corpus. (In 1992, a large corpus was a 1000-pageencyclopedia; today it would be a 100-million-page Web corpus.) The work concentrated ontemplates that are very general (not tied to a specific domain) and have high precision (arealmost always correct when they match) but low recall (do not always match). Here is one ofthe most productive templates:NP such as NP (, NP)* (,)? ((and or) NP)`?Here the bold words and commas must appear literally in the text, but the parentheses arefor grouping, the asterisk means repetition of zero or more, and the question mark meansoptional. NA is a variable standing for a noun phrase; Chapter 23 describes how to identifynoun phrases; for now just assume that we know some words are nouns and other words (suchas verbs) that we can reliably assume are not part of a simple noun phrase. This templatematches the texts "diseases such as rabies affect your dog" and "supports network protocolssuch as DNS," concluding that rabies is a disease and DNS is a network protocol. Similartemplates can be constructed with the key words "including," "especially," and "or other." Ofcourse these templates will fail to match many relevant passages, like "Rabies is a disease."That is intentional. The "NP is a NP" template does indeed sometimes denote a subcategoryrelation, but it often means something else, as in "There is a God" or "She is a little tired."With a large corpus we can afford to be picky; to use only the high-precision templates. We'llMISS many statements of a subcategory relationship, but most likely we'll find a paraphraseof the statement somewhere else in the corpus in a form we can use.22.4.5 Automated template constructionThe subcategory relation is so fundamental that is worthwhile to handcraft a few templates tohelp identify instances of it occurring in natural language text. But what about the thousandsof other relations in the world? There aren't enough AI grad students in the world to createand debug templates for all of them. Fortunately, it is possible to learn templates from a fewexamples, then use the templates to learn more examples, from which more templates can belearned, and so on. In one of the first experiments of this kind, Brin (1999) started with a dataset of just five examples:("Isaac Asirnov", "The Robots of Dawn")

("David Brin", "Startide Rising")

("James Gleick", "Chaos—Making a New Science")

("Charles Dickens", "Great Expectations")

(" William Shakespeare", "The Comedy of Errors")

Clearly these are examples of the author—title relation, but the learning system had no knowl-edge of authors or titles. The words in these examples were used in a search over a Webcorpus, resulting in 199 matches. Each match is defined as a tuple of seven strings,(Author, Title, Order; Prefix, Middle, Posybc, URL) ,where Order is true if the author came first and false if the title came first, Middle is thecharacters between the author and title, Prefix is the 10 characters before the match, Suffix isthe 10 characters after the match, and URL is the Web address where the match was made.Given a set of matches, a simple template-generation scheme can find templates toexplain the matches. The language of templates was designed Lu have a close mapping to thematches themselves, to be amenable to automated learning, and to emphasize high precisionpossibly at the risk of lower recall).

Each template has the same seven components as amatch. The Author and Title arc rcgcxcs consisting of any characters (but beginning andending in letters) and constrained to have a length from half the minimum length of theexamples to twice the maximum length. The prefix, middle, and postfix are restricted toliteral strings, not regexes. The middle is the easiest to learn: each distinct middle string inthe set of matches is a distinct candidate template. For each such candidate, the template'sPrefix is then defined as the longest common suffix of all the prefixes in the matches, and thePostfix is defined as the longest common prefix of all the postfixes in the matches. If either ofthese is of length zero, then the template is rejected. The URL of the template is defined asthe longest prefix of the LIRLs in the matches.In the experiment run by Brin, the first 199 matches generated three templates. Themost productive template was.<1,I><B> Ede </B> by Author (URL: www.sff . net/ locus /cThe three templates were then used to retrieve 4047 more (author, title) examples. The exam-ples were then used to generate more templates, and so on, eventually yielding over 15,000titles. Given a good set of templates, the system can collect a good set of examples. Given agood set of examples, the system can build a good set of templates.The biggest weakness in this approach is the sensitivity to noise. If one of the firstfew templates is incorrect, errors can propagate quickly. One way to limit this problem is tonot accept a new example unless it is verified by multiple templates, and not accept a newtemplate unless it discovers multiple examples that are also found by other templates.

22.4.6 Machine reading

MACHINE READING

Automated template construction is a big step up from handcrafted template construction, butit still requires a handful of labeled examples of each relation to get started. To build a largeontology with many thousands of relations, even that amount of work would be onerous; wewould like to have an extraction system with no human input of any kind—a system that couldread on its own and build up its own database. Such a system would he relation-independent;would work for any relation. In practice, these systems work on all relations in parallel,because of the I/O demands of large corpora_ They behave less like a traditional information-extraction system that is targeted at a few relations and more like a human reader who learnsfrom the text itself; because of this the field has been called machine reading.A representative machine-reading system is TEXTRUNNER (Banko and Etzioni, 2008).TEXTRUNNER uses cotraining to boost its performance, but it needs something to bootstrapfrom. In the case of Hearst (1992), specific patterns (e.g., such as) provided the bootstrap, andfor Brin (1998), it was a set of five author–title pairs. For TEXTRUNNER, the original inspi-ration was a taxonomy of eight very general syntactic templates, as shown in Figure 22.3. Itwas felt that a small number of templates like this could cover most of the ways that relation-ships are expressed in English. The actual bootsrapping starts from a set of labelled examplesthat are extracted from the Penn Treebank, a corpus of parsed sentences. For example, fromthe parse of the sentence "Einstein received the Nobel Prize in 1921," TEXTRUNNER is ableextract the relation ("Einstein," "received," "Nobel Prize").Given a sct of labeled examples of this type, TEXTRUNNER trains a linear-chain CRFto extract further examples from unlabeled text. The features in the CRF include functionwords like "to - and "of" and 'the," but not nouns and verbs (and not noun phrases or verbphrases). Because TEXTRUNNER is domain-independent, it cannot rely on predefined listsof nouns and verbs.to'ISrpe Template Example Frequency

Verb NP I Verb NP 2

NP 1 NP Prep NP 2

NP I Verb Prep NP 2

NP 1 to Verb NP 2

NP I Verb NP 2 Noun

NP 1 (, 1 and - :) NP 2 NP X established Y

X settlement with YX moved to YX plans to acquire Y 38%23%16%Noun—PrepVerb — PrepInfinitiveModifierNoun-CoordinateVerb - CoordinateAppositiveNP 1 CIand) NP 2 VerbNP 1 NP (: ,)? NP 2Xis Y winnerX - Y dealX, Y mergeX hometown : Y9%5%2%1%I% Figure 22.3 Eight general templates that cover about 95% of the ways that relations are expressed in English.TEXTRUNNER achieves a precision of 88% and recall of 45% (Ft of 60%) on a largeTEXTRUNNER has extracted hundreds of millions of facts from a corpus of ahalf-billion Web pages. For example, even though it has no predefined medical knowledge,Web corpus.it has extracted over 2000 answers to the query [what kills bacteria]; correct answers includeantibiotics, ozone, chlorine, Cipro, and broccoli sprouts. Questionable answers include "wa-ter," which came from the sentence "Boiling water for at least ID minutes will kill bacteria."It would be better to attribute this to "boiling water" rather than just "water."With the techniques outlined in this chapter and continual new inventions, we are start-ing to get closer to thegoal of machine reading.