# ARTIFICIAL INTELLIGENCE
# UNIT-I Question-Bank
# Introduction

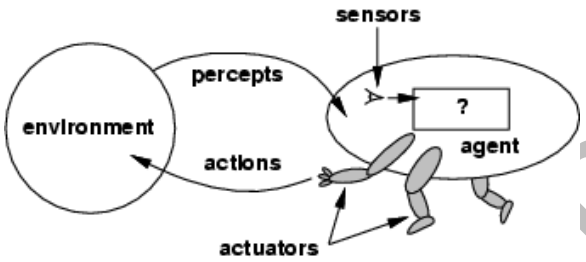
---

**1) What is AI?**

| | |
|---|---|
| Systems that think like humans | Systems that think rationally |
| Systems that act like humans | Systems that act rationally |

**2) Define an agent.**

An **agent** is anything that can be viewed as **perceiving** its **environment** through **sensors** and **acting upon** that environment through **actuators.**




**3) What is an agent function?**

An agent's behavior is described by the **agent function** that maps any given **percept sequence** to an **action.**

**4) Differentiate an agent function and an agent program.**

| Agent Function | Agent Program |
|---|---|
| An abstract mathematical description | A concrete implementation,running on the agent Architecture. |

**5) What can Ai do today?**

- Autonomous Planning and Scheduling
  - Spacecraft control
  - Goal-directed planning, detection, diagnosis, problem recovery
- Game Planing
  - IBM Deep Blue
  - World chess champion
- Autonomous Control
  - CMU NAVLAB
  - Computer-controlled mini-van
  - Crossed the US without human control over 98% of the time

Powered By www.technoscriptz.com

- Diagnosis
  - Medical diagnosis in several areas of medicine (e.g., pathology)
  - Explanation, justification for decisions
- Logistics Planning
  - DOD's Dynamic Analysis and Replanning Tool
  - Logistics planning of 50,000 vehicles, cargo, people
  - Embarkation, destination, route, conflict resolution
  - Paid back all of DARPA's 30-year investment in AI
- Robotics
  - Robotic surgical assistants
  - Cooperating autonomous robots in reconnaissance
  - Exploration of the Solar System

**6) What is a task environment? How it is specified?**

**Task environments** are essentially the "problems" to which rational agents are the "solutions."

A Task environment is specified using PEAS (Performance, Environment, Actuators, Sensors) description.

**7) Give an example of PEAS description for an automated taxi.**

| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Taxi driver | Safe: fast, legal, comfortable trip, maximize profits | Roads, other traffic, pedestrians, customers | Steering, accelerator, brake, signal, horn, display | Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard |

**Figure 2.4** PEAS description of the task environment for an automated taxi.

**8) Give PEAS description for different agent types.**

Notes-me.com

| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Medical diagnosis system | Healthy patient, minimize costs, lawsuits | Patient, hospital, staff | Display questions, tests, diagnoses, treatments, referrals | Keyboard entry of symptoms, findings, patient's answers |
| Satellite image analysis system | Correct image categorization | Downlink from orbiting satellite | Display categorization of scene | Color pixel arrays |
| Part-picking robot | Percentage of parts in correct bins | Conveyor belt with parts; bins | Jointed arm and hand | Camera, joint angle sensors |
| Refinery controller | Maximize purity, yield, safety | Refinery, operators | Valves, pumps, heaters, displays | Temperature, pressure, chemical sensors |
| Interactive English tutor | Maximize student's score on test | Set of students, testing agency | Display exercises, suggestions, corrections | Keyboard entry |

**Figure 2.5** Examples of agent types and their PEAS descriptions.

**9) List the properties of task environments.**

- Fully observable vs. partially observable**.**
- Deterministic vs. stochastic.
- Episodic vs. sequential
- Static vs, dynamic.
- Discrete vs. continuous.
- Single agent vs. multiagent**.**

**10) Write a function for the table driven agent.**

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  static: percepts, a sequence, initially empty
          table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

**Figure 2.7** The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It keeps track of the percept sequence using its own private data structure.

**Powered By www.technoscriptz.com**

**11) What are the four different kinds of agent programs?**

Simple reflex agents;
Model-based reflex agents;
Goal-based agents; and
Utility-based agents.

**12) Explain a simple reflex agent with a diagram.**

**Simple reflex agents**

The simplest kind of agent is the **simple reflex agent.** These agents select actions on the basis AGENT of the *current* percept, ignoring the rest of the percept history.

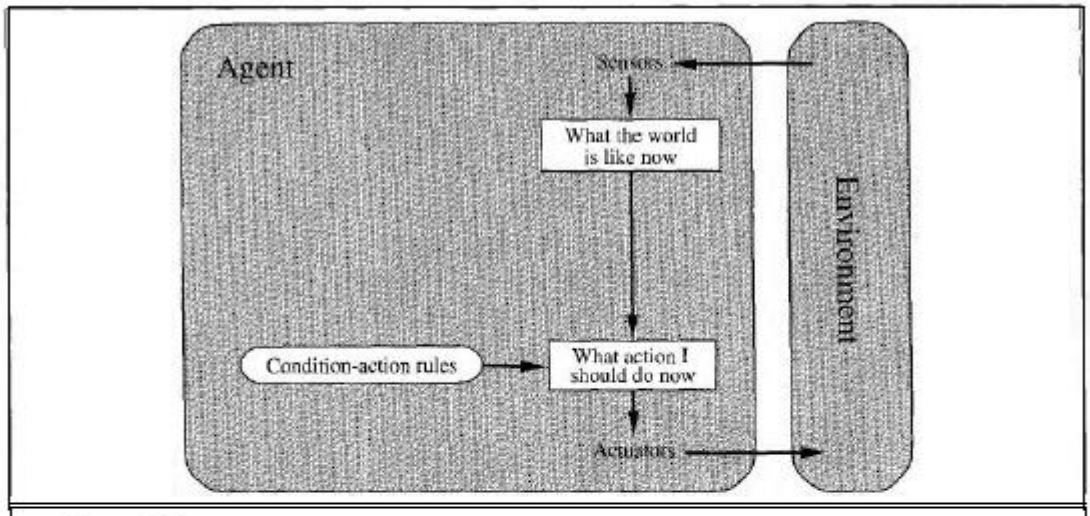



**Figure 2.9** Schematic diagram of a simple reflex agent.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  static: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

**Figure 2.10** A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

**13) Explain with a diagram the model based reflex agent.**

**Powered By www.technoscriptz.com**

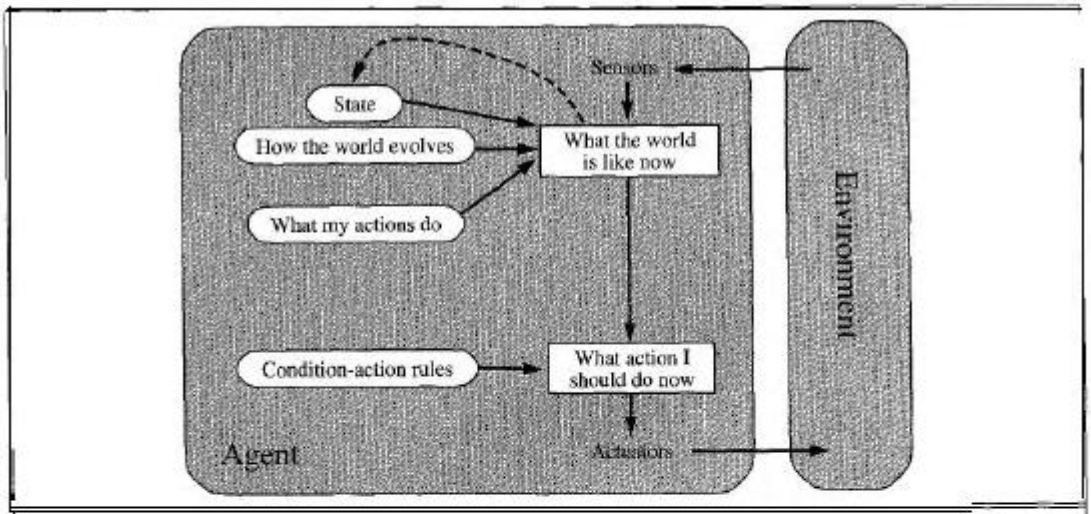



**Figure 2.11** *A* model-based reflex agent.

```
function REFLEX-AGENT-WITH-STATE(percept) returns 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

  state ← UPDATE-STATE(state, action, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

**Figure 2.12** A model-based reflex agent. It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.

**13a) Explain with a diagram the goal based reflex agent.**

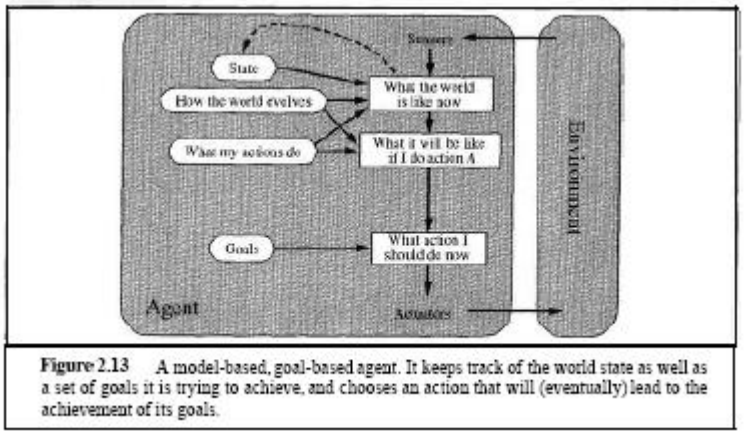



**Figure 2.13** A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes

**Powered By www.technoscriptz.com**

situations that are desirable-for example, being at the passenger's destination.

**13b) What are utility basedagents?**

Goals alone are not really enough to generate high-quality behavior in most environments. For example, there are many action sequences that will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others.
A **utility function** maps a state (or a sequence of states) onto a real number, which describes the associated degree of happiness.

**13c) What are learning agents?**

A learning agent can be divided into four conceptual components, as shown in Fig-2.15. The most important distinction is between the learning element, which is re-
ELEMENT sponsible for making improvements, and the performance element, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses
CRITIC feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

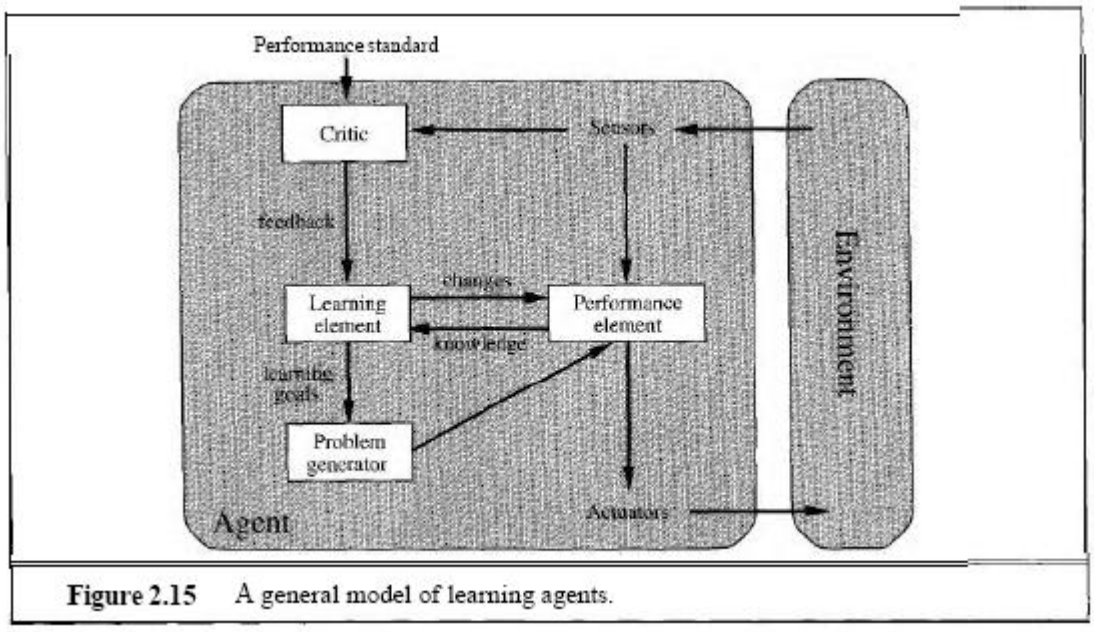



**Figure 2.15** A general model of learning agents.

## Searching Techniques

**14) Define the problem solving agent.**

A Problem solving agent is a **goal-based** agent . It decide what to do by finding sequence of actions that lead to desirable states. The agent can adopt a goal and aim at satisfying it.
Goal formulation is the first step in problem solving.

**15) Define the terms goal formulation and problem formulation.**

**Goal formulation**,based on the current situation and the agent's performance measure,is the first step in problem solving.

The agent's task is to find out which sequence of actions will get to a goal state.

**Problem formulation** is the process of deciding what actions and states to consider given a goal

**16) List the steps involved in simple problem solving agent.**

(i) Goal formulation
(ii) Problem formulation
(iii) Search
(iv) Search Algorithm
(v) Execution phase

**17) Define search and search algorithm.**

The process of looking for sequences actions from the current state to reach the goal state is called **search.**

The **search algorithm** takes a **problem** as **input** and returns a **solution** in the form of **action sequence.** Once a solution is found,the **execution phase** consists of carrying out the recommended action..

**18) What are the components of well-defined problems?**

- The **initial state** that the agent starts in . The initial state for our agent of example problem is described by *In(Arad)*
- A **Successor Function** returns the possible **actions** available to the agent. Given a state x,SUCCESSOR-FN(x) returns a set of {action,successor} ordered pairs where each action is one of the legal actions in state x,and each successor is a state that can be reached from x by applying the action.
  For example,from the state In(Arad),the successor function for the Romania problem would return

{ [Go(Sibiu),In(Sibiu)],[Go(Timisoara),In(Timisoara)],[Go(Zerind),In(Zerind)] }

- Thr **goal test** determines whether the given state is a goal state.

- A **path cost** function assigns numeric cost to each action. For the Romania problem the cost of path might be its length in kilometers.

**19) Differentiate toy problems and real world problems.**

| TOY PROBLEMS | REAL WORLD PROBLEMS |
|---|---|
| A **toy problem** is intended to illustrate various problem solving methods. It can be easily used by different researchers to compare the performance of algorithms. | A **real world problem** is one whose solutions people actually care about. |

**20) Give examples of real world problems.**

(i) Touring problems
(ii) Travelling Salesperson Problem(TSP)
(iii) VLSI layout
(iv) Robot navigation
(v) Automatic assembly sequencing
(vi) Internet searching

**21) List the criteria to measure the performance of different search strategies.**

Notesengine.com

- **Completeness** : Is the algorithm guaranteed to find a solution when there is one?
- **Optimality** : Does the strategy find the optimal solution?
- **Time complexity** : How long does it take to find a solution?
- **Space complexity** : How much memory is needed to perform the search?

**22) Differentiate Uninformed Search(Blind search) and Informed Search(Heuristic Search) strategies.**

| Uninformed or Blind Search | Informed or Heuristic Search |
|---|---|
| ○ No additional information beyond that provided in the problem definition<br>○ Not effective<br>○ No information about number of steps or path cost | ○ More effective<br>○ Uses problem-specific knowledge beyond the definition of the problem itself. |

**23) Define Best-first-search.**

Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on the evaluation function f(n ). Traditionally,the node with the *lowest* evaluation function is selected for expansion.

Notesengine.com

---

# CS2351 –ARTIFICIAL INTELLIGENCE

**VI SEMESTER CSE**

---

# UNIT-II
# (2) SEARCHING TECHNIQUES

---

## 2.1 INFORMED SEARCH AND EXPLORATION

**2.1.1 Informed(Heuristic) Search Strategies**
**2.1.2 Heuristic Functions**
**2.1.3 Local Search Algorithms and Optimization Problems**
**2.1.4 Local Search in Continuous Spaces**
**2.1.5 Online Search Agents and Unknown Environments**

---

## 2.2 CONSTRAINT SATISFACTION PROBLEMS(CSP)

**2.2.1 Constraint Satisfaction Problems**
**2.2.2 Backtracking Search for CSPs**
**2.2.3 The Structure of Problems**

---

## 2.3 ADVERSARIAL SEARCH

**2.3.1 Games**
**2.3.2 Optimal Decisions in Games**
**2.3.3 Alpha-Beta Pruning**
**2.3.4 Imperfect ,Real-time Decisions**
**2.3.5 Games that include Element of Chance**

---

Notesengine.com

## 2.1 INFORMED SEARCH AND EXPLORATION

**2.1.1 Informed(Heuristic) Search Strategies**

**Informed search strategy** is one that uses problem-specific knowledge beyond the definition of the problem itself. It can find solutions more efficiently than uninformed strategy.

**Best-first search**

**Best-first search** is an instance of general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function** f(n). The node with lowest evaluation is selected for expansion,because the evaluation measures the distance to the goal. This can be implemented using a priority-queue,a data structure that will maintain the fringe in ascending order of f-values.

### 2.1.2. Heuristic functions

A **heuristic function** or simply a **heuristic** is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search.

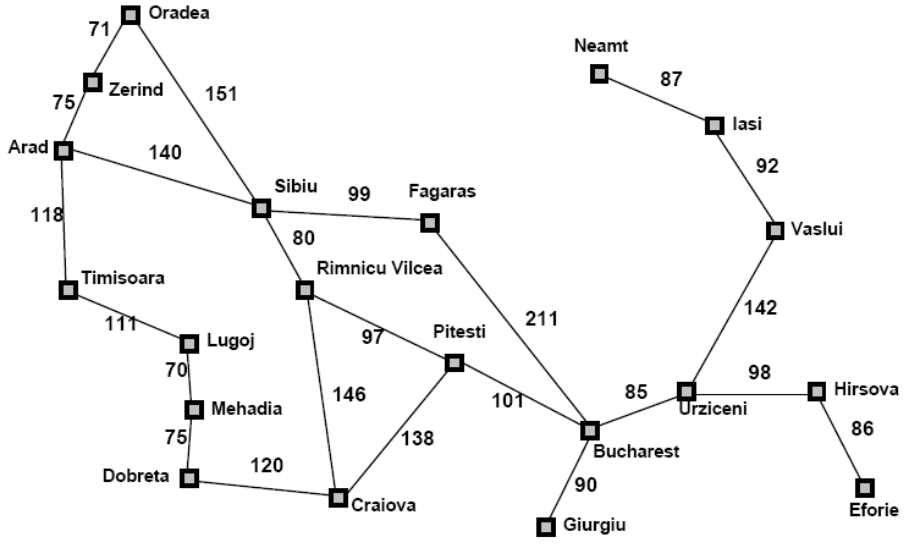
The key component of Best-first search algorithm is a **heuristic function**,denoted by h(n):

h(n) = extimated cost of the **cheapest path** from node n to a **goal node**.

For example,in Romania,one might estimate the cost of the cheapest path from Arad to Bucharest via a **straight-line distance** from Arad to Bucharest(Figure 2.1).
Heuristic function are the most common form in which additional knowledge is imparted to the search algorithm.

**Greedy Best-first search**

**Greedy best-first search** tries to expand the node that is closest to the goal,on the grounds that this is likely to a solution quickly.
It evaluates the nodes by using the heuristic function f(n) = h(n).
Taking the example of **Route-finding problems** in Romania , the goal is to reach Bucharest starting from the city Arad. We need to know the straight-line distances to Bucharest from various cities as shown in Figure 2.1. For example, the initial state is In(Arad) ,and the straight line distance heuristic $h_{SLD}$(In(Arad)) is found to be 366.
Using the **straight-line distance** heuristic $\mathbf{h_{SLD}}$ ,the goal state can be reached faster.




| Straight−line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

**Figure 2.1** Values of $h_{SLD}$ - straight line distances to Bucharest

Powered By www.technoscriptz.com

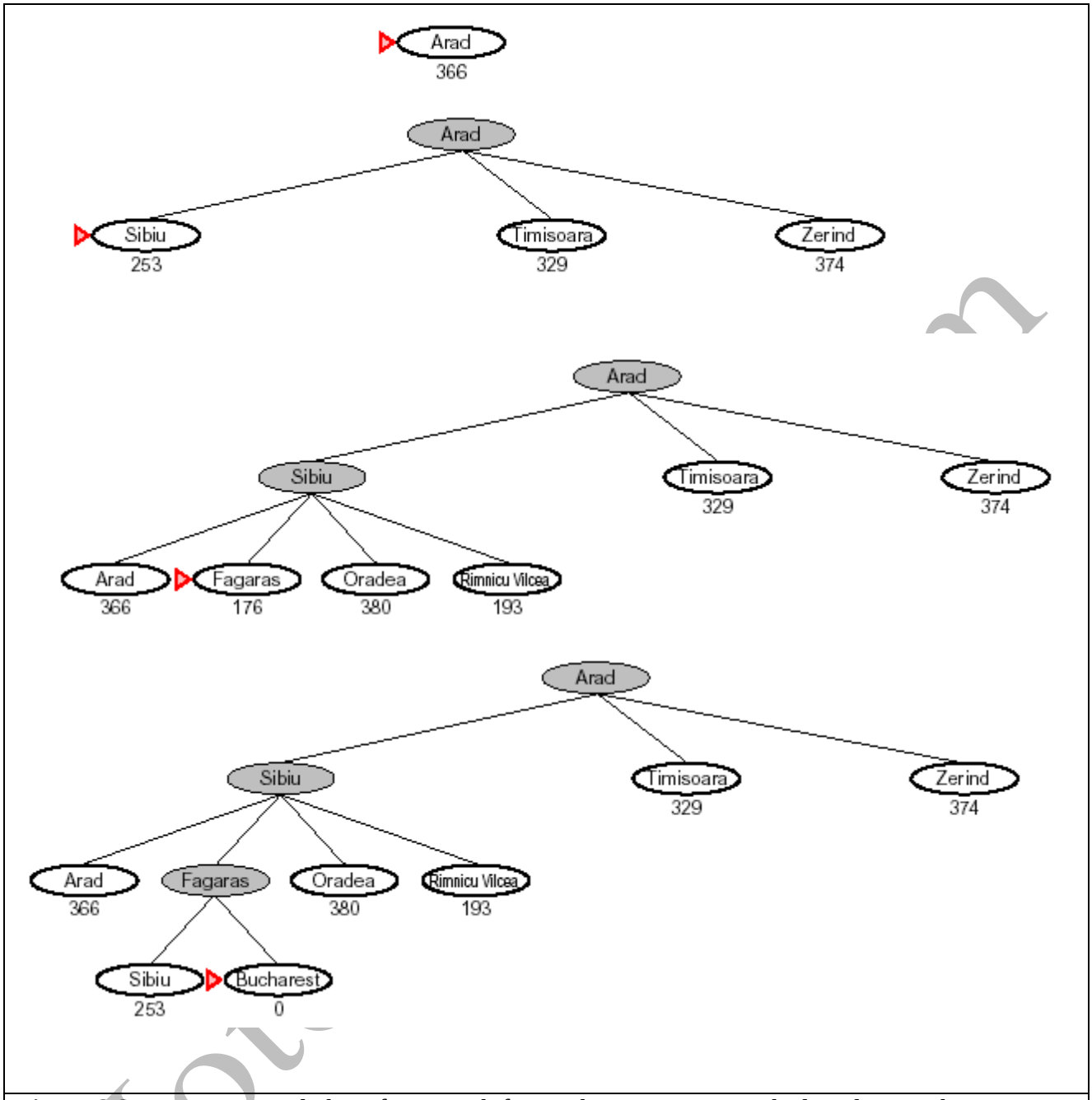



**Figure 2.2** stages in greedy best-first search for Bucharest using straight-line distance heuristic $h_{SLD}$. Nodes are labeled with their h-values.

Figure 2.2 shows the progress of greedy best-first search using $h_{SLD}$ to find a path from Arad to Bucharest. The first node to be expanded from Arad will be Sibiu,because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras,because it is closest. Fagaras in turn generates Bucharest,which is the goal.

**Powered By www.technoscriptz.com**

Properties of greedy search

- **Complete??** No–can get stuck in loops, e.g.,
  Iasi ! Neamt ! Iasi ! Neamt !
  Complete in finite space with repeated-state checking
- **Time??** O(bm), but a good heuristic can give dramatic improvement
- **Space??** O(bm)—keeps all nodes in memory
- **Optimal??** No

Greedy best-first search is not optimal,and it is incomplete.
The worst-case time and space complexity is $O(b^m)$,where m is the maximum depth of the search space.

## A* Search

**A* Search** is the most widely used form of best-first search. The evaluation function f(n) is obtained by combining

(1) **g(n) =** the cost to reach the node,and
(2) **h(n) =** the cost to get from the node to the **goal** :

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

$A^*$ Search is both optimal and complete. $A^*$ is optimal if h(n) is an admissible heuristic. The obvious example of admissible heuristic is the straight-line distance $h_{SLD}$. It cannot be an overestimate.
$A^*$ Search is optimal if h(n) is an admissible heuristic – that is,provided that h(n) never overestimates the cost to reach the goal.
An obvious example of an admissible heuristic is the straight-line distance $h_{SLD}$ that we used in getting to Bucharest. The progress of an $A^*$ tree search for Bucharest is shown in Figure 2.2.
The values of 'g ' are computed from the step costs shown in the Romania map( figure 2.1). Also the values of $h_{SLD}$ are given in Figure 2.1.

## Recursive Best-first Search(RBFS)

Recursive best-first search is a simple recursive algorithm that attempts to mimic the operation of standard best-first search,but using only linear space. The algorithm is shown in figure 2.4.
Its structure is similar to that of recursive depth-first search,but rather than continuing indefinitely down the current path,it keeps track of the f-value of the best alternative path available from any ancestor of the current node. If the current node exceeds this limit,the recursion unwinds back to the alternative path. As the recursion unwinds,RBFS replaces the f-value of each node along the path with the best f-value of its children.
Figure 2.5 shows how RBFS reaches Bucharest.

Notesengine.com

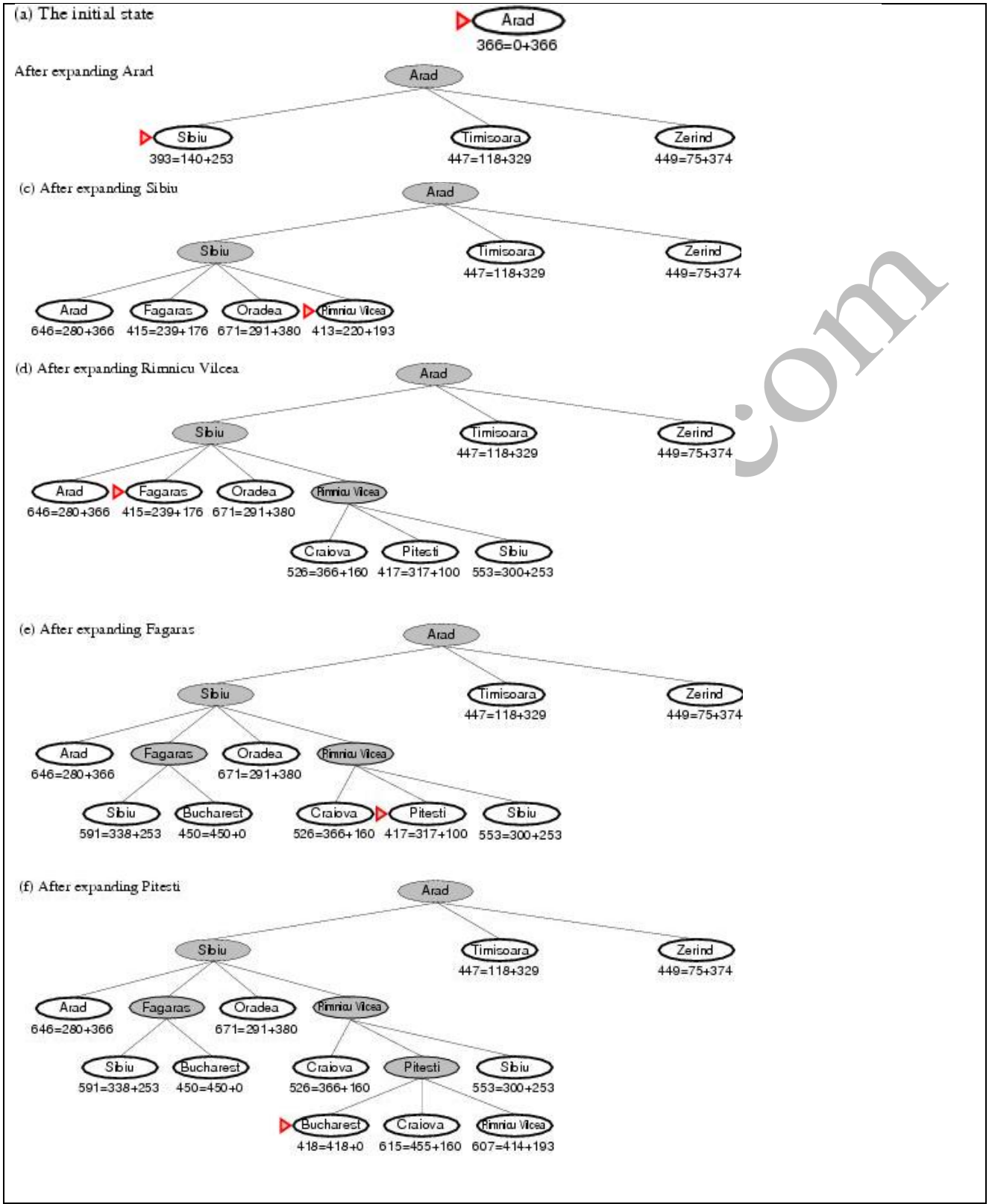



**Figure 2.3** Stages in A* Search for Bucharest. Nodes are labeled with f = g + h . The h-values are the straight-line distances to Bucharest taken from figure 2.1

.com

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RFBS( problem, node, f_limit) return a solution or failure and a new f-
    cost limit
    if GOAL-TEST[problem](STATE[node]) then return node
    successors ← EXPAND(node, problem)
    if successors is empty then return failure, ∞
    for each s in successors do
        f [s] ← max(g(s) + h(s), f [node])
    repeat
        best ← the lowest f-value node in successors
        if f [best] > f_limit then return failure, f [best]
        alternative ← the second lowest f-value among successors
        result, f [best] ← RBFS(problem, best, min(f_limit, alternative))
        if result ≠ failure then return result
```

**Figure 2.4** The algorithm for recursive best-first search

Notesengine.com

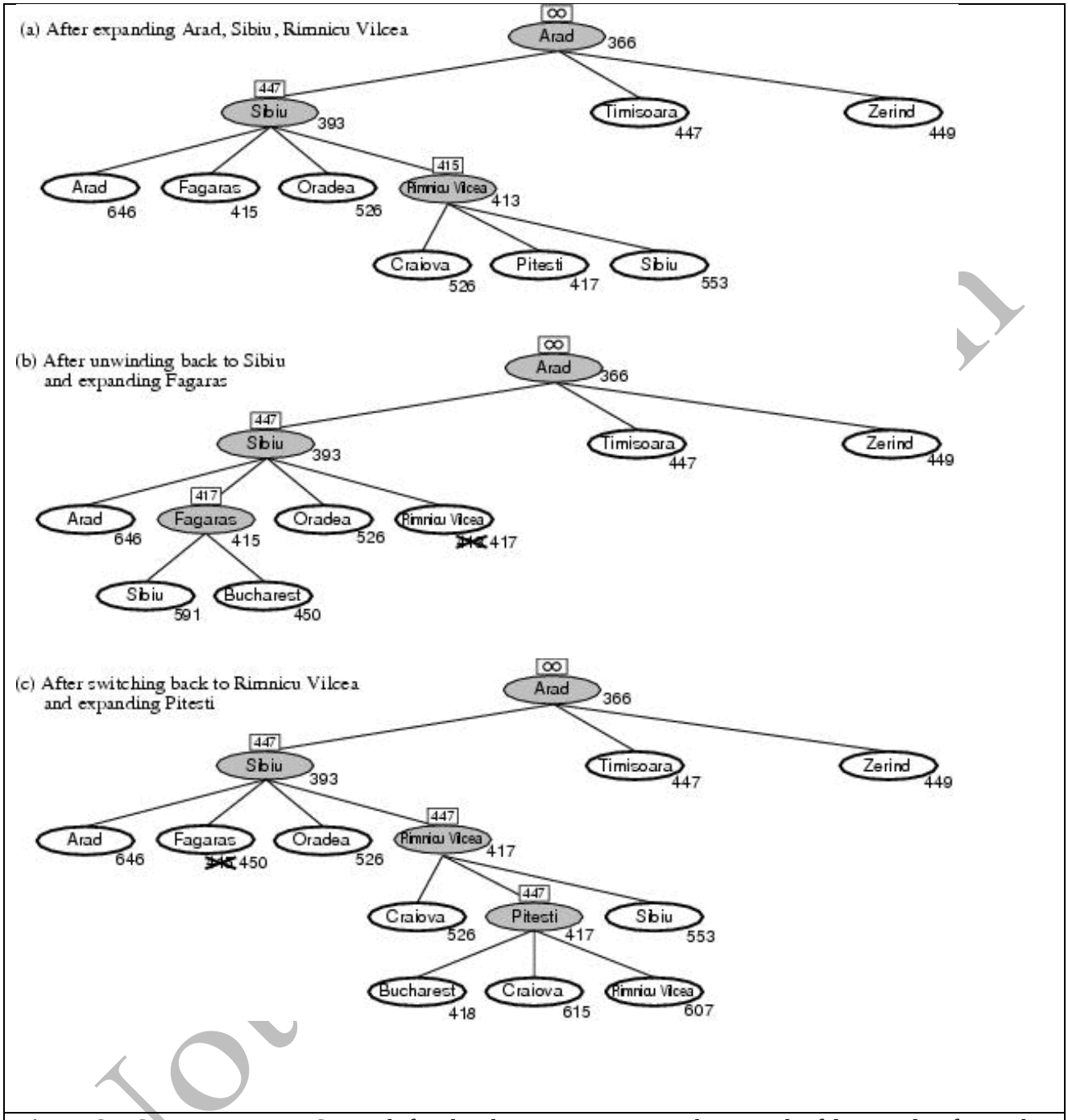



**Figure 2.5** Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras). (b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea;then Fagaras is expanded,revealing a best leaf value of 450.
(c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed upto Fagaras; then Rimni Vicea is expanded. This time because the best alternative path(through Timisoara) costs atleast 447,the expansion continues to Bucharest

RBFS **Evaluation** :

RBFS is a bit more efficient than IDA*

- Still excessive node generation (mind changes)

**Powered By www.technoscriptz.com**

Like A*, optimal if *h(n)* is admissible
Space complexity is *O(bd)*.

- IDA* retains only one single number (the current f-cost limit)

Time complexity difficult to characterize

- Depends on accuracy if h(n) and how often best path changes.

IDA* en RBFS suffer from ***too little*** memory.

## 2.1.2 Heuristic Functions

A **heuristic function** or simply a heuristic is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search

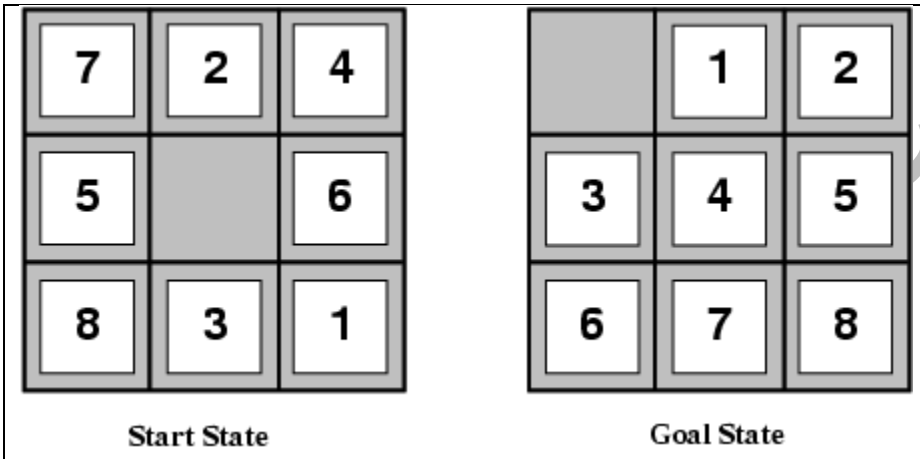



**Figure 2.6** A typical instance of the 8-puzzle.
The solution is 26 steps long.

**The 8-puzzle**

The 8-puzzle is an example of Heuristic search problem. The object of the puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration(Figure 2.6)

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

By keeping track of repeated states,we could cut this down by a factor of about 170,000,because there are only 9!/2 = 181,440 distinct states that are reachable. This is a manageable number ,but the corresponding number for the 15-puzzle is roughly $10^{13}$.

If we want to find the shortest solutions by using $A^*$,we need a heuristic function that never overestimates the number of steps to the goal.

The two commonly used heuristic functions for the 15-puzzle are :

(1) $h_1$ = the number of misplaced tiles.

For figure 2.6 ,all of the eight tiles are out of position,so the start state would have $h_1 = 8$. $h_1$ is an admissible heuristic.

Notesen.com

**(2)** $h_2$ = the sum of the distances of the tiles from their goal positions. This is called **the city block distance** or **Manhattan distance.**

$h_2$ is admissible ,because all any move can do is move one tile one step closer to the goal. Tiles 1 to 8 in start state give a Manhattan distance of

$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18.$

Neither of these overestimates the true solution cost ,which is 26.

**The Effective Branching factor**

One way to characterize the **quality of a heuristic** is the **effective branching factor b*.** If the total number of nodes generated by A* for a particular problem is **N**,and the **solution depth** is **d,**then $b^*$ is the branching factor that a uniform tree of depth d would have to have in order to contain N+1 nodes. Thus,

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

For example,if $A^*$ finds a solution at depth 5 using 52 nodes,then effective branching factor is 1.92. A well designed heuristic would have a value of $b^*$ close to 1,allowing failru large problems to be solved.

To test the heuristic functions $h_1$ and $h_2$,1200 random problems were generated with solution lengths from 2 to 24 and solved them with iterative deepening search and with $A^*$ search using both $h_1$ and $h_2$. Figure 2.7 gives the averaghe number of nodes expanded by each strategy and the effective branching factor.

The results suggest that $h_2$ is better than $h_1$,and is far better than using iterative deepening search. For a solution length of 14,$A^*$ with $h_2$ is 30,000 times more efficient than uninformed iterative deepening search.

| | Search Cost | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | $A^*(h_1)$ | $A^*(h_2)$ | IDS | $A^*(h_1)$ | $A^*(h_2)$ |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | – | 539 | 113 | – | 1.44 | 1.23 |
| 16 | – | 1301 | 211 | – | 1.45 | 1.25 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 20 | – | 7276 | 676 | – | 1.47 | 1.27 |
| 22 | – | 18094 | 1219 | – | 1.48 | 1.28 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

**Figure 2.7** Comparison of search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and $A^*$ Algorithms with $h_1$,and $h_2$. Data are average over 100 instances of the 8-puzzle,for various solution lengths.

**Inventing admissible heuristic functions**

- **Relaxed problems**
  - A problem with fewer restrictions on the actions is called a ***relaxed problem***
  - The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
  - If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then *h1(n)* gives the shortest solution

**Powered By www.technoscriptz.com**

- If the rules are relaxed so that a tile can move to *any adjacent square,* then *h2(n)* gives the shortest solution

## 2.1.3 LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- For example,in the 8-queens problem,what matters is the final configuration of queens,not the order in which they are added.

- In such cases, we can **use local search algorithms.** They operate using a **single current state**(rather than multiple paths) and generally move only to neighbors of that state.
- The important applications of these class of problems are (a) integrated-circuit design,(b)Factory-floor layout,(c) job-shop scheduling,(d)automatic programming,(e)telecommunications network optimization,(f)Vehicle routing,and (g) portfolio management.

**Key advantages of Local Search Algorithms**

(1) They use very little memory – usually a constant amount; and

(2) they can often find reasonable solutions in large or infinite(continuous) state spaces for which systematic algorithms are unsuitable.

**OPTIMIZATION PROBLEMS**

Inaddition to finding goals,local search algorithms are useful for solving pure **optimization problems**,in which the aim is to find the **best state** according to an **objective function**.

**State Space Landscape**

To understand local search,it is better explained using **state space landscape** as shown in figure 2.8.

A landscape has both "**location**" (defined by the state) and "**elevation**"(defined by the value of the heuristic cost function or objective function).

If elevation corresponds to **cost**,then the aim is to find the **lowest valley** – a **global minimum**; if elevation corresponds to an **objective function**,then the aim is to find the **highest peak** – a **global maximum.**

Local search algorithms explore this landscape. A complete local search algorithm always finds a **goal** if one exists; an **optimal** algorithm always finds a **global minimum/maximum**.

Notesengine.com

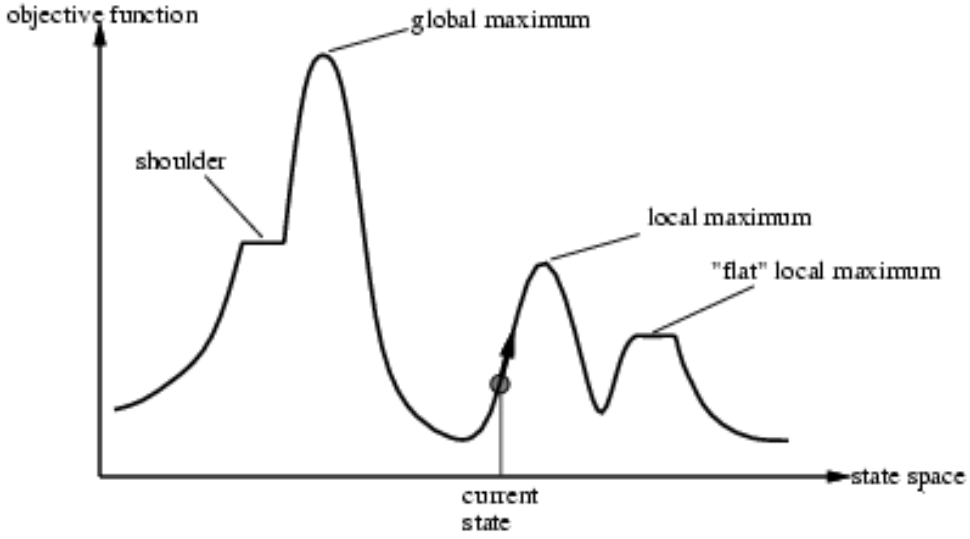



**Figure 2.8** A one dimensional **state space landscape** in which elevation corresponds to the **objective function**. The aim is to find the global maximum. Hill climbing search modifies the current state to try to improve it ,as shown by the arrow. The various topographic features are defined in the text

## Hill-climbing search

The **hill-climbing** search algorithm as shown in figure 2.9, is simply a loop that continually moves in the direction of increasing value – that is,**uphill**. It terminates when it reaches a "**peak**" where no neighbor has a higher value.

```
function HILL-CLIMBING( problem) return a state that is a local maximum
        input: problem, a problem
        local variables: current, a node.
                         neighbor, a node.

        current ← MAKE-NODE(INITIAL-STATE[problem])
        loop do
                neighbor ← a highest valued successor of current
                if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
                current ← neighbor
```

**Figure 2.9** The hill-climbing search algorithm (steepest ascent version),which is the most basic local search technique. At each step the current node is replaced by the best neighbor;the neighbor with the highest VALUE. If the heuristic cost estimate h is used,we could find the neighbor with the lowest h.

Hill-climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next. Greedy algorithms often perform quite well.

## Problems with hill-climbing

Hill-climbing often gets stuck for the following reasons :

- **Local maxima** : a local maximum is a peak that is higher than each of its neighboring states,but lower than the global maximum. Hill-climbing algorithms that reach the vicinity

Notesengine.com

of a local maximum will be drawn upwards towards the peak,but will then be stuck with nowhere else to go

- **Ridges** : A ridge is shown in Figure 2.10. Ridges results in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- **Plateaux** : A plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum,from which no uphill exit exists,or a shoulder,from which it is possible to make progress.

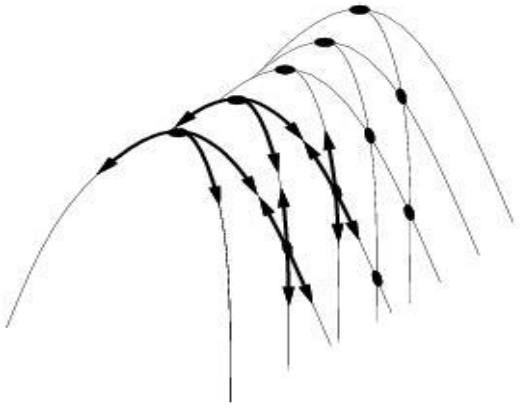


**Figure 2.10** Illustration of why ridges cause difficulties for hill-climbing. The grid of states(dark circles) is superimposed on a ridge rising from left to right,creating a sequence of local maxima that are not directly connected to each other. From each local maximum,all th available options point downhill.

## Hill-climbing variations

- **Stochastic hill-climbing**
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- **First-choice hill-climbing**
  - cfr. stochastic hill climbing by generating successors randomly until a better one is found.
- **Random-restart hill-climbing**
  - Tries to avoid getting stuck in local maxima.

## Simulated annealing search

A hill-climbing algorithm that never makes "downhill" moves towards states with lower value(or higher cost) is guaranteed to be incomplete,because it can stuck on a local maximum.In contrast,a purely random walk –that is,moving to a successor choosen uniformly at random from the set of successors – is complete,but extremely inefficient.

Simulated annealing is an algorithm that combines hill-climbing with a random walk in someway that yields both efficiency and completeness.

Figure 2.11 shows simulated annealing algorithm. It is quite similar to hill climbing. Instead of picking the best move,however,it picks the random move. If the move improves the situation,it is always accepted. Otherwise,the algorithm accepts the move with some probability less than 1. The probability decreases exponentially with the "badness" of the move – the amount $\triangle$ E by which the evaluation is worsened.

Notesengine.com

Simulated annealing was first used extensively to solve VLSI layout problems in the early 1980s. It has been applied widely to factory scheduling and other large-scale optimization tasks.

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

**Figure 2.11** The simulated annealing search algorithm,a version of stochastic hill climbing where some downhill moves are allowed.

## Genetic algorithms

A Genetic algorithm(or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states,rather than by modifying a single state.
Like beam search,Gas begin with a set of k randomly generated states,called the population. Each state,or individual,is represented as a string over a finite alphabet – most commonly,a string of 0s and 1s. For example,an 8 8-quuens state must specify the positions of 8 queens,each in acolumn of 8 squares,and so requires 8 x $\log_2$ 8 = 24 bits.

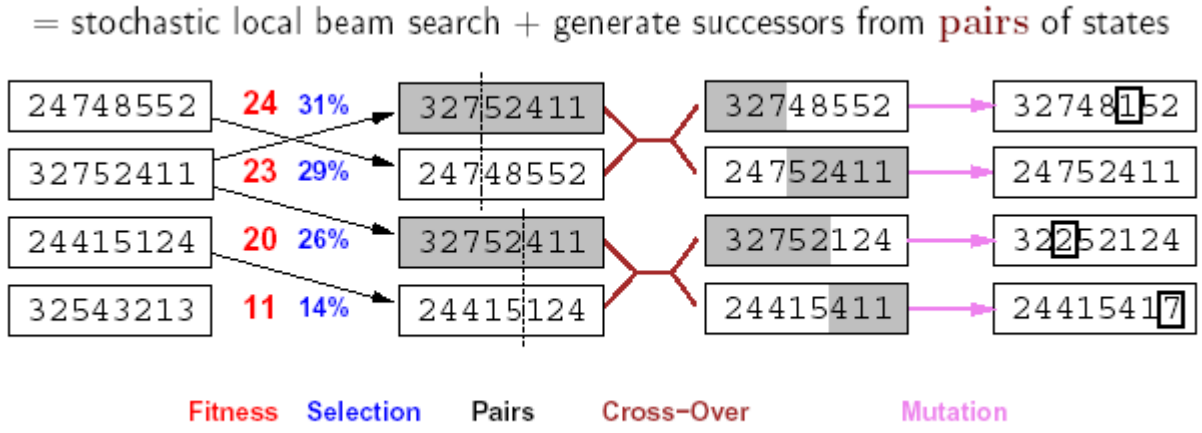



**Figure 2.12** The genetic algorithm. The initial population in (a) is ranked by the fitness function in (b),resulting in pairs for mating in (c). They produce offspring in (d),which are subjected to

**Powered By www.technoscriptz.com**

mutation in (e).

Figure 2.12 shows a population of four 8-digit strings representing 8-queen states. The production of the next generation of states is shown in Figure 2.12(b) to (e).
In (b) each state is rated by the evaluation function or the **fitness function**.
In (c),a random choice of two pairs is selected for reproduction,in accordance with the probabilities in (b).
Figure 2.13 describes the algorithm that implements all these steps.

```
function GENETIC_ALGORITHM( population, FITNESS-FN) return an individual
    input: population, a set of individuals
           FITNESS-FN, a function which determines the quality of the individual
    repeat
        new_population ← empty set
        loop for i from 1 to SIZE(population) do
            x ← RANDOM_SELECTION(population, FITNESS_FN)
    y ← RANDOM_SELECTION(population, FITNESS_FN)
            child ← REPRODUCE(x,y)
            if (small random probability) then child ← MUTATE(child )
            add child to new_population
        population ← new_population
    until some individual is fit enough or enough time has elapsed
    return the best individual
```

**Figure 2.13** A genetic algorithm. The algorithm is same as the one diagrammed in Figure 2.12,with one variation:each mating of two parents produces only one offspring,not two.

## 2.1.4 LOCAL SEARCH IN CONTINUOUS SPACES

- We have considered algorithms that work only in discrete environments, but real-world environment are continuous
- Local search amounts to maximizing a continuous objective function in a multi-dimensional vector space.
- This is hard to do in general.
- Can immediately retreat
    - Discretize the space near each state
    - Apply a discrete local search strategy (e.g., stochastic hill climbing, simulated annealing)
- Often resists a closed-form solution
    - Fake up an empirical gradient
    - Amounts to greedy hill climbing in discretized state space
- Can employ Newton-Raphson Method to find maxima
- Continuous problems have similar problems: plateaus, ridges, local maxima, etc.

## 2.1.5 Online Search Agents and Unknown Environments

**Online search problems**

- Offline Search (all algorithms so far)

Notesengine.com

- Compute complete solution, ignoring environment Carry out action sequence

- Online Search
  - Interleave computation and action
  - Compute—Act—Observe—Compute—·
- Online search good
  - For dynamic, semi-dynamic, stochastic domains
  - Whenever offline search would yield exponentially many contingencies
- Online search necessary for exploration problem
  - States and actions unknown to agent
  - Agent uses actions as experiments to determine what to do

Examples

Robot exploring unknown building
Classical hero escaping a labyrinth

- Assume agent knows
  - Actions available in state $s$
    Step-cost function $c(s,a,s')$
    State s is a goal state
- When it has visited a state s previously Admissible heuristic function h(s )
- Note that agent doesn't know outcome state ($s'$) for a given action (a) until it tries the action (and all actions from a state s )
- Competitive ratio compares actual cost with cost agent would follow if it knew the search space
- No agent can avoid dead ends in all state spaces
  - Robotics examples: Staircase, ramp, cliff, terrain
- Assume state space is safely explorable—some goal state is always reachable

**Online Search Agents**

- Interleaving planning and acting hamstrings offline search
  - A* expands arbitrary nodes without waiting for outcome of action Online algorithm can expand only the node it physically occupies Best to explore nodes in physically local order
  - Suggests using depth-first search
  - Next node always a child of the current
- When all actions have been tried, can't just drop state
  Agent must physically backtrack
- Online Depth-First Search
  - May have arbitrarily bad competitive ratio (wandering past goal) Okay for exploration; bad for minimizing path cost
- Online Iterative-Deepening Search
  - Competitive ratio stays small for state space a uniform tree

**Online Local Search**

- Hill Climbing Search
  - Also has physical locality in node expansions
    Is, in fact, already an online search algorithm
  - Local maxima problematic: can't randomly transport agent to new state in

Notesengine.com

effort to escape local maximum

- Random Walk as alternative
  - Select action at random from current state
  - Will eventually find a goal node in a finite space
  - Can be very slow, esp. if "backward" steps as common as "forward"
- Hill Climbing with Memory instead of randomness
  - Store "current best estimate" of cost to goal at each visited state Starting estimate is just h(s )
  - Augment estimate based on experience in the state space Tends to "flatten out" local minima, allowing progress Employ optimism under uncertainty
  - Untried actions assumed to have least-possible cost Encourage exploration of untried paths

**Learning in Online Search**

- Rampant ignorance a ripe opportunity for learning Agent learns a "map" of the environment
- Outcome of each action in each state
- Local search agents improve evaluation function accuracy
- Update estimate of value at each visited state
- Would like to infer higher-level domain model
- Example: "Up" in maze search increases y -coordinate Requires
- Formal way to represent and manipulate such general rules (so far, have hidden rules within the successor function)
- Algorithms that can construct general rules based on observations of the effect of actions

# 2.2 CONSTRAINT SATISFACTION PROBLEMS(CSP)

A **Constraint Satisfaction Problem**(or CSP) is defined by a set of **variables ,** $X_1, X_2, \ldots X_n$, and a set of constraints $C_1, C_2, \ldots, C_m$. Each variable $X_i$ has a nonempty **domain** D,of possible **values**. Each constraint $C_i$ involves some subset of variables and specifies the allowable combinations of values for that subset.

A **State** of the problem is defined by an **assignment** of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$. An assignment that does not violate any constraints is called a **consistent** or **legal assignment.** A complete assignment is one in which every variable is mentioned,and a **solution** to a CSP is a complete assignment that satisfies all the constraints.

Some CSPs also require a solution that maximizes an **objective function**.

**Example for Constraint Satisfaction Problem :**

Figure 2.15 shows the map of Australia showing each of its states and territories. We are given the task of coloring each region either red,green,or blue in such a way that the neighboring regions have the same color. To formulate this as CSP ,we define the variable to be the regions :WA,NT,Q,NSW,V,SA, and T. The domain of each variable is the set {red,green,blue}.The constraints require neighboring regions to have distinct colors;for example,the allowable combinations for WA and NT are the pairs {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}.

The constraint can also be represented more succinctly as the inequality WA not = NT,provided the constraint satisfaction algorithm has some way to evaluate such expressions.) There are many possible solutions such as

{ WA = red, NT = green,Q = red, NSW = green, V = red ,SA = blue,T = red}.

**Powered By www.technoscriptz.com**

Notesengine.com

It is helpful to visualize a CSP as a constraint graph,as shown in Figure 2.15(b). The nodes of the graph corresponds to variables of the problem and the arcs correspond to constraints.

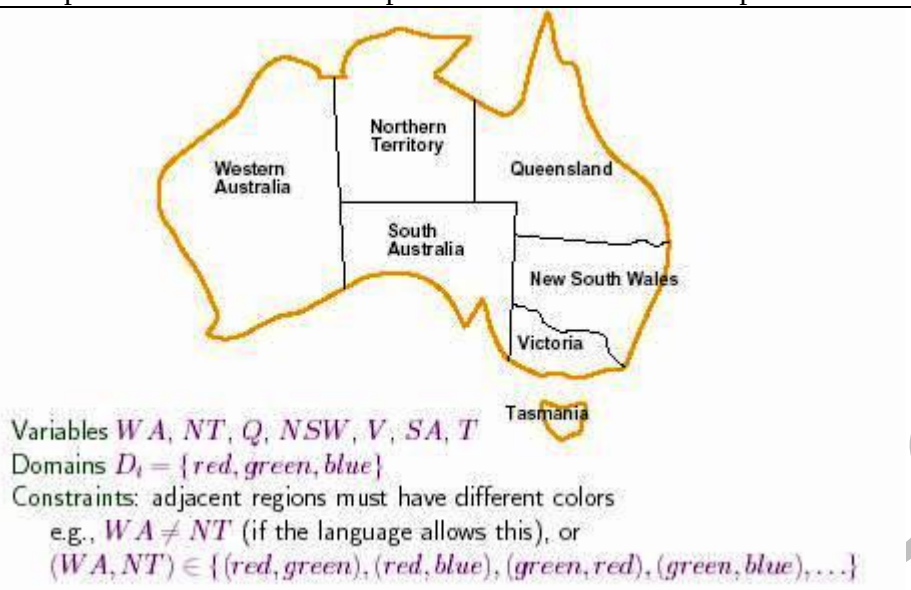



**Figure 2.15** (a) Principle states and territories of Australia. Coloring this map can be viewed as aconstraint satisfaction problem. The goal is to assign colors to each region so that no neighboring regions have the same color.

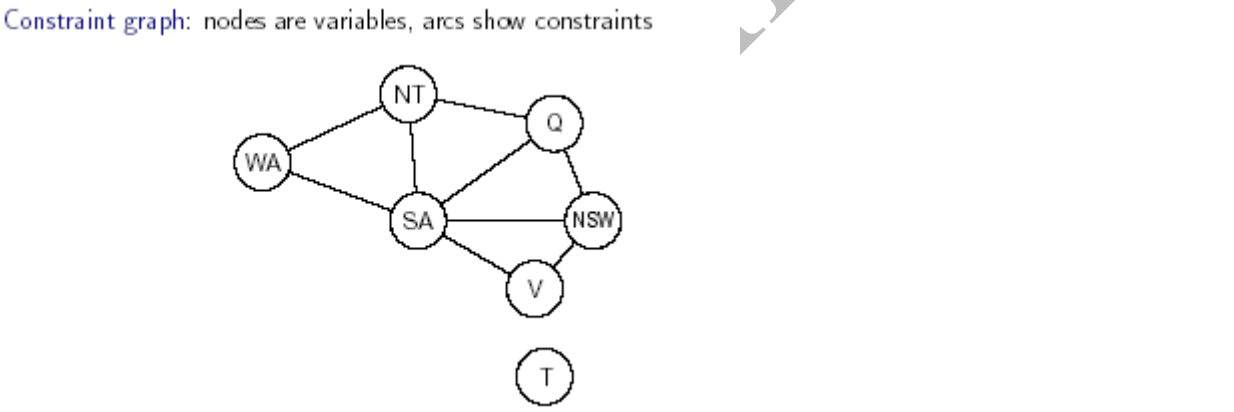



**Figure 2.15** (b) The map coloring problem represented as a constraint graph.

CSP can be viewed as a standard search problem as follows :

- **Initial state** : the empty assignment {},in which all variables are unassigned.
- **Successor function** : a value can be assigned to any unassigned variable,provided that it does not conflict with previously assigned variables.
- **Goal test** : the current assignment is complete.
- **Path cost** : a constant cost(E.g.,1) for every step.

Every solution must be a complete assignment and therefore appears at depth n if there are n variables.

Depth first search algorithms are popular for CSPs

**Varieties of CSPs**

**(i) Discrete variables**

**Finite domains**

The simplest kind of CSP involves variables that are **discrete** and have **finite domains.** Map coloring problems are of this kind. The 8-queens problem can also be viewed as finite-domain

CSP,where the variables $Q_1$,$Q_2$,…..$Q_8$ are the positions each queen in columns 1,….8 and each variable has the domain {1,2,3,4,5,6,7,8}. If the maximum domain size of any variable in a CSP is d,then the number of possible complete assignments is $O(d^n)$ – that is,exponential in the number of variables. Finite domain CSPs include **Boolean CSPs**,whose variables can be either *true* or *false*.

**Infinite domains**

Discrete variables can also have **infinite domains** – for example,the set of integers or the set of strings. With infinite domains,it is no longer possible to describe constraints by enumerating all allowed combination of values. Instead a constraint language of algebric inequalities such as $Startjob_1 + 5 <= Startjob_3$.

**(ii) CSPs with continuous domains**

CSPs with continuous domains are very common in real world. For example ,in operation research field,the scheduling of experiments on the Hubble Telescope requires very precise timing of observations; the start and finish of each observation and maneuver are continuous-valued variables that must obey a variety of astronomical,precedence and power constraints. The best known category of continuous-domain CSPs is that of **linear programming** problems,where the constraints must be linear inequalities forming a *convex* region. Linear programming problems can be solved in time polynomial in the number of variables.

**Varieties of constraints :**

**(i) unary constraints** involve a single variable.

Example : SA # green

(ii) Binary constraints involve paris of variables.

Example : SA # WA

(iii) Higher order constraints involve 3 or more variables.

Example : cryptarithmetic puzzles.

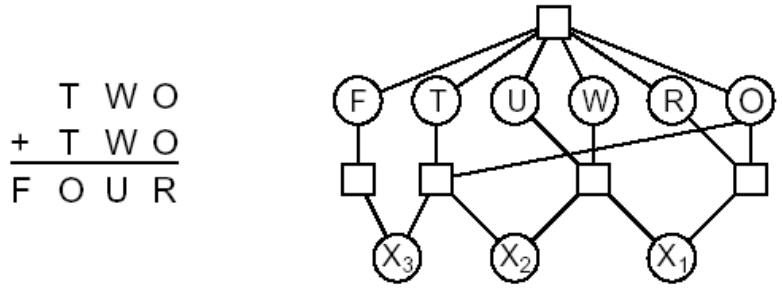



**Figure 2.16** (a) Cryptarithmetic problem. Each letter stands for a distinct digit;the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct,with the added restriction that no leading zeros are allowed. (b) The constraint hypergraph for the cryptarithmetic problem,showint the *Alldiff* constraint as well as the column addition constraints. Each constraint is a square box connected to the variables it contains.

**Powered By www.technoscriptz.com**

## 2.2.2 Backtracking Search for CSPs

The term **backtracking search** is used for depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The algorithm is shown in figure 2.17.

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```

**Figure 2.17** A simple backtracking algorithm for constraint satisfaction problem. The algorithm is modeled on the recursive depth-first search

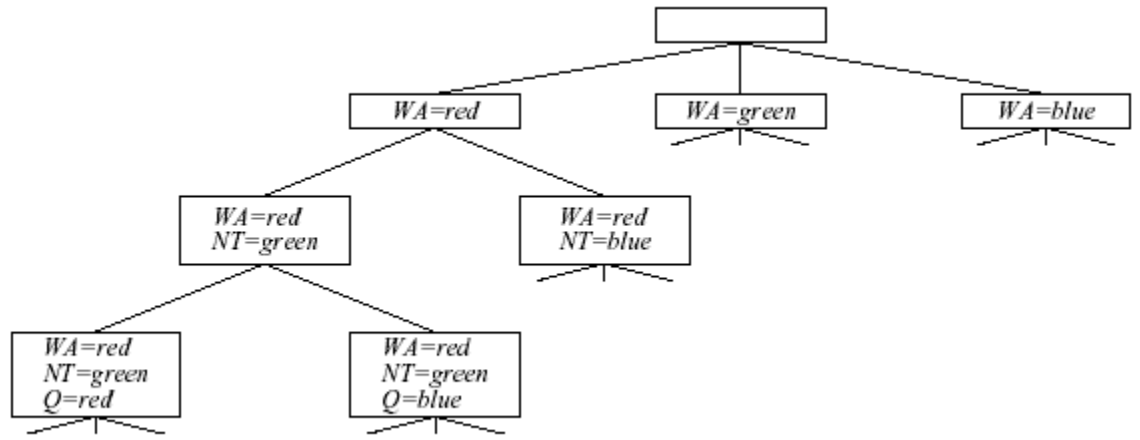



**Figure 2.17(b)** Part of search tree generated by simple backtracking for the map coloring problem.

**Propagating information through constraints**

So far our search algorithm considers the constraints on a variable only at the time that the variable is chosen by SELECT-UNASSIGNED-VARIABLE. But by looking at some of the constraints earlier in the search, or even before the search has started, we can drastically reduce the search space.

**Forward checking**

One way to make better use of constraints during search is called **forward checking**. Whenever a variable X is assigned, the forward checking process looks at each unassigned variable Y that is connected to X by a constraint and deletes from Y 's domain any value that is inconsistent with the value chosen for X. Figure 5.6 shows the progress of a map-coloring search with forward checking.

**Powered By www.technoscriptz.com**

| | *WA* | *NT* | *Q* | *NSW* | *V* | *SA* | *T* |
|---|---|---|---|---|---|---|---|
| Initial domains | R G B | R G B | R G B | R G B | R G B | R G B | R G B |
| After *WA*=*red* | (R) | G B | R G B | R G B | R G B | G B | R G B |
| After *Q*=*green* | (R) | B | (G) | R B | R G B | B | R G B |
| After *V*=*blue* | (R) | B | (G) | R | (B) | | R G B |

**Figure 5.6** The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables $NT$ and $SA$. After $Q = green$, *green* is deleted from the domains of $NT$, $SA$, and $NSW$. After $V = blue$, *blue* is deleted from the domains of $NSW$ and $SA$, leaving $SA$ with no legal values.

## Constraint propagation

Although forward checking detects many inconsistencies, it does not detect all of them.
**Constraint propagation** is the general term for propagating the implications of a constraint on one variable onto other variables.

### Arc Consistency

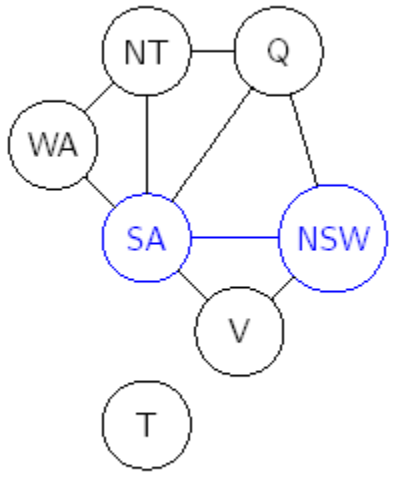



Figure: Australian Territories

- One method of constraint propagation is to enforce **arc consistency**
  - Stronger than forward checking
  - Fast
- *Arc* refers to a *directed* arc in the constraint graph
- Consider two nodes in the constraint graph (e.g., *SA* and *NSW*)
  - An arc is **consistent** if
  - For every value $x$ of *SA*
  - There is some value $y$ of *NSW* that is consistent with $x$
- Examine arcs for consistency in *both* directions

### k-Consistency

- Can define stronger forms of consistency

*k*-Consistency

A CSP is *k*-**consistent** if, for any consistent assignment to $k-1$ variables, there is a consistent assignment for the *k*-th variable

- **1-consistency (node consistency)**
  - Each variable by itself is consistent (has a non-empty domain)
- **2-consistency (arc consistency)**
- **3-consistency (path consistency)**
  - Any pair of adjacent variables can be extended to a third

Powered By www.technoscriptz.com

## Local Search for CSPs

- Local search algorithms good for many CSPs
- Use complete-state formulation
  - Value assigned to every variable
  - Successor function changes one value at a time
- Have already seen this
  - Hill climbing for 8-queens problem (AIMA § 4.3)
- Choose values using **min-conflicts** heuristic
  - Value that results in the minimum number of conflicts with other variables

## 2.2.3 The Structure of Problems
## Problem Structure

- Consider ways in which the structure of the problem's constraint graph can help find solutions
- Real-world problems require decomposition into subproblems

## Independent Subproblems

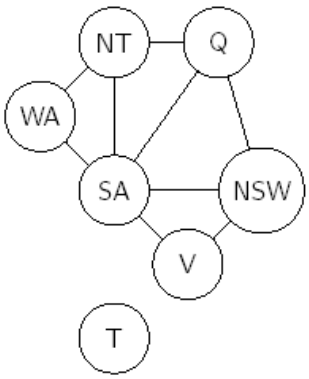



Figure: Australian Territories

- $T$ is not connected
- Coloring $T$ and coloring remaining nodes are **independent subproblems**
- *Any* solution for $T$ combined with *any* solution for remaining nodes solves the problem
- Independent subproblems correspond to **connected components** of the constraint graph
- Sadly, such problems are rare

## Tree-Structured CSPs

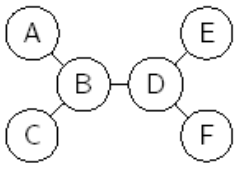



Figure: Tree-Structured CSP

- In most cases, CSPs are connected
- A simple case is when the constraint graph is a **tree**
- Can be solved in time linear in the number of variables
  - Order variables so that each parent precedes its children
  - Working "backward," apply arc consistency between child and parent
  - Working "forward," assign values consistent with parent

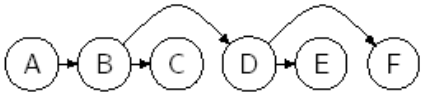



Figure: Linear ordering

Powered By www.technoscriptz.com

# 2.4 ADVERSARIAL SEARCH

Competetive environments,in which the agent's goals are in conflict,give rise to **adversarial search** problems – often known as **games.**

### 2.4.1 Games

Mathematical **Game Theory**,a branch of economics,views any **multiagent environment** as a **game** provided that the impact of each agent on the other is "significant",regardless of whether the agents are cooperative or competitive. In,**AI**,"games" are deterministic,turn-taking,two-player,zero-sum games of perfect information. This means deterministic,fully observable environments in which there are two agents whose actions must alternate and in which the **utility values** at the end of the game are always equal and opposite. For example,if one player wins the game of chess(+1),the other player necessarily loses(-1). It is this opposition between the agents' utility functions that makes the situation **adversarial.**

**Formal Definition of Game**

We will consider games with two players,whom we will call **MAX** and **MIN**. MAX moves first,and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A **game** can be formally defined as a **search problem** with the following components :

- The **initial state**,which includes the board position and identifies the player to move.
- A **successor function**,which returns a list of (*move*,*state*) pairs,each indicating a legal move and the resulting state.
- A **terminal test**,which describes when the game is over. States where the game has ended are called **terminal states**.
- A **utility function** (also called an objective function or payoff function),which give a numeric value for the terminal states. In chess,the outcome is a win,loss,or draw,with values +1,-1,or 0. he payoffs in backgammon range from +192 to -192.

## Game Tree

The **initial state** and **legal moves** for each side define the **game tree** for the game. Figure 2.18 shows the part of the game tree for tic-tac-toe (noughts and crosses). From the initial state,MAX has nine possible moves. Play alternates between MAX's placing an X and MIN's placing a 0 until we reach leaf nodes corresponding to the terminal states such that one player has three in a row or all the squares are filled. He number on each leaf node indicates the utility value of the terminal state from the point of view of MAX;high values are assumed to be good for MAX and bad for MIN. It is the MAX's job to use the search tree(particularly the utility of terminal states) to determine the best move.

Notesengine.com

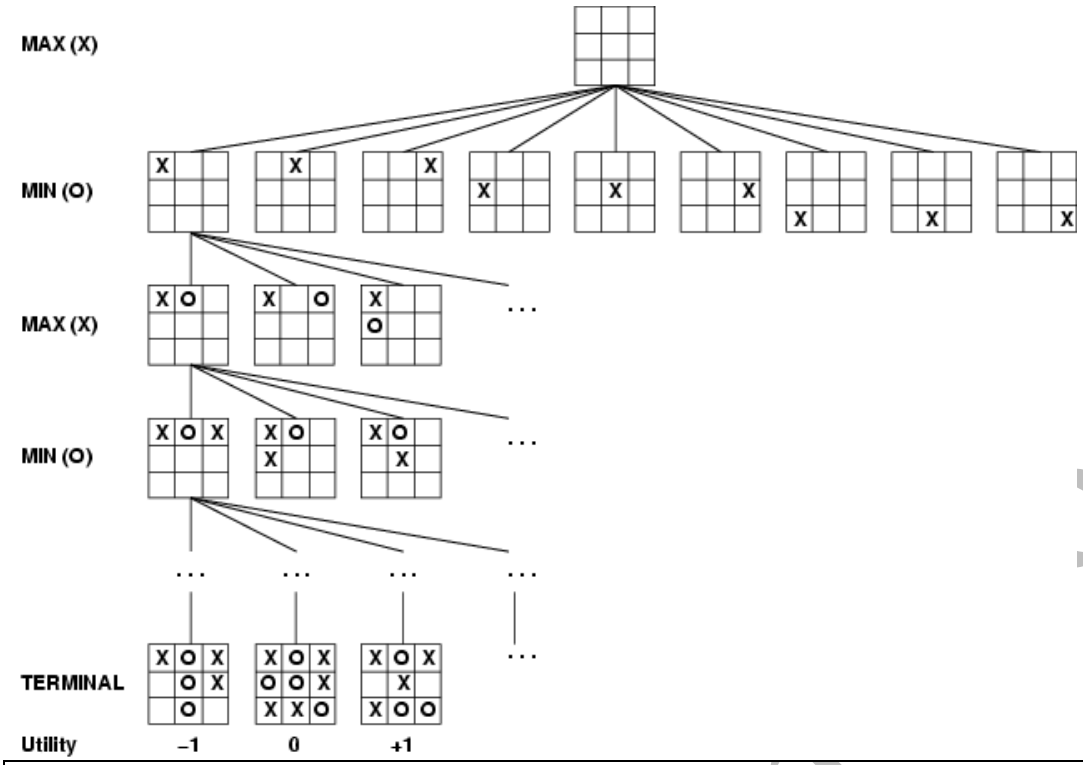
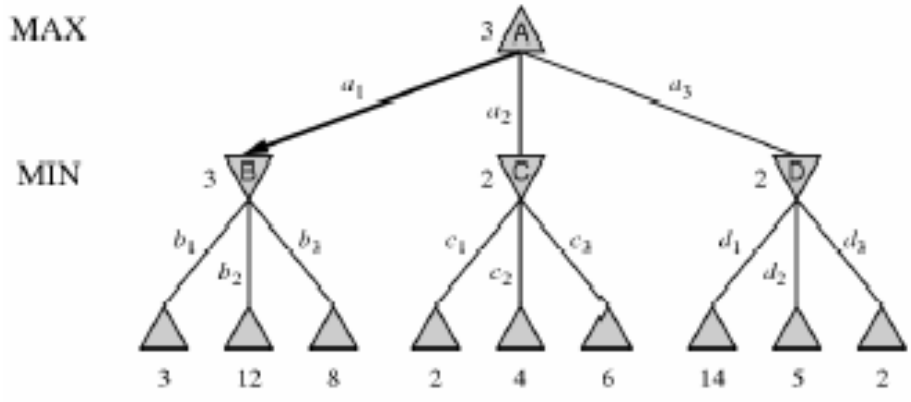



**Figure 2.18** A partial search tree . The top node is the initial state,and MAX move first,placing an X in an empty square.

### 2.4.2 Optimal Decisions in Games

In normal search problem,the **optimal solution** would be a sequence of move leading to a **goal state** – a terminal state that is a win. In a game,on the other hand,MIN has something to say about it,MAX therefore must find a contingent **strategy**,which specifies MAX's move in the **initial state**,then MAX's moves in the states resulting from every possible response by MIN,then MAX's moves in the states resulting from every possible response by MIN those moves,and so on. An **optimal strategy** leads to outcomes at least as good as any other strategy when one is playing an infallible opponent.




Powered By www.technoscriptz.com

**Figure 2.19** A two-ply game tree. The △ nodes are "MAX nodes",in which it is AMX's turn to move,and the ▽ nodes are "MIN nodes". The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is $a_1$,because it leads to the successor with the highest minimax value,and MIN's best reply is $b_1$,because it leads to the successor with the lowest minimax value.

## Minimax Search: Algorithm

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value                    For MAX Node
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s))
  return v
```

```
function MIN-VALUE(state) returns a utility value                    For MIN Node
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s))
  return v
```

**Figure 2.20** An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move,that is,the move that leads to the outcome with the best utility,under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree,all the way to the leaves,to determine the backed-up value of a state.

**The minimax Algorithm**

The minimax algorithm(Figure 2.20) computes the minimax decision from the current state. It uses a simple recursive computation of the minimax values of each successor state,directly implementing the defining equations. The recursion proceeds all the way down to the leaves of the tree ,and then the minimax values are **backed up** through the tree as the recursion unwinds**.** For example in Figure 2.19,the algorithm first recourses down to the three bottom left nodes,and uses the utitliy function on them to discover that their values are 3,12,and 8 respectively. Then it takes the minimum of these values,3,and returns it as the backed-up value of node B. A similar process gives the backed up values of 2 for C and 2 for D. Finally,we take the maximum of 3,2,and 2 to get the backed-up value of 3 at the root node. The minimax algorithm performs a complete depth-first exploration of the game tree. If the maximum depth of the tree is *m*,and there are b legal moves at each point,then the time

Powered By www.technoscriptz.com

complexity of the minimax algorithm is $O(b^m)$. The space complexity is O(bm) for an algorithm that generates successors at once.

### 2.4.3 Alpha-Beta Pruning

The problem with minimax search is that the number of game states it has to examine is **exponential** in the number of moves. Unfortunately,we can't eliminate the exponent,but we can effectively cut it in half. By performing **pruning,**we can eliminate large part of the tree from consideration. We can apply the technique known as **alpha beta pruning** ,when applied to a minimax tree ,it returns the same move as **minimax** would,but **prunes away** branches that cannot possibly influence the final decision.

Alpha Beta pruning gets its name from the following two parameters that describe bounds on the backed-up values that appear anywhere along the path:

- α : the value of the best(i.e.,highest-value) choice we have found so far at any choice point along the path of MAX.
- β: the value of best (i.e., lowest-value) choice we have found so far at any choice point along the path of MIN.

Alpha Beta search updates the values of α and β as it goes along and prunes the remaining branches at anode(i.e.,terminates the recursive call) as soon as the value of the current node is known to be worse than the current α and β value for MAX and MIN,respectively. The complete algorithm is given in Figure 2.21.

The effectiveness of alpha-beta pruning is highly dependent on the order in which the successors are examined. It might be worthwhile to try to examine first the successors that are likely to be the best. In such case,it turns out that alpha-beta needs to examine only $O(b^{d/2})$ nodes to pick the best move,instead of $O(b^d)$ for minimax. This means that the effective branching factor becomes sqrt(b) instead of b – for chess,6 instead of 35. Put anotherway alpha-beta cab look ahead roughly twice as far as minimax in the same amount of time.

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s, α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

Powered By www.technoscriptz.com

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

**Figure 2.21** The alpha beta search algorithm. These routines are the same as the minimax routines in figure 2.20,except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β

## 2.4.4 Imperfect ,Real-time Decisions

The minimax algorithm generates the entire game search space,whereas the alpha-beta algorithm allows us to prune large parts of it. However,alpha-beta still has to search all the way to terminal states for atleast a portion of search space. Shannon's 1950 paper,Programming a computer for playing chess,proposed that programs should **cut off** the search earlier and apply a heuristic **evaluation function** to states in the search,effectively turning nonterminal nodes into terminal leaves. The basic idea is to alter minimax or alpha-beta in two ways :

(1) The utility function is replaced by a heuristic evaluation function EVAL,which gives an estimate of the position's utility,and

(2) the terminal test is replaced by a **cutoff test** that decides when to apply EVAL.

## 2.4.5 Games that include Element of Chance

**Evaluation functions**

An evaluation function returns an estimate of the expected utility of the game from a given position,just as the heuristic function return an estimate of the distance to the goal.

**Games of imperfect information**

- Minimax and alpha-beta pruning require too much leaf-node evaluations.
  May be impractical within a reasonable amount of time.
- SHANNON (1950):
  - Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)
  - Apply heuristic evaluation function EVAL (replacing utility function of alpha-beta)

**Cutting off search**

Change**:**

- **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

  into
- **if** CUTOFF-TEST(*state,depth*) **then return** EVAL(*state*)

Notesengine.com

Introduces a fixed-depth limit *depth*

- Is selected so that the amount of time will not exceed what the rules of the game allow.

When cuttoff occurs, the evaluation is performed.

**Heuristic EVAL**

Idea: produce an estimate of the expected utility of the game from a given position.

Performance depends on quality of EVAL.

Requirements:

- EVAL should order terminal-nodes in the same way as UTILITY.
- Computation may not take too long.
- For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.

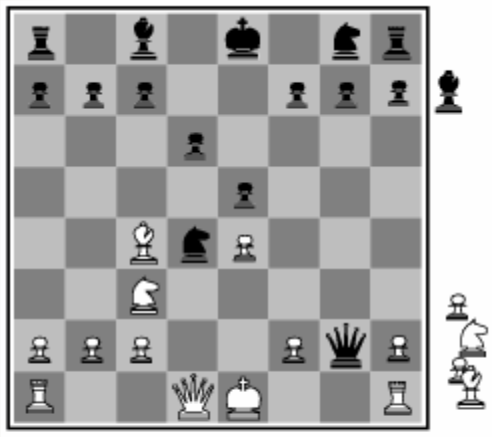
Only useful for quiescent (no wild swings in value in near future) states

**Weighted Linear Function**

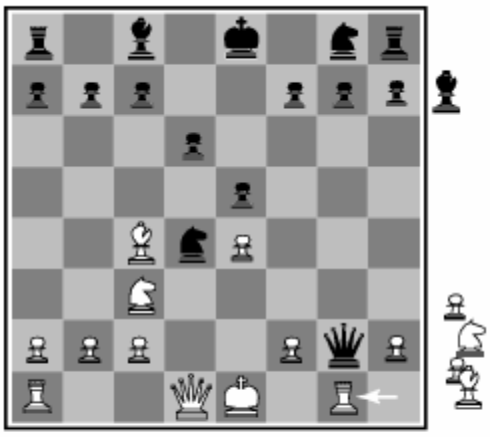
The introductory chess books give an approximate material value for each piece : each pawn is worth 1,a knight or bishop is worth 3,a rook 3,and the queen 9. These feature values are then added up toobtain the evaluation of the position. Mathematically,these kind of evaluation fuction is called weighted linear function,and it can be expressed as :

$$Eval(s) = w1\ f1(s) + w2\ f2(s) + \ldots + wn\ fn(s)$$

- e.g., w1 = 9 with
  f1(s) = (number of white queens) – (number of black queens), etc.



(a) White to move



(b) White to move

(a) Black has an advantage of a knight and two pawns and will win the game.
(b) Black will lose after white captures the queen.

## Games that include chance

Powered By www.technoscriptz.com

In real life,there are many unpredictable external events that put us into unforeseen situations. Many games mirror this unpredictability by including a random element,such as throwing a dice. **Backgammon** is a typical game that combines luck and skill. Dice are rolled at the beginning of player's turn to determine the legal moves. The backgammon position of Figure 2.23,for example,white has rolled a 6-5,and has four possible moves.

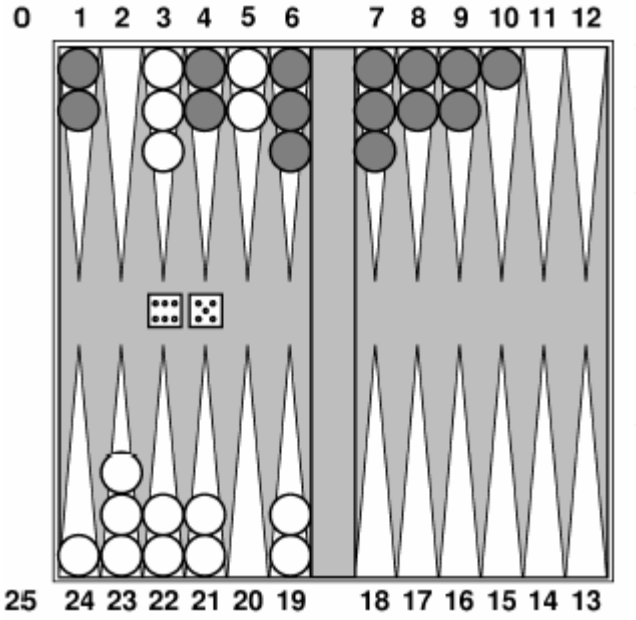



**Figure 2.23** A typical backgammon position. The goal of the game is to move all one's pieces off the board. White moves clockwise toward 25,and black moves counterclockwise toward 0. A piece can move to any position unless there are multiple opponent pieces there; if there is one opponent ,it is captured and must start over. In the position shown,white has rolled 6-5 and must choose among four legal moves (5-10,5-11),(5-11,19-24),(5-10,10-16),and (5-11,11-16)

- White moves clockwise toward 25
- Black moves counterclockwise toward 0
- A piece can move to any position unless there are multiple opponent pieces there; if there is one opponent, it is captured and must start over.
- White has rolled 6-5 and must choose among four legal moves: (5-10, 5-11), (5-11, 19-24) (5-10, 10-16), and (5-11, 11-16)

---

Notesengine.com

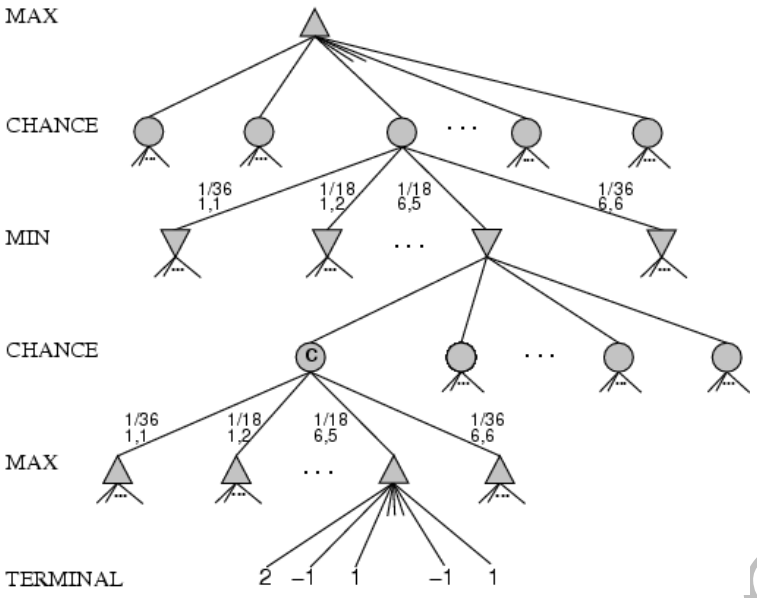



**Figure 2-24** Schematic game tree for a backgammon position.

**Expected minimax value**

EXPECTED-MINIMAX-VALUE(*n*)=

UTILITY(*n*) If *n* is a terminal

MINIMAX-VALUE(*s*) max*s* ∈ *successors(n)* If *n* is a max node

MINIMAX-VALUE(*s*) min*s* ∈ *successors(n)* If *n* is a max node

EXPECTEDMINIMAX(*s*) $\sum s \in successors(n)\ P(s)$ . If *n* is a chance node

These equations can be backed-up recursively all the way to the root of the game tree.

---

Notesengine.com

# UNIT-III Question and Answers

**(1) How Knowledge is represented?**

A variety of ways of knowledge(facts) have been exploited in AI programs.
Facts : truths in some relevant world. These are things we want to represent.

**(2) What is propositional logic?**

It is a way of representing knowledge.
In logic and mathematics, a **propositional calculus** or **logic** is a formal system in which formulae representing *propositions* can be formed by combining atomic propositions using *logical connectives*
Sentences considered in propositional logic are not arbitrary sentences but are the ones that are either true or false, but not both. This kind of sentences are called **propositions**.

**Example**

Some facts in propositional logic:

It is raning. - RAINING
It is sunny - SUNNY
It is windy - WINDY

If it is raining ,then it is not sunny - RAINING -> ¬ SUNNY

**(3) What are the elements of propositional logic?**

Simple sentences which are true or false are basic propositions. Larger and more complex sentences are constructed from basic propositions by combining them with **connectives**. Thus **propositions** and **connectives** are the basic elements of propositional logic. Though there are many connectives, we are going to use the following **five basic connectives** here:

NOT, AND, OR, IF_THEN (or IMPLY), IF_AND_ONLY_IF.

They are also denoted by the symbols:

¬, ∧, ∨, →, ↔, respectively.

**(4) What is inference?**

Inference is deriving new sentences from old.

**(5) What are modus ponens?**

There are standard patterns of inference that can be applied to derive chains of conclusions that lead to the desired goal. These patterns of inference are called **inference rules.** The best-known rule is called **Modus Ponens** and is written as follows:

Notesengine.com

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

The notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and $\alpha$ are given, then the sentence $\beta$ can be inferred. For example, if $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ and $(WumpusAhead \wedge WumpusAlive)$ are given, then *Shoot* can be inferred.

Another useful inference rule is **And-Elimination**, which says that, from a conjunction, any of the conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha}$$

For example, from $(WumpusAhead \wedge WumpusAlive)$, *WumpusAlive* can be inferred.

**(6) What is entailment?**

Propositions tell about the notion of truth and it can be applied to logical reasoning. We can have logical entailment between sentences. This is known as entailment where a sentence follows logically from another sentence. In mathematical notation we write :

$$\alpha \models \beta$$

**(7) What are knowledge based agents?**

The central component of a knowledge-based agent is its knowledge base, or KB. Informally, a knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.

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

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

**Figure 7.1** A generic knowledge-based agent.

Figure 7.1 shows the outline of a knowledge-based agent program. Like all our agents, it takes a percept as input and returns an action. The agent maintains a knowledge base, KB, which may initially contain some **background knowledge.** Each time the agent program is called, it does three things. First, it TELLS the knowledge base what it perceives. Second, it ASKS the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.

(8) **Explain in detail the connectives used in propositional logic.**

The **syntax** of propositional logic defines the allowable sentences. The **atomic sentences-** the indivisible syntactic elements-consist of a single **proposition symbol.** Each such symbol

Powered By www.technoscriptz.com

stands for a proposition that can be true or false. We will use uppercase names for symbols: P, Q, R, and so on.

**Complex sentences** are constructed from simpler sentences using **logical connectives.** There are five connectives in common use:

¬ (not). A sentence such as $\neg W_{1,3}$ is called the **negation** of $W_{1,3}$. **A literal** is either an atomic sentence (a **positive literal)** or a negated atomic sentence (a **negative literal).**

∧ (and). A sentence whose main connective is ∧, such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction;** its parts are the **conjuncts.** (The ∧ looks like an "A" for "And.")

∨ (or). A sentence using ∨, such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction** of the **disjuncts** $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$. (Historically, the ∨ comes from the Latin "vel," which means "or." For most people, it is easier to remember as an upside-down ∧.)

⇒ (implies). A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as ⊃ or →.

⇔ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional.**

Figure 7.7 gives a formal grammar of propositional logic;

| | | |
|---|---|---|
| *Sentence* | → | *AtomicSentence* \| *ComplexSentence* |
| *AtomicSentence* | → | **True** \| **False** \| Symbol |
| *Symbol* | → | *P* \| **Q** \| **R** \| ... |
| *ComplexSentence* | → | ¬ *Sentence* |
| | \| | ( *Sentence* ∧ *Sentence* ) |
| | \| | ( *Sentence* ∨ *Sentence* ) |
| | \| | ( *Sentence* ⇒ *Sentence* ) |
| | \| | ( *Sentence* ⇔ *Sentence* ) |

**Figure 7.7** A BNF (Backus–Naur Form) grammar of sentences in propositional logic.

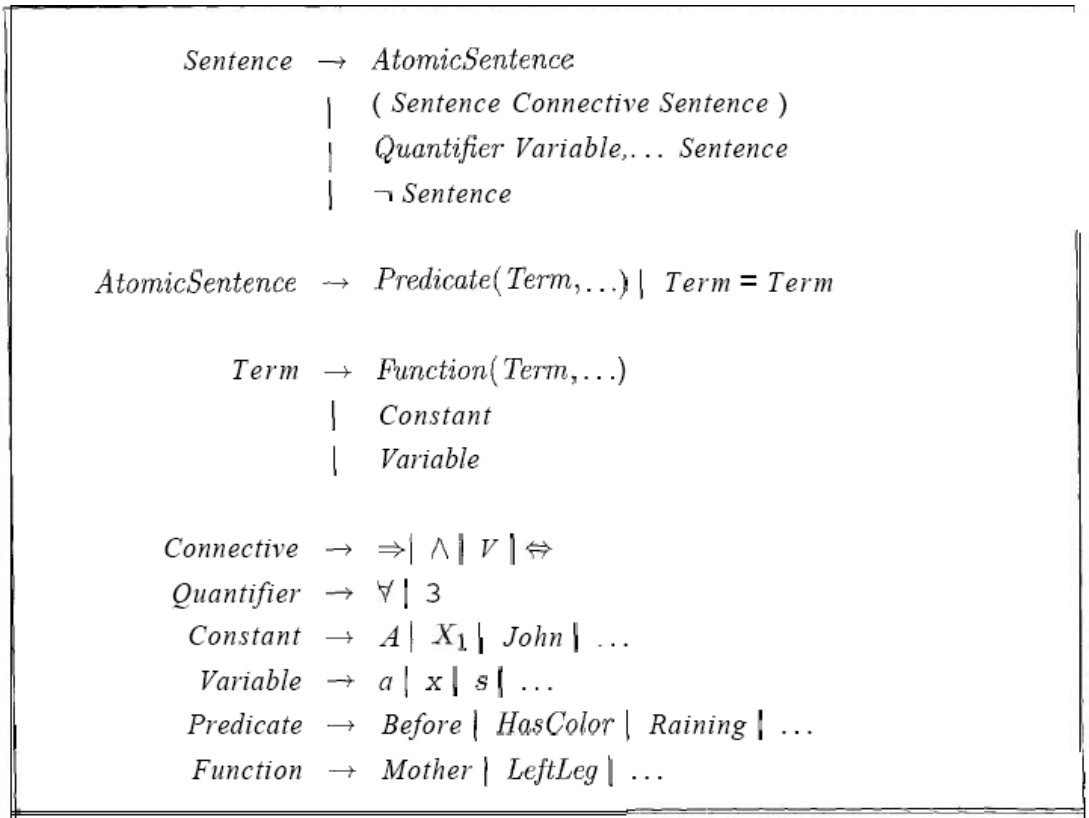
**(9) Define First order Logic?**

Whereas propositional logic assumes the world contains facts,
first-order logic (like natural language) assumes the world contains

Objects: people, houses, numbers, colors, baseball games, wars, …
Relations: red, round, prime, brother of, bigger than, part of, comes between, …
Functions: father of, best friend, one more than, plus, …

**(10) Specify the syntax of First-order logic in BNF form.**

**Powered By www.technoscriptz.com**



$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable,\ldots\ Sentence \\
&\mid \neg\ Sentence \\
\\
AtomicSentence &\rightarrow Predicate(Term,\ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term,\ldots) \\
&\mid Constant \\
&\mid Variable \\
\\
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLeg \mid \ldots
\end{aligned}
$$

**Figure 8.3** The syntax of first-order logic with equality, specified in Backus–Naur form. (See page 984 if you are not familiar with this notation.) The syntax is strict about parentheses; the comments about parentheses and operator precedence on page 205 apply equally to first-order logic.

(11) **Compare different knowledge representation languages.**

| Language | Ontological Commitment (What exists in the world) | Epistemological Commitment (What an agent believes about facts) |
|---|---|---|
| Propositional logic | facts | true/false/unknown |
| First-order logic | facts, objects, relations | true/false/unknown |
| Temporal logic | facts, objects, relations, times | true/false/unknown |
| Probability theory | facts | degree of belief $\in [0, 1]$ |
| Fuzzy logic | facts with degree of truth $\in [0, 1]$ | known interval value |

**Figure 8.1** Formal languages and their ontological and epistemological commitments.

(12) **What are the syntactic elements of First Order Logic**?

**Powered By www.technoscriptz.com**

The basic syntactic elements of first-order logic are the symbols that stand for objects,
relations, and functions. The symbols,come in three kinds:
a) constant symbols, which stand for objects;
b) predicate symbols, which stand for relations;
c) and function symbols, which stand for functions.
We adopt the convention that these symbols will begin with uppercase letters.
Example:
Constant symbols :
Richard and John;

predicate symbols :
Brother, OnHead, Person, King, and Crown;
function symbol :
LeftLeg.

**(13) What are quantifiers?**

There is need to express properties of entire collections of objects,instead of enumerating the objects by name. Quantifiers let us do this.
FOL contains two standard quantifiers called

a) Universal (∀**)** and
b) Existential (∃**)**

**Universal quantification**

(∀x) P(x) : means that P holds for **all** values of x in the domain associated with that variable
E.g., (∀x) dolphin(x) => mammal(x)

**Existential quantification**

(∃ x)P(x) means that P holds for **some** value of x in the domain associated with that variable
E.g., (∃ x) mammal(x) ^ lays-eggs(x)
Permits one to make a statement about some object without naming it

**(14) Explain Universal Quantifiers with an example.**

Rules such as "All kings are persons," is written in first-order logic as
∀x King(x) => Person(x)
where ∀ is pronounced as " For all .."

Thus, the sentence says, "For all *x,* if *x* is a king, then z is a person."
The symbol x is called a variable(lower case letters)
The sentence ∀**x** P,where P is a logical expression says that P is true for every object x.

**(15) Explain Existential quantifiers with an example.**

Universal quantification makes statements about every object.

Notesengine.com

It is possible to make a statement about some object in the universe without naming it,by using an existential quantifier.
Example
"King John has a crown on his head"
$\exists x$ Crown(x) ^ OnHead(x,John)

$\exists x$ is pronounced "There exists an x such that .." or " For some x .."

**(16) What are nested quantifiers?**

**Nested quantifiers**

We will often want to express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type. For example, "Brothers are siblings" can be written as

$$\forall x \ \forall y \ \ Brother(x, y) \Rightarrow Sibling(x, y).$$

Example-2
"Everybody loves somebody" means that
for every person,there is someone that person loves

$\forall x \exists y$ Loves(x,y)

**(17) Explain the connection between $\forall$ and $\exists$**

"Everyone likes icecream " is equivalent
"there is no one who does not like ice cream"
This can be expressed as :
$\forall x$ Likes(x,IceCream) is equivalent to
$\neg \exists \neg$Likes(x,IceCream)

**(18) What are the steps associated with the knowledge Engineering process?**

Discuss them by applying the steps to any real world application of your choice.

**Knowledge Engineering**

The general process of knowledge base constructiona process is called knowledge engineering.

A knowledge engineer is someone who investigates a particular domain, learns what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain. We will illustrate the knowledge engineering process in an electronic circuit domain that should already be fairly familiar,

**The steps associated with the knowledge engineering process are :**

1. ***Identfy the task***.

. The task will determine what knowledge must be represented in order to connect problem instances to answers. This step is analogous to the PEAS process for designing agents.
*2.* ***Assemble the relevant knowledge***. The knowledge engineer might already be an expert

Notesengine.c

in the domain, **or** might need to work with real experts to extract what they know-a process called **knowledge acquisition.**

**3.** ***Decide on a vocabulary of predicates, functions, and constants***. That is, translate the important domain-level concepts into logic-level names.

Once the choices have been made. the result is a vocabulary that is known as the **ontology** of the domain. The word *ontology* means a particular theory of the nature of being or existence.

*4. Encode general /cnowledge about the domain.* The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content. Often, this step reveals misconceptions or gaps in the vocabulary that must be fixed by returning to step 3 and iterating through the process.

*5. Encode a description of the specijic problem insttznce.*

For a logical agent, problem instances are supplied by the sensors, whereas a "disembodied" knowledge base is supplied with additional sentences in the same way that traditional programs are supplied with input data.

6. *Pose queries to the inference procedure and get answers.* This is where the reward is: we can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing.

*7. Debug the knowledge base.*

*∀x NumOfLegs(x,4) => Mammal(x)*

*Is false for reptiles ,amphibians.*

To understand this seven-step process better, we now apply it to an extended example-the domain of electronic circuits.

## The electronic circuits domain

We will develop an ontology and knowledge base that allow us to reason about digital circuits of the kind shown in Figure 8.4. We follow the seven-step process for knowledge engineering.

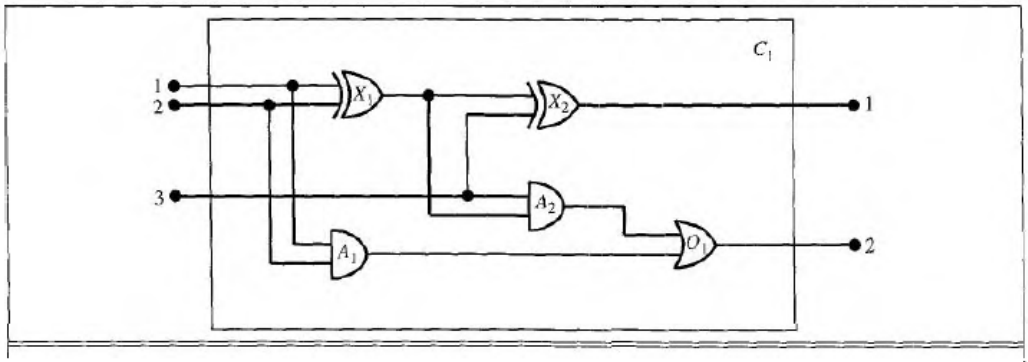



**Figure 8.4** A digital circuit C1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates and one OR gate.

**Identify the task**

engine.com

There are many reasoning tasks associated with digital circuits. At the highest level, one analyzes the circuit's functionality. For example, what are all the gates connected to the first input terminal? Does the circuit contain feedback loops? These will be our tasks in this section.

**Assemble the relevant knowledge**

What do we know about digital circuits? For our purposes, they are composed of wires and gates. Signals flow along wires to the input terminalls of gates, and each gate produces a signal on the output terminal that flows along another wire.

**Decide on a vocabulary**

We now know that we want to talk about circuits, terminals, signals, and gates. The next step is to choose functions, predicates, and constants to represent them. We will start from individual gates and move up to circuits.

First, we need to be able to distinguish a gate from other gates. This is handled by naming gates with constants: *X I , X2,* and so on

**Encode general knowledge of the domain**

One sign that we have a good ontology is that there are very few general rules which need to be specified. A sign that we have a good vocabulary is that each rule can be stated clearly and concisely. With our example, we need only seven simple rules to describe everything we need to know about circuits:

1. If two terminals are connected, then they have the same signal:
2. The signal at every terminal is either 1 or 0 (but not both):
3. Connected is a commutative predicate:
4. An OR gate's output is 1 if and only if any of its inputs is 1:
5. An **A.ND** gate's output is 0 if and only if any of its inputs is 0:
6. An XOR gate's output is 1 if and only if its inputs are different:
7. A NOT gate's output is different from its input:

**Encode the specific problem instance**

The circuit shown in Figure 8.4 is encoded as circuit *C1* with the following description. First, we categorize the gates:

*Type(X1)= XOR Type(X2)= XOR*

**Pose queries to the inference procedure**

What combinations of inputs would cause the first output of *Cl* (the sum bit) to be 0 and the second output of *C1* (the carry bit) to be l?

**Debug the knowledge base**

We can perturb the knowledge base in various ways to see what kinds of erroneous behaviors emerge.

## (19) Give examples on usage of First Order Logic.

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:

*TELL(KB, King (John))* .

Notesengine.com

Where KB is knowledge base.
*TELL(KB, ∀x King(x) => Person(x))*.

We can ask questions of the knowledge base using ASK. For example,

$$\text{ASK}(KB, King(John))$$

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, ∃**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 William" 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.*

**(20) What is universal instantiation?**

**Inference rules for quantifiers**

Let us begin with universal quantifiers. Suppose our knowledge base contains the standard folkloric axiom stating that all greedy kings are evil:

$$\forall x \;\; King(x) \wedge Greedy(x) \Rightarrow Evil(x) .$$

Then it seems quite permissible to infer any of the following sentences:

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John).$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard) .$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)).$$

The rule of **Universal Instantiation** (UI for short) says that we can infer any sentence obtained by substituting a **ground term** (a term without variables) for the variable. [1] To write out the inference rule formally, we use the notion of **substitutions** introduced in Section 8.3. Let $\text{SUBST}(\theta, a)$ denote the result of applying the substitution $\theta$ to the sentence $a$. Then the rule is written

$$\frac{\forall v \;\; a}{\text{SUBST}(\{v/g\}, \alpha)}$$

for any variable $v$ and ground term $g$. For example, the three sentences given earlier are obtained with the substitutions $\{x/John\}$, $\{x/Richard\}$, and $\{x/Father(John)\}$.

The corresponding **Existential Instantiation** rule: for the existential quantifier is slightly more complicated. For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \;\; \alpha}{\text{SUBST}(\{v/k\}, \alpha)} .$$

Universal instantiation (UI)

rgine.com

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

for any variable $v$ and ground term $g$

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:
  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$
  .
  .
  .

Existential instantiation (EI)

- For any sentence α, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

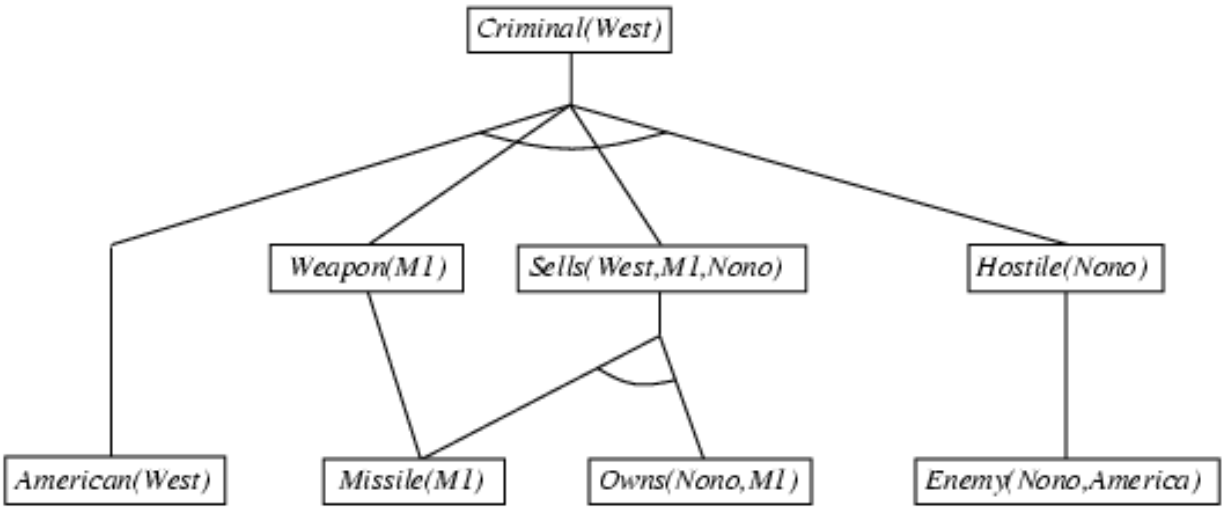
$$Crown(C_1) \wedge OnHead(C_1,John)$$

provided $C_1$ is a new constant symbol, called a Skolem constant

**(21) What is forward chaining? Explain with an example.**

Using a deduction to reach a conclusion from a set of antecedents is called forward chaining. In other words,the system starts from a set of facts,and a set of rules,and tries to find the way of using these rules and facts to deduce a conclusion or come up with a suitable couse of action. This is known as data driven reasoning.

EXAMPLE




Powered By www.technoscriptz.com

The proof tree generated by forward chaining.
Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Col. West is a criminal

... it is a crime for an American to sell weapons to hostile nations:
*American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):
*Owns(Nono,$M_1$) and Missile($M_1$)*
… all of its missiles were sold to it by Colonel West
*Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
Missiles are weapons:
*Missile(x) ⇒ Weapon(x)*
An enemy of America counts as "hostile“:
*Enemy(x,America) ⇒ Hostile(x)*
West, who is American …
*American(West)*
The country Nono, an enemy of America …
*Enemy(Nono,America)*

Note:
(a) The initial facts appear in the bottom level
(b) Facts inferred on the first iteration is in the middle level
(c) The facts inferered on the 2nd iteration is at the top level

Forward chaining algorithm

**function** FOL-FC-ASK($KB, \alpha$) **returns** a substitution or *false*

**repeat until** *new* is empty
  *new* ← { }
  **for each** sentence $r$ **in** $KB$ **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
    **for each** $\theta$ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in $KB$
      $q' \leftarrow$ SUBST($\theta, q$)
      **if** $q'$ is not a renaming of a sentence already in $KB$ or *new* **then do**
        add $q'$ to *new*
        $\phi \leftarrow$ UNIFY($q', \alpha$)
        **if** $\phi$ is not *fail* **then return** $\phi$
  add *new* to $KB$
**return** *false*

(22) **What is backward chaining ? Explain with an example.**

ssengine.com

Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived.

In **backward chaining** ,we start from a **conclusion**,which is the hypothesis we wish to prove,and we aim to show how that conclusion can be reached from the rules and facts in the data base.

The conclusion we are aiming to prove is called a goal ,and the reasoning in this way is known as **goal-driven**.

**Backward chaining example**

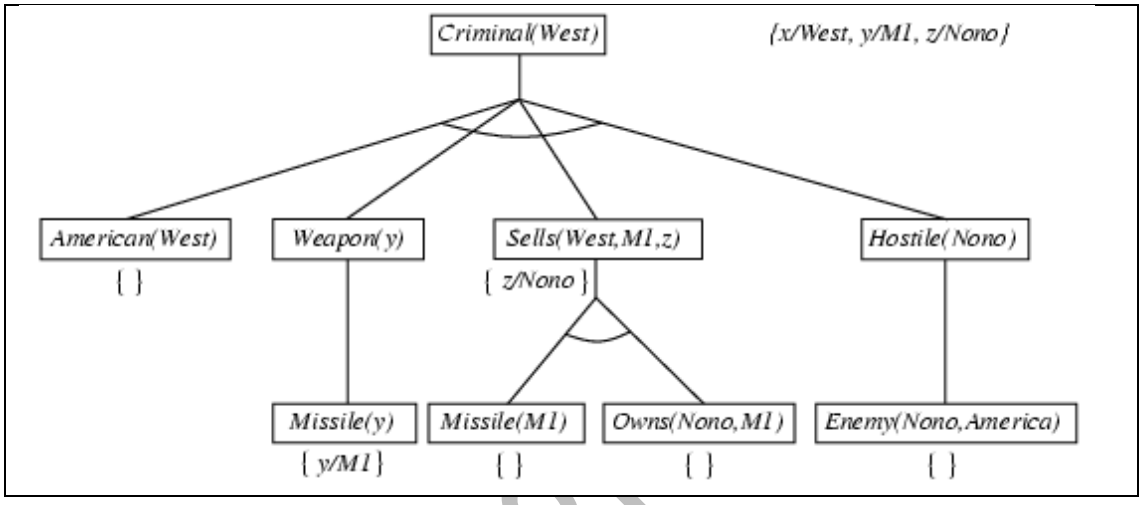



Fig : Proof tree constructed by backward chaining to prove that West is criminal.

Note:

(a) To prove Criminal(West) ,we have to prove four conjuncts below it.

(b) Some of which are in knowledge base,and others require further backward chaining.

(23) Explain conjunctive normal form for first-order logic with an example.

## Conjunctive normal form for first-order logic

As in the propositional case, first-order resolution requires that sentences be in **conjunctive normal form** (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.[6] Literals can contain variables, which are assumed to be universally quantified. For example, the sentence

$$\forall x \;\; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x).$$

*Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.* In particular, the CNF sentence will be unsatisfiable just when the original sentence

**Powered By www.technoscriptz.com**

is unsatisfiable, so we have a basis for doing proofs by contradiction on the CNF sentences. Here we have to eliminate existential quantifiers. We will illustrate the procedure by translating the sentence "Everyone who loves all animals is loved by someone," or

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$$

The steps are as follows:

◇ Eliminate implications:

$$ti\,x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)]\ \mathbf{v}\ [\exists y\ Loves(y,x)]$$

◇ Move ¬ inwards: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

$$\neg\forall x\ p \quad \text{becomes} \quad 3x\ \neg p$$
$$\neg\exists x\ p \quad \text{becomes} \quad \forall x\ \neg p\,.$$

Our sentence goes through the following transformations:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]\,.$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]\,.$$
$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)]\ \mathbf{v}\ [\exists y\ Loves(y,x)]\,.$$

Notice how a universal quantifier $(\forall y)$ in the premise of the implication has become an existential quantifier. The sentence now reads "Either there is some animal that $x$ doesn't love, or (if this is not the case) someone loves $x$." Clearly, the meaning of the original sentence has been preserved.

**(24) What is Ontological Engineering?**

Ontology refers to organizing every thing in the world into hierarch of categories. Representing the abastract concepts such as Actions,Time,Physical Objects,and Beliefs is called Ontological Engineering.

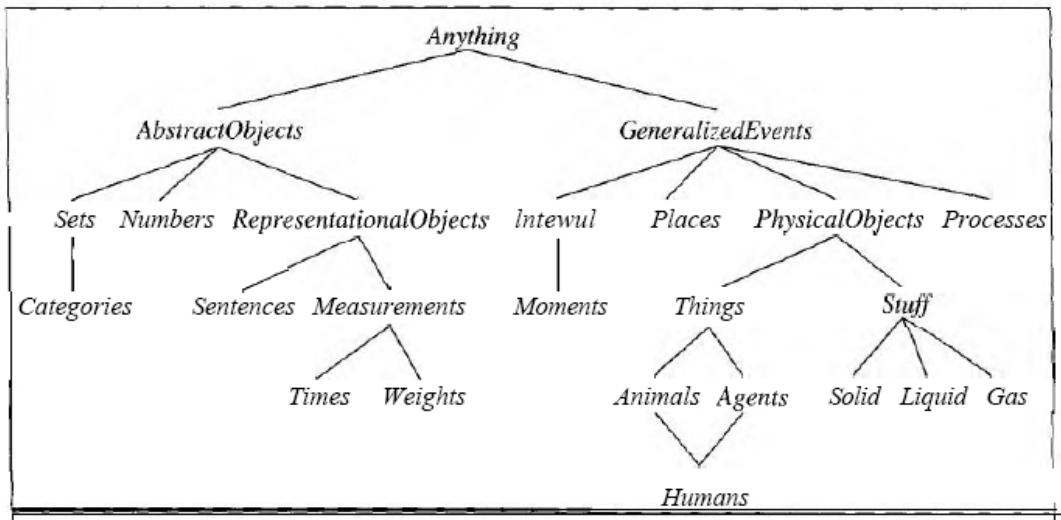



**Figure 10.1** The upper ontology of the world, showing the topics to be covered later in the chapter. Each arc indicates that the lower concept is a specialization of the upper one.

(25) How categories are useful in Knowledge representation?

**Powered By www.technoscriptz.com**

## CATEGORIES AND OBJECTS

The organization of objects into **categories** is a vital part of knowledge representation. Although interaction with the world takes place at the level of individual objects, *much reasoning takes place at the level of categories.*

### (26) What is taxonomy?

Subclass relations organize categories into a taxonomy, or taxonomic hierarchy. Taxonomies have been used explicitly for centuries in technical fields. For example, systematic biology aims to provide a taxonomy of all living and extinct species; library science has developed a taxonomy of all fields of knowledge, encoded as the Dewey Decimal system; and
tax authorities and other government departments have developed extensive taxoriornies of occupations and commercial products. Taxonomies are also an important aspect of general commonsense knowledge.
First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members:

- An object is a member of a category. For example:
  $BB_9 \in Basketballs$
- A category is a subclass of another category. For example:
  $Basketballs \subset Balls$
- All members of a category have some properties. For example:
  $x \in Basketballs \Rightarrow Round(x)$
- Members of a category can be recognized by some properties. For example:
  $Orange(x) \wedge Round(z) \wedge Diameter(x) = 9.5'' \wedge x \in Balls \Rightarrow x \in Basketballs$
- A category as a whole has some properties. For example:
  $Dogs \in DomesticatedSpecies$

:Notice that because *Dogs* is a category and is a member of *DomesticatedSpecies*, the latter must be a category of categories. One can even have categories of categories of categories, but they are not much use.

### (27) What is physical composition?

**Physical composition**

The idea that one object can be part of another is a familiar one. One's nose is part of one's head, Romania is part of Europe, and this chapter is part of this book. We use the general *PartOf* relation to say that one thing is part of another. Objects can be grouped into *PartOf* hierarchies, reminiscent of the *Subset* hierarchy:

$PartOf(Bucharest, Romania)$
$PartOf(Romania, EasternEurope)$
$PartOf(EasternEurope, Europe)$
$PartOf(Europe, Earth)$.

The *PartOf* relation is transitive and reflexive; that is,

$PartOf(x,y) \wedge PartOf(y,z) \Rightarrow PartOf(x,z)$.
$PartOf(x,x)$.

Therefore, we can conclude $PartOf(Bucharest, Earth)$.

**Powered By www.technoscriptz.com**

(28) **Explain the Ontology of Situation calculus**.

**Situations** are logical terms consisting of the initial situation (usually called *So)* and all situations that are generated by applying an action to a situation. The function *Result(a, s)* (sometimes called *Do)* names the situation that results when action *a* is executed in situation *s.* Figure 10.2 illustrates this idea.

**Fluents** are functions and predicates that vary from one situation to the next, such as the location of the agent or the aliveness of the wumpus. The dictionary says a fluent is something that fllows, like a liquid. In this use, it means flowing or changing across situations. By convention, the situation is always the last argument of a fluent. For example, *lHoldzng(G1, So)* says that the agent is not holding the gold GI in the initial situation *So. Age( Wumpus, So)* refers to the wumpus's age in *So.*

**Atemporal** or **eternal** predicates and functions are also allowed. Examples include the predicate Gold (GI) and the function *LeftLeg Of* ( *Wumpus).*

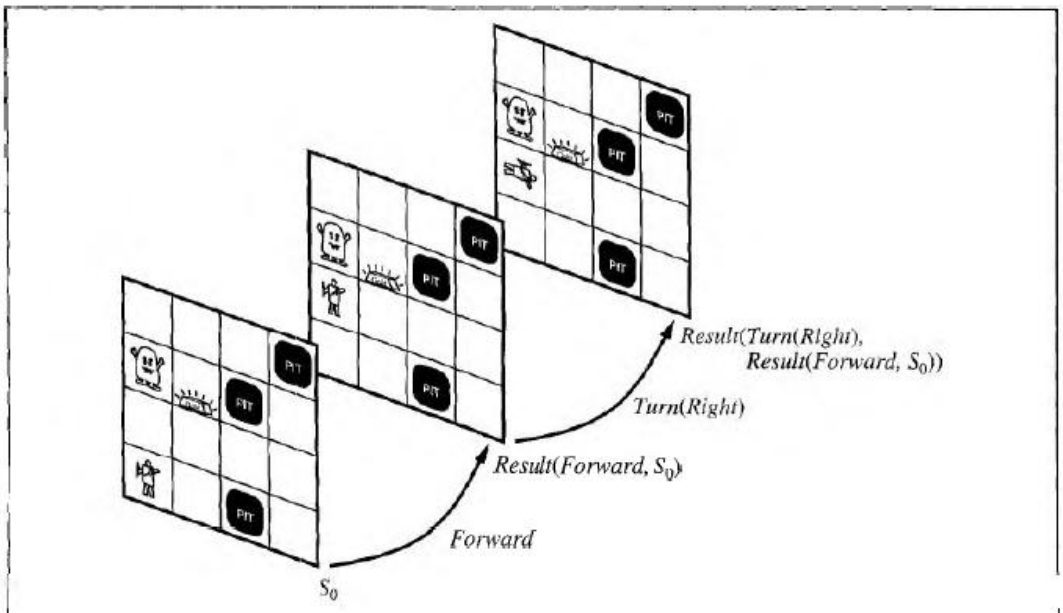



**Figure 10.2** In situation calculus, each situation (except $S_0$) is the result of an action.

(29) What is event calculus?

## Time and event calculus

Situation calculus works well when there is a single agent performing instantaneous, discrete actions. When actions have duration and can overlap with each other, situation calculus becomes somewhat awkward. Therefore, we will cover those topics with an alternative for- EVENTCALCULUS malism known as **event calculus,** which is based on points in time rather than on situations.

(The terms "event-' and "action" may be used interchangeably. Informally, "event" connotes a wider class of actions, including ones with no explicit agent. These are easier to handle in event calculus than in situation calculus.)

In event calculus, fluents hold at points in time rather than at situations, and the calculus is designed to allow reasoning over intervals of time. The event calculus axiom says that a

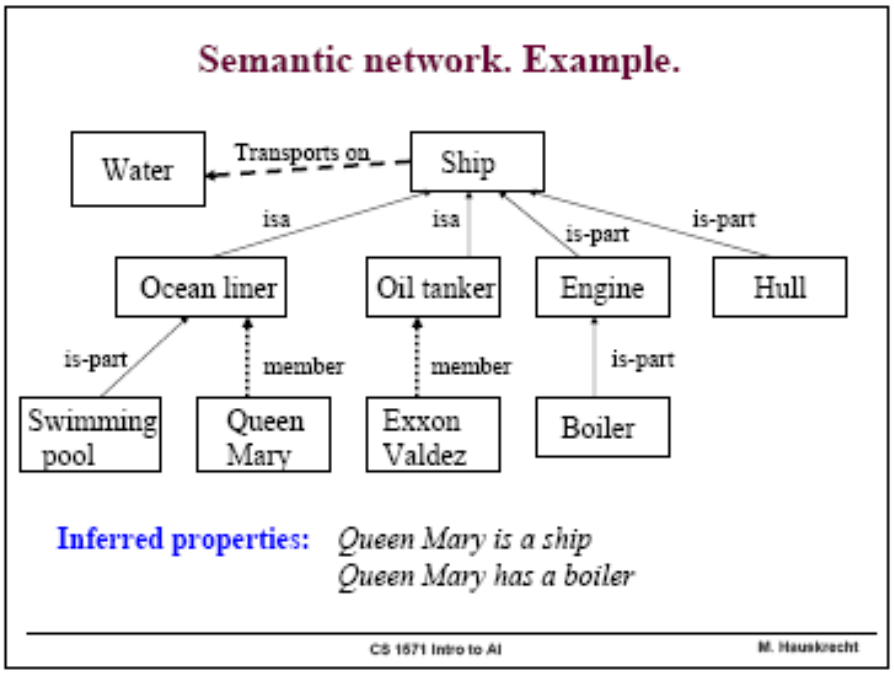
**Powered By www.technoscriptz.com**

fluent is true at a point in time if the fluent was initiated by an event at some time in the past and was not terminated by an intervening event. The *Initiates* and *Terminates* relations play a role similar to the *Result* relation in situation calculus; *Initiates(e, f , t)* means that the occurrence of event *e* at time *t* causes fluent *f* to become true, while *Terminates* (w , f, *t)* means that *f* ceases to be true. We use *Happens(e, t)* to mean that event *e* happens at time *t,*

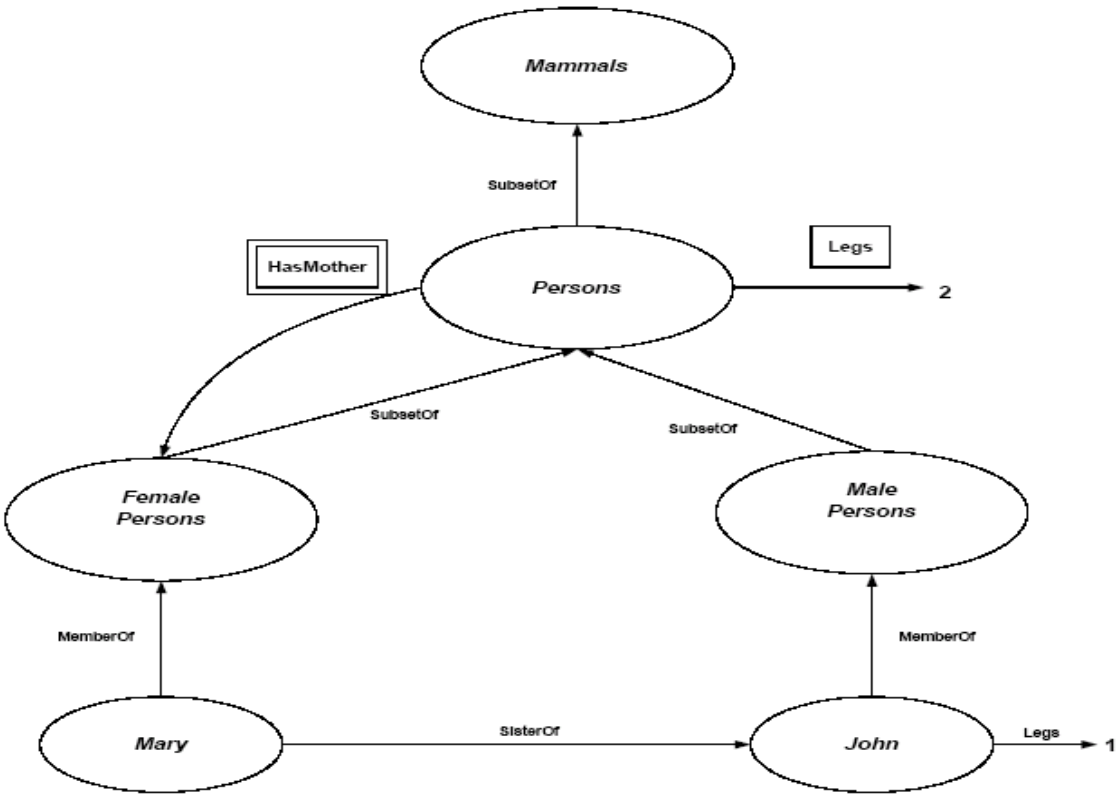
(30) What are semantic networks?

(31) Semantic networks are capable of representing individual objects,categories of objects,and relation among objects. Objects or Ctegory names are represented in ovals and are connected by labeled arcs.

Semantic network example




**Powered By www.technoscriptz.com**




Notesengi

**Powered By www.technoscriptz.com**

# Artificial Intelligence I

Matthew Huntbach, Dept of Computer Science, Queen Mary and Westfield College, London, UK E1 4NS. Email: mmh@dcs.qmw.ac.uk. Notes may be used with the permission of the author.

# Notes on Reasoning with Uncertainty

So far we have dealt with knowledge representation where we know that something is either true or false. Stepping beyond this assumption leads to a large body of work in AI, which there is only time in this course to consider very briefly. Three different approaches, representing three different areas of uncertainty, are considered. The first two represent two different forms of quantitative uncertainty; that is where we attempt to give numerical values expressing the degree to which we are uncertain about pieces of knowledge. Probabilistic reasoning deals with cases where something is definitely true or false, but we do not know which and so we reason about the chances that something is true or false. Fuzzy logic deals with cases where something could be partially true; we can think of it as dealing with "shades of grey" rather than the "black or white" of classical logic.

The third approach, truth maintenance systems, is just one example of a non-monotonic reasoning system, that is one where adding new items of knowledge may cause conclusions we had previously drawn to become invalid. It is a form of common-sense reasoning, as mentioned in the first set of notes for this course, where we can start off by making assumptions which may not actually be correct, so that we can express any conclusion drawn as dependent on assumptions, and we can change assumptions as necessary, particularly when we find they lead to conflicting conclusions.

## Probabilistic Reasoning

I mentioned briefly in the set of notes on predicate logic that one of the problems with the use of classical logic as a knowledge representation system is that it assumes that we can define all values as either completely true or completely false. In many cases we may wish to assign fractional values representing the fact that we are either uncertain as to whether something is true or false, but have some idea as to which is more likely, or that we wish to regard a value as only partially true.

On the uncertainty issue, the most long-established way of dealing with it is probability theory. There is not space in this course to go into much detail on probability theory (there are other courses taught in this college which do that) so I will just give some basics which fit into the approach to knowledge representation taken so far.

In the probabilistic approach, a logic sentence is labelled with a real number in the range 0 to 1, 0 meaning the sentence is completely false, and 1 meaning it is completely false. A label of 0.5 means there is equal chance that the sentence is true or false.

Simple probability is often illustrated by games of chance. If a coin is flipped, there is an equal chance of it landing on the heads side or the tails side, so if we say that $H_1$ stands for "on the first toss of the coin it lands heads-up", we can express this by $pr(H_1)=0.5$. Let us say that $H_2$ stands for "on the second toss of the coin it lands heads-up", then we know that $pr(H_2)$ is 0.5 as well. It is well known that the probability of two successive events occurring which do not interfere or depend on each other can be obtained by multiplying the probabilities of each occurring. We know that whether a coin lands heads or tails is in no way dependent in how it previously landed (assuming the coin is perfectly balanced), so that the probability of the first and second toss both landing on heads is 0.5*0.5=0.25. We can write this $pr(H_1 \wedge H_2)=0.25$, and in general for two independent events $P$ and $Q$, $pr(P \wedge Q)=pr(P)*pr(Q)$.

In many cases, however, we cannot assume this independence. For example, let us suppose that in a college of 1000 students, 100 students take the logic course and 200 take the programming course. If $P_F$ stands for "student Fred takes the programming course" and $L_F$ stands for "Student Fred takes the logic course" where we know no more about Fred except that he is one of the 1000 students at the college, we can say that $pr(P_F)=0.2$ and $pr(L_F)=0.1$. It would be wrong to conclude though that $pr(P_F \wedge L_F)=0.2*0.1=0.02$, since we know that in fact the two subjects are related and many who take one would take the other. If however, we have another student, Wilma, and all we know about Wilma is that she is also a student at the same college, then we can correctly conclude

$pr(P_F \wedge L_W)=0.2*0.1=0.02$ where $L_W$ stands for "Wilma takes logic" since what one arbitary student takes has no dependence on what another arbitary student takes. In all cases in probability theory if $pr(P)$ is the probability that $P$ is true, then the probability that $P$ is false is $1-pr(P)$, so we can say that $pr(P)=1-pr(\neg P)$.

If we know in fact that 60 of the 200 students taking the programming course also take the logic course, then we can say that the probability that Fred takes the logic course if we know that he takes the programming course is $\frac{60}{200}$ = 0.3. We write this $pr(L_F|P_F)=0.3$, where in general $pr(Q|P)$ means "the probability of $Q$ when we know that $P$ is true". So from this, if we don't know whether $Q$ is true, but we know the probability of $Q$ being true if $P$ is true, we can calculate the probability of $P$ and $Q$ being true: $pr(P \wedge Q)=pr(P)*pr(Q|P)$. So $pr(P_F \wedge L_F)= 0.2*0.3=0.06$. If $P$ and $Q$ are independent then $pr(Q|P)=pr(Q)$.

Since $\wedge$ is a commutative operator (i.e., we can swap its arguments around without affecting its meaning) $pr(P \wedge Q)=pr(Q \wedge P)$, so $pr(P)*pr(Q|P)=pr(Q)*pr(P|Q)$. From this, it follows that:

$$pr(P|Q)=\frac{pr(P)*pr(Q|P)}{pr(Q)} \quad .$$

This is known as *Bayes' rule* after the philosopher who discovered it. In our example, it means that the probability that Fred takes the programming course if we know that he takes the logic course is $\frac{0.2*0.3}{0.1}$ =0.6. Typically this rule is used to find the probability of some hypothesis $P$ being true given that we know some evidence $Q$.

Note that since $P \vee Q$ is equivalent to $\neg(\neg P \wedge \neg Q)$, then using the rule for $pr(\neg P)$ above we can obtain $pr(P \vee Q) = pr(P)+pr(Q)-pr(P \wedge Q)$. We can also obtain a value for $pr(P \rightarrow Q)$ in this manner, using the fact that $P \rightarrow Q$ is just another way of writing $\neg P \vee Q$. A distinction should be made between $pr(P \rightarrow Q)$ and $pr(Q|P)$, which are two separate things. $pr(P \rightarrow Q)$ is just the probability that $P \rightarrow Q$ holds for some arbitary $x$, recalling that $P \rightarrow Q$ is always true when $P$ is false.

A more general statement of Bayes' theorem covers the case where we have some evidence E, and a range of hypotheses, $H_1, H_2, \ldots, H_n$ which could be drawn from it. The probability of any particular hypothesis is:

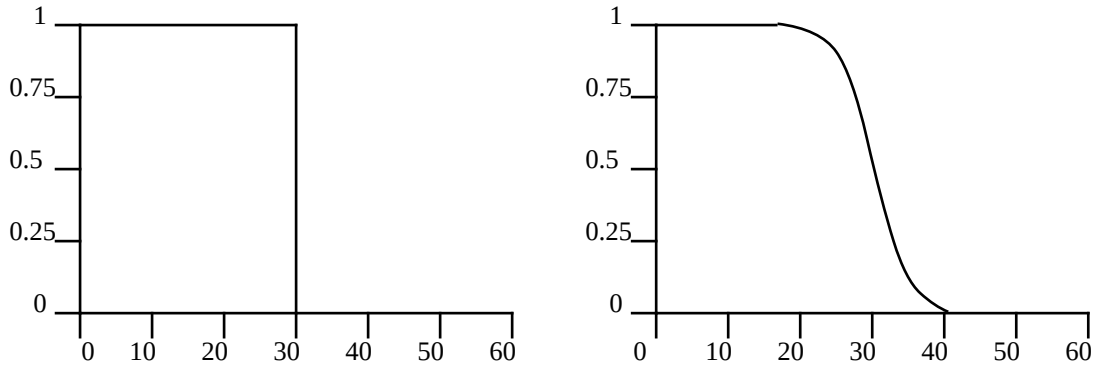
$$pr(H_i|E) = \frac{pr(E|H_i)*pr(H_i)}{\sum_{k=1}^{n}(pr(E|H_k)*pr(H_k))}$$

The problem with this is it makes the assumption that the statistical data on the relationships of the evidence with the hypotheses are known, and that all relationships between evidence and hypotheses $pr(E|H_k)$ are independent of each other. In general this indicates that as the number of factors we are considering increases it becomes increasingly difficult to be able to account for the possible interferences between them.

## Fuzzy Logic

Fuzzy logic, developed by the computer scientist Lotfi Zadeh, is an alternative more intuitive approach to reasoning with imprecise knowledge. It does not claim to have the mathematical precision of probability theory, but rather to capture common-sense notions.

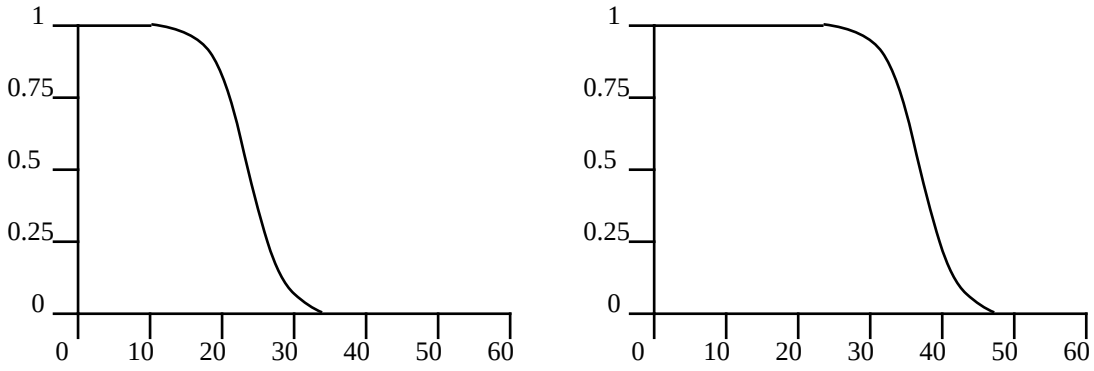
The background to the notion of fuzzy logic is the idea that a truth value may apply partially, and that human reasoning often uses this idea of partial truth. For example, the sentence "Fred is young" might be taken as definitely true if Fred is 19 years old, definitely false is Fred is 50, but applying partially if Fred is 28. In standard logic we might have a sharp cutoff point, say 30, and we would then say for any person $p$ with age below 30, $young(p)$ is true, but for any person $q$ above 30 $young(p)$ is false. In fuzzy logic a numerical value between 0 and 1 is given to express the degree to which a predicate applies. This can be expressed in graphical form, where the graph on the left represents the classical logic approach (the x-axis represents age, the y-axis degree of truthfulness):




This represents a situation where anyone under the age of 20 is reckoned as definitely young, anyone over the age of 40 is reckoned as definitely not young, but the predicate is partial or vague for those between the ages of 20 and 40. More precisely, where a predicate p is defined by the set of elements for each of which p(x) is true, in fuzzy logic a predicate p is defined by a set of pairs ‹x,$\mu_p$(x)›, where $\mu_p$(x) is the degree of membership of x in the set which defines p.

Note the difference between this and the probabilistic reasoning described above. When we said that pr($L_F$) was 0.5 we did not mean that Fred was half taking the logic course, we meant that the probability that Fred was taking the course was 0.5. If in our example above the age of Fred is 30, if Fred is represented by f, and the fuzzy predicate young by y, then from the graph above $\mu_y$(f) is 0.5, meaning that Fred is indeed "half young".

Fuzzy logic also captures the idea of qualifiers or *hedges* to predicates in natural language, such as "Fred is very young" or "Fred is fairly young". These might be represented in fuzzy logic by graphs which move the curbed part to the left and to the right of the scale respectively:




The degree of membership of an item x in the union of two fuzzy sets A and B is the maximum of its degree of membership in A or B, while the degree of membership in the intersection is the minimum. So the truth value of P∧Q for two sentences in fuzzy logic P and Q is the minimum of the truth values of P and Q, while the truth value of P∨Q is the maximum of the truth value for P and Q. The truth value for ¬P is the truth value for P subtracted from 1. So negation is dealt with in a similar way to probabilistic logic, but the connectives are not.

As an example, let us consider reasoning about the set of subjects in a hypothetical Philosophy department. This set, called Ω, is the "frame of discernment" (set of all possible values x for an attribute). In our example, this set is:

{Epistemology, Methodology, Aesthetics, Metaphysics, Logic, Politics, Theology}

Suppose we have a predicate Interesting, representing how interesting each subject is. In classical logic we would simply gave to divide the subjects up into those that are interesting and those that are not. If we suppose that Aesthetics, Politics and Theology are interesting, we could represent the predicate Interesting by the set

{Aesthetics, Politics, Theology}

However, the reality is that we will not have a straightforward division into interesting and non-interesting subjects, but rather rate them on a scale of interestingness (note also that, of course, what

one person finds interesting another may not, but for the sake of space we shall not explore this additional factor). Rather than impose an arbitary cut-off point on this scale, fuzzy logic would rate the subjects on a 0 to 1 scale. So perhaps `Aesthetics` has an interest factor of 0.9, `Politics` an interest factor of 0.7, `Theology` an interest factor of 0.8, `Metaphysics` and `Methodology` each an interest factor of 0.5, `Logic` an interest factor of 0.3, and `Epistemology` an interest factor of 0.1. We can represent this by the fuzzy set:

```
{<Epistemology,0.1>, <Methodology,0.5>, <Aesthetics,0.9>, <Metaphysics,0.5>,
                <Logic,0.3>, <Politics,0.7>, <Theology,0.8>}
```

which we shall call `I`. The function $\mu_I$ gives the interest factor of any particular subject, so $\mu_I$`(Epistemology)` is 0.1, $\mu_I$`(Methodology)` is 0.5, and so on. The cutoff point for the classical logic version was between 0.5 and 0.7. In fact there is a name for this: if A is a fuzzy set in $\Omega$ then the $\alpha$-cut $A_\alpha$ is defined by

$$A_\alpha = \{\omega \in \Omega \mid \mu_A(\omega) \geq \alpha\}$$

The set we suggested for the classical logic set for `Interesting` above might perhaps be the $I_{0.6}$ cut. Similarly, $I_{0.4}$ is `{Methodology, Aesthetics, Metaphysics, Politics, Theology}` (note that though we are restricting values of $\mu_I$`(x)` to single decimal places for convenience, this is not essential).

Dealing with negation, `Interesting(x)` is $\mu_I$`(x)`, so `¬Interesting(x)` is `1-`$\mu_I$`(x)`. For example, `¬Interesting(Politics)` is `0.3`. Indeed, we can define a set `¬I` representing the fuzzy set of uninteresting things simply by replacing all the $\mu_I$`(x)` in `I` with $1-\mu_I$`(x)`:

```
{<Epistemology,0.9>, <Methodology,0.5>, <Aesthetics,0.1>, <Metaphysics,0.5>,
                <Logic,0.7>, <Politics,0.3>, <Theology,0.2>}
```

Now let us suppose there is another fuzzy set, `H`, represent how hard each subject is:

```
{<Epistemology,0.7>, <Methodology,0.6>, <Aesthetics,0.4>, <Metaphysics,0.9>,
                <Logic,0.5>, <Politics,0.2>, <Theology,0.8>}
```

Applying the rules of fuzzy set theory, the set `H`$\cap$`I` is:

```
{<Epistemology,0.1>, <Methodology,0.5>, <Aesthetics,0.4>, <Metaphysics,0.5>,
                <Logic,0.3>, <Politics,0.2>, <Theology,0.8>}
```

while the set `H`$\cup$`I` is:

```
{<Epistemology,0.7>, <Methodology,0.6>, <Aesthetics,0.9>, <Metaphysics,0.9>,
                <Logic,0.5>, <Politics,0.7>, <Theology,0.8>}
```

So we now have a measure of the degree to which a subject fits into the class of "Hard and interesting things"represented by the set `H`$\cap$`I`, and a measure of the extent to which it falls into the class of "Hard or interesting things" represented by the set `H`$\cup$`I`.

Note that whereas in classical logic, `A`$\wedge$`¬A` is always false, so the set representing `A`$\wedge$`¬A` is always the empty set, this is not so in fuzzy logic. The set `I`$\wedge$`¬I`, representing the class of things which are both interesting and not interesting is:

```
{<Epistemology,0.1>, <Methodology,0.5>, <Aesthetics,0.1>, <Metaphysics,0.5>,
                <Logic,0.3>, <Politics,0.3>, <Theology,0.2>}
```

so it can be seen that the subjects which have the highest degree of membership of this class are those that are in the middle of the interesting–not-interesting scale. Clearly, the maximum degree to which any item can have a membership of the set `A`$\wedge$`¬A` is 0.5.

Most of the expected properties of a boolean algebra apply, for example, de Morgan's rules: `¬(A`$\wedge$`B)` is equal to `¬A`$\vee$`¬B`. We could define a $\rightarrow$ symbol where as in classical logic `A`$\rightarrow$`B` is equivalent to `¬A`$\vee$`B`, so for example, the set representing `Hard`$\rightarrow$`Interesting` is:

```
{<Epistemology,0.3>, <Methodology,0.5>, <Aesthetics,0.9>, <Metaphysics,0.5>,
                <Logic,0.5>, <Politics,0.8>, <Theology,0.8>}
```

which could be seen as a measure to which the various subjects fit the notion that the hard subjects are the interesting one. In fact classical predicate logic could be considered just the special case of fuzzy logic where $\mu(\omega)$ is restricted to be 0 or 1.

Recalling the suggestion that $\forall x A(x)$ might be considered as equivalent to $A(n_1) \wedge A(n_2) \wedge \ldots A(n_m)$ for $n_1 \ldots n_m$ ranging over the objects in the universe, $\forall x S$ for $S$ a sentence in fuzzy logic is simply the minimum $\mu$ value for any pair in the set representing $S$, while similarly since $\exists x A(x)$ was considered as representing $A(n_1) \vee A(n_2) \vee \ldots A(n_m)$, has the value of the maximum $\mu$ value for any pair in the set representing $S$. So $\forall x \mathtt{Interesting}(x)$ has the value 0.1 and $\exists x \mathtt{Interesting}(x)$ has the value 0.9.

## Truth Maintenance

Truth Maintenance Systems (TMSs) are methods of keeping track of reasoning that may change with time as new items of knowledge are added. They are by their nature non-monotonic, as they are based on the idea that some "facts" may only be assumptions which are believed until disproved by further knowledge being learnt. Truth Maintenance Systems keep a record showing which items of knowledge are currently believed or disbelieved along with links to other items of knowledge on which this belief or disbelief is based. Because of this, if a new fact is learnt or an assumption changed the ripple effect it has on other beliefs based on the previous assumptions can be managed by tracing along the links. Although the name "Truth Maintenance System" has stuck as it was used to describe early versions of this sort of system, a better name that has been proposed and is also used is *Reason Maintenance System*, as these systems are more about belief and disbelief and managing the reaosn for belief or disbelief than about truthhood or falsehood.

Truth maintenance systems are another form of knowledge representation which is best visualised in terms of graphs. The idea is that each item of knowledge which a system may believe is represented by a node in the graph. These nodes are labelled either *in* (for a node which represents something which is currently believed) or *out* (representing something which is not currently believed). Note the distinction between in and out representing beliefs, and true and false representing absolute values.

There are several forms of truth maintenance system. For illustration we shall start off with a simple one. The links in the graph are directed and represent *justifications*, and are of two sorts, positive and negative. A positive link from node $A$ to node $B$ represents the idea that believing in $A$ is a reason for believing in $B$, or $A$ is a justification for believing in $B$. It is equivalent to the logic $A \rightarrow B$. A negative link from $A$ to $B$ represents the idea that believing in $A$ is a reason for not believing in $B$, the equivalent to logic $A \rightarrow \neg B$.

For example, let us suppose we are representing the reasoning over whether we should put a roof on a pen in which we are going to keep some animal called Fred. We might then reason

"If Fred can fly, the pen needs a roof" or $\mathtt{Fly} \rightarrow \mathtt{Roof}$

"If Fred is a bird, Fred can fly" or $\mathtt{Bird} \rightarrow \mathtt{Fly}$

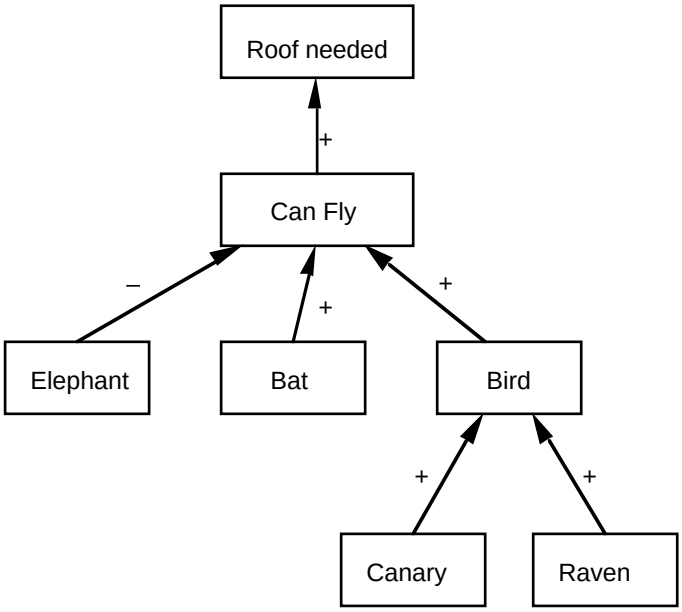
"If Fred is an elephant, Fred cannot fly" or $\mathtt{Elephant} \rightarrow \neg \mathtt{Fly}$.

"If Fred is a bat, Fred can fly" or $\mathtt{Bat} \rightarrow \mathtt{Fly}$

"If Fred is a canary, Fred is a bird" or $\mathtt{Canary} \rightarrow \mathtt{Bird}$.

"If Fred is a raven, Fred is a bird" or $\mathtt{Raven} \rightarrow \mathtt{Bird}$.

This gives us the graph:



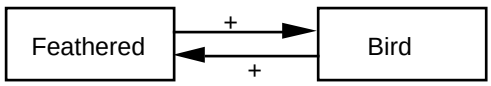

Reasoning works by labelling the leaf nodes (i.e. those without any incoming arcs) as *in* or *out*. This represents our *assumptions*. If we assume Fred is a canary, and Fred is not an Elephant, a Bat or a Raven, we label the node `Canary` as *in*, and the nodes `Elephant`, `Bat` and `Raven` as *out*. Any internal node which has at least one positive arc leading from an *in* node, and no negative arc from an *in* node is then labelled as *in*, any internal node which has at least one negative arc from an *in* node, and no positive arc from an *in* node is labelled as *out*. This process continues until all nodes with at least one arc coming from an *in* node are labelled as *in* or *out*. In the above example, this will result in `Can Fly` and `Roof needed` being labelled as in.

Note that if a node is *out*, it is not used for any justification of any sort, so there is no negation-as-failure effect of assuming that if we can't find a justification for something it must necessarily be false. For example, let us suppose we assume Fred is neither an elephant, a bat, a canary or a raven. This means there are no justifications for the node `Bird`. In Prolog, with negation as failure we would then assume that Fred is not a bird. However in the TMS if we want to assume that Fred is a bird of a sort we don't yet know we would label `Bird` as *in*, and in general any nodes without justifications may be labelled as *in* or *out*, representing further assumptions.

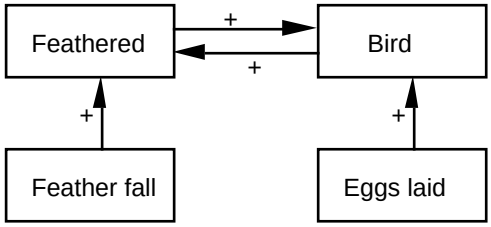
We can change our assumptions as we like, so any node labelled *in* or *out* without justifications can be changed to *out* or *in*. Any such change of assumptions means that other nodes which have arcs leading from these assumptions may have to be changed, and the change will ripple through the system. Suppose in the the above example we changed our assumptions and wanted to assume that Fred is an elephant, and not a bat, canary or raven. We would then label `Elephant` as *in*, and `Bat`, `Canary` and `Raven` as *out*. This would mean that `Can Fly` would have to have its label changed to *out*. `Roof needed`, though, would not necessarily have to have its label changed. It would have no *in* justifications necessitating it being labelled either *in* or *out*, and could be labelled in either way representing an assumption. This makes sense since, though we have ruled out Fred having the ability to fly being an argument for needing a roof, we still need to account for the possibility that there is some other reason not yet known why a roof might be needed.

Now suppose that we started off with our original assumptions that `Canary` is *in*, while `Elephant`, `Bat` and `Raven` are *out*, and then decided to change our minds and assume that Fred is an elephant without also dropping the assumption that Fred is a canary. This would cause `Can Fly` to have both a negative justification *in* and a positive justification *in*. This situation is termed a contradiction: we have made assumptions that lead to a contradictory conclusion, both that Fred can fly and Fred cannot fly. When this happens, the TMS goes through a process known as *dependency-directed backtracking*, which means that it traces down the arcs from *in* nodes leading to the contradictory node to find all the assumptions on which the contradictory justifications depend i.e. all nodes reached by tracing down the links and getting to the point where a node has no further *in* links leading to it. One of the assumptions would then have to be changed; in this case it would note that the assumptions are `Elephant` and `Canary`, and one of these would have to be made *out* to remove the contradiction.

Problems arise in this truth maintenance system if there are circular chains of justifications. For example, we might argue that "Fred has feathers" is a justification for assuming Fred is a bird, while if we know Fred is a bird we can assume Fred has feathers. This would lead to the structure:

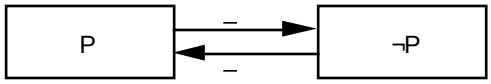



If this occurred as part of the justification graph, on what is said above we could say that neither `Bird` nor `Feathered` is just an assumption since both have justifications supporting them. However we have here what is known popularly as a "circular argument" – if we have no other support for it (that is a positive justification pointing inwards to one of the nodes from a node which is not part of the circle) the entire argument is just an assumption, and the same would apply for a circular argument of three or more nodes. We might decide that if we have observed Fred laying eggs that is a justification for Fred being a bird, or if we have observed a feather fall from Fred that is a justification for Fred being feathered:

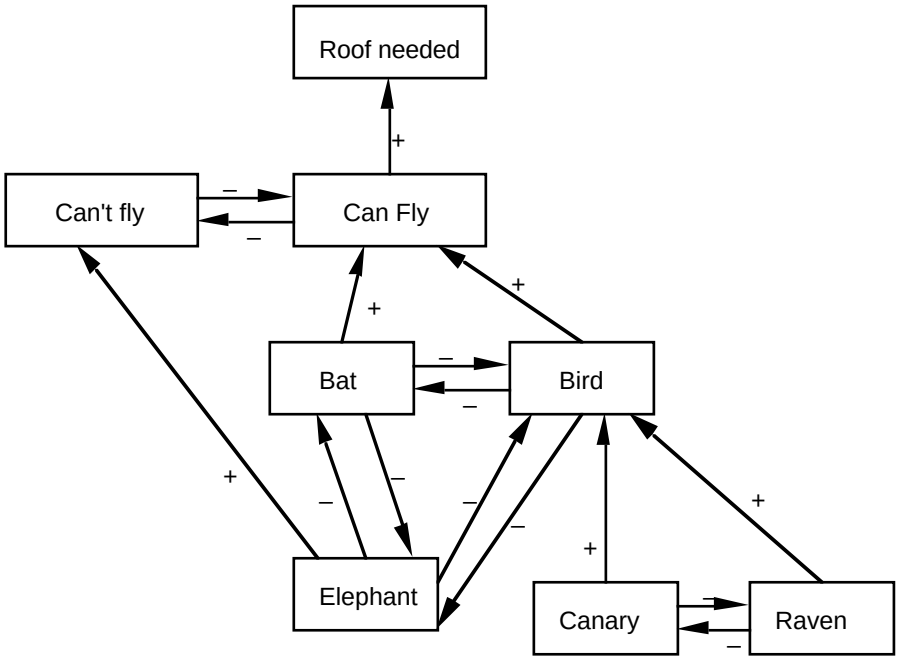



In this case if either `Feather fall` or `Eggs laid` becomes in, it is a support for the whole circular argument.

Note that mutual negative justifications are acceptable, and in fact give the effect of negation:




Here it is not possible for both `P` and `¬P` to be *`in`* without it causing a contradiction. It is possible for both to be *`out`*, representing the case where `P` is unknown. This can be generalised so that if there is a set of options out of which at most one can be true, the node for each option is linked to the rest by mutual negative links. If we want to show that something can only be one out of a bat, a bird and an elephant, and also only one out of a raven and a canary, we could modify our diagram to that below. Note how we have also introduced a separate node for the negation of `Can Fly`. Note that this graph does not rule out the possibility of a bird which is neither a raven nor a canary. In this case, `Bird` would be *`in`*, but both `Raven` and `Canary` would be *`out`*.

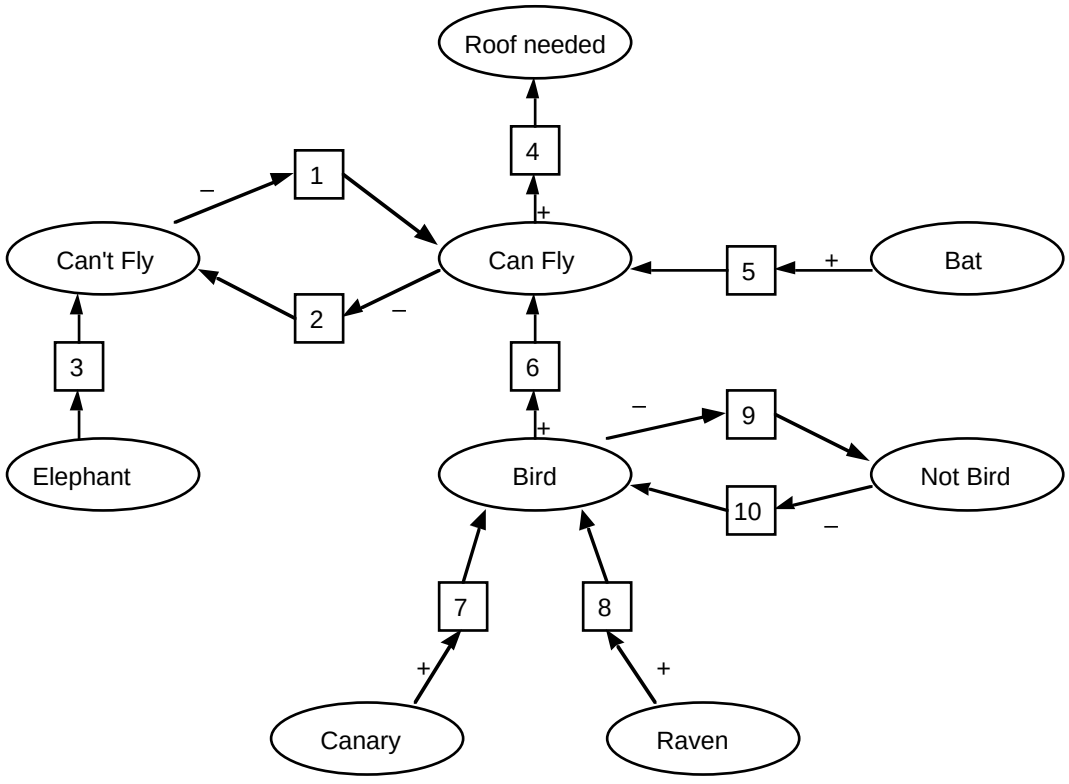


The truth-maintenance system described above was a deliberately simplistic one given to demonstrate some of the ideas behind truth maintenance. It suffers in particular from the fact that the *out*ness of some node cannot be used as a justification for the *in*ness of another. This means, for example, that we cannot have a full negation effect, since we cannot use ¬P becoming *out* as a justification for P coming *in*, rather both P and ¬P stay *out* (representing P is unknown) unless a change of assumption is made.

## Doyle's Truth Maintenace System

The original Truth Maintenance System, as proposed by Jon Doyle, had a rather more complex and hence more powerful system of assumptions and justifications. In Doyle's TMS a justification is not a single belief, but a set of beliefs, or rather two sets of beliefs. For a justification to be *valid*, all of its first set of beliefs must be *in*, and all of its second set *out*. For this reason, these two lists are referred to as the *inlist* and *outlist* respectively. A belief is *in* if at least one of its justifications is valid, but unlike the system described above there is no concept of negative justifications forcing a belief to be *out*. A belief is an *assumption* if it relies on a justification with a non-empty outlist, that is it is *in* because of the outness of some other beliefs. An assumption may be justified by its own negation being *out*, similar to the circular belief structure described above. A justification with both inlist and outlist empty is always valid, so any belief it justifies will always be *in*. Such a justification is referred to as a *premise* justification.

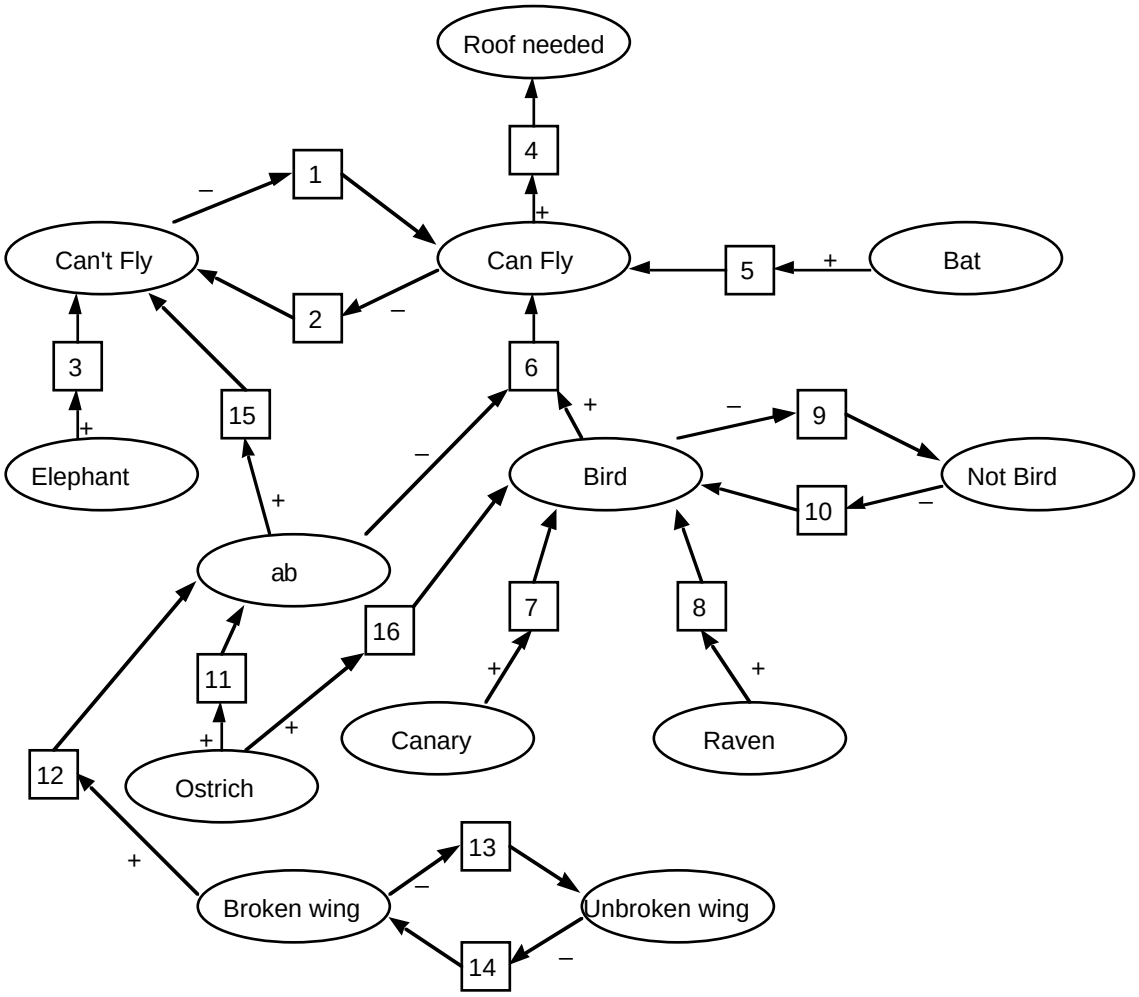
A graphical representation of the set of beliefs and justifications requires a graph with two different sorts of nodes. Such a graph, illustrating the situation described above is shown below. Beliefs are represented by elliptical nodes, and justifications by square nodes. For convenience in discussion, the justifications are numbered.




The resulting graph is similar to that shown before as in this case each justification has a single belief. In this graph, however, a justification supported by a belief being *out* can be used to make another belief *in*, for example making `Can't Fly` *out* makes justification 1 valid, and therefore the belief `Can Fly` becomes *in*. `Can Fly` is justified by any of justifications 1, 5 or 6 being *in*. In practice there would have to be some further beliefs and justifications not shown, for example `Elephant` would have to be justified by the outness of a node `Not Elephant` and vice versa.

Unlike our previous system, this one can correctly deal with default reasoning. Previously we noted that we would like to say that we will assume that a bird can fly providing it is not an abnormal non-flying bird, such as an ostrich, or one whose wing is broken. Adding this to our system creates a

justification for `Can Fly` which depends both on the inness of `Bird` and the outness of `ab` (meaning "Fred is a non-flying bird"). Such a set of beliefs is shown below:

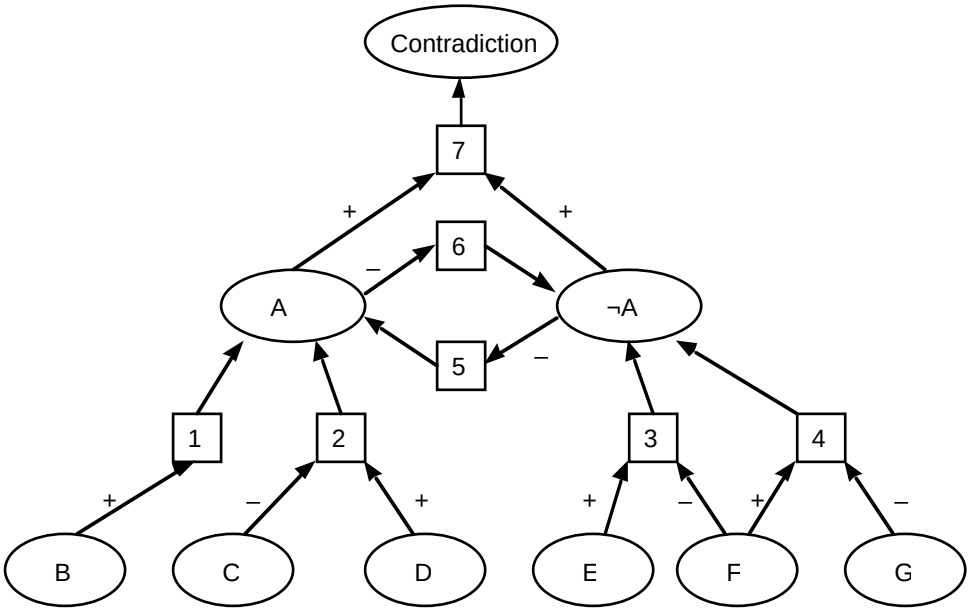



Here, we can start off just by assuming Fred is a bird by having `Bird` *`in`*, supported by `Not Bird` being *`out`*, while if `Canary`, `Raven` and `Ostrich` are also *`out`* we are not making any assumptions as to the sort of bird Fred might be. We will also assume that Fred does not have a broken wing, so `Broken Wing` is *`out`*. This means there is no justification for `ab`, so we can make `ab` *`out`*, making justification 6 valid, and hence `Can Fly` is *`in`*. We have successfully shown that from the assumption that Fred is a bird and without any further assumptions, we can draw the further assumption that Fred can fly.

If, however, we then add the assumption that Fred is an ostrich or Fred has a broken wing by making either `Ostrich` or `Broken Wing` *`in`*, we have a valid justification for `ab`. This means that 15 becomes a valid justification supported by `ab`, while 6 becomes invalid since one of its outlist is *`in`*. `Can't fly` becomes *`in`*, supported by justification 15, while `Can fly` goes *`out`* because 6 is invalid, and 1 too is invalid since 1's outlist now also has an *`in`* in it; we assume that `Bat` is also *`out`*, so the remaining possible justification for `Can Fly`, 5, is invalid.

Note that, correctly, the simple assertion that Fred is an ostrich will lead to justifying both that Fred is a bird, and that Fred cannot fly, while the assertion that Fred is a canary or Fred is a raven will lead to justifying both that Fred is a bird and that Fred can fly.

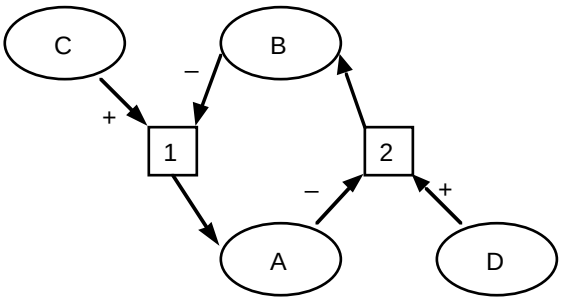
As described so far, this truth maintenance system does not have any way of removing assumptions that lead to contradictions, or indeed of recognising contradictions. Two facts which are contradictory may both be in so long as they both have valid assumptions. The way contradictions re dealt with is to have explicit contradiction nodes. If a contradiction node becomes *`in`*, it is noted and at least one of the assumptions that led to it becoming *`in`* has to be retracted. As an abstract example, consider the following diagram:



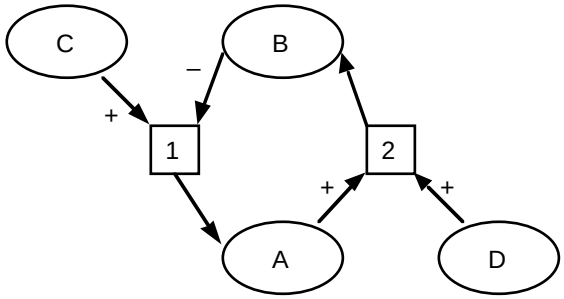

in a situation where B, D and E are assumed *in*, while C, F and G are assumed *out*. This makes 1, 2 and3 valid justifications, so both A and ¬A are *in*, since a belief is *in* so long as any of its justifications are valid. 4 is not a valid justification, since it requires F to be *in* whereas it is *out*. The fact that both 5 and 6 are invalid does not make A or ¬A *out*, since they both have other valid justifications. However, a contradiction node is added with justification 7 which becomes valid only when A and ¬A are *in*. The contradiction being *in* requires sufficient assumptions being changed to make it *out*.

The assumptions that are changed must be sufficient either to make A *out* or to make ¬A *out*. This means that both 1 and 2 must be made invalid or 3 must be made invalid. To make 1 and 2 invalid, the assumption for B must be changed to make it *out* and also C must be made *in* and D *out*. To make ¬A *out*, 3 must be made invalid, but note this must be done in a way that does not make 4 valid. So if F were made *in* that would not be sufficient since it would then make make 4 valid so ¬A would still be supported, though by 4 rather than 3. So to make ¬A *out* either E must be made *out*, or both F and G must be made *in*.

Note the difficulty that cycles of justifications may cause in this Truth Maintenance system. If there is an even number of non-monotonic justifications (i.e. links labelled -) in a cycle, the result is that there is more than one way in which the beliefs in the cycle can be *in* or *out*. In the case below if C and D are *in* and there are no more justifications for A and B, then if A is *in*, 2 is not a valid justification so B is *out* making 1 a valid justification supporting A's *in*ness. If A is *out* then 2 is a valid justification supporting B's *in*ness and making 1 invalid so there is no support for A so it stays *out*. A *in* and B *out*, or A *out* and B *in* are two different sets of beliefs one could infer from assuming C and D are *in*.




If there is an odd number of non-monotonic justifications in a cycle the problem is there may be no possible way of making the beliefs *in* or *out*, for example in:




if C and D are *in* and there are no further justifications for A and B, with A *in*, 2 is a valid justification, making B *in*, but then 1 is invalid making A *out*. But with A *out*, 2 is invalid so B is out, which makes 1 valid putting A *in* and so on.

## Further Reading

Books on probability are more likely to be found in the Mathematics section than in the Artificial Intelligence section of the library. Two books which are particularly on the application of probability theory to artificial intelligence are:

F.Bacchus *Representing and Reasoning with Probabilistic Knowledge* MIT Press 1990.

R.E.Neapolitan *Probabilistic Reasoning in Expert Systems* Wiley 1990.

There has been a recent growth in interest in fuzzy logic, and it has found a variety of applications. Three books discussing it are:

A.Kandel *Fuzzy Techniques in Pattern Recognition* Wiley 1982.

C.V.Negoita *Expert Systems and Fuzzy Systems* Benjamin/Cummings 1985.

B.Souč ek *Fuzzy Holographic and Parallel Intelligence* Wiley 1992.

Doyle's original paper on his Truth Maintenance System, as well as many other key papers on issues in non-monotonic reasoning are found in the collection

M.L.Ginsberg (ed) *Readings in Nonmonotonic Reasoning* Morgan Kaufmann 1987.

Another book on non-monotonic reasoning is:

W.Lukaszewicz *Non-monotonic Reasoning* Ellis Horwood 1990.

A book on truth maintenance systems is:

B.Smith and G.Kelleher (eds) *Reason Maintenance Systems and their Applications* Ellis Horwood 1988.

# UNIT-V Question and Answers

**(1) Define communication.**

**Communication** is the intentional exchange of information brought about by the production and perception of **signs** drawn from a shared system of conventional signs. Most animals use signs to represent important messages: food here, predator nearby etc. In a partially observable world, communication can help agents be successful because they can learn information that is observed or inferred by others.

**(2) What is speech act?**

What sets humans apart from other animals is the complex system of structured messages known as **language** that enables us to communicate most of what we know about the world. This is known as speech act.

**Speaker, hearer,** and **utterance** are generic terms referring to any mode of communication. The term **word** is used to refer to any kind of conventional communicative sign.

(3) **What are the capabilities gained by an agent from speech act?**.

- **Query** other agents about particular aspects of the world. This is typically done by asking questions: *Have you smelled the wumpus anywhere?*
- **Inform** each other about the world. This is done by making representative statements: *There's a breeze here in 3 4.* Answering a question is another Pund of informing.
- **Request** other agents to perform actions: *Please help me carry the gold.* Sometimes **indirect speech act** (a request in the form of a statement or question) is considered more polite: *I could use some help carrying this.* An agent with authority can give commands *(Alpha go right; Bravo and Charlie go lejl),* and an agent with power can make a threat *(Give me the gold, or else).* Together, these kinds of speech acts are called **directives.**
- **Acknowledge** requests: **OK.**
- **Promise** or commit to a plan: *I'll shoot the wumpus; you grab the gold.*

**(4) Define formal language.**

**A formal language** is defined as a (possibly infinite) set of' **strings.** Each string is a concatenation of **terminal symbols,** sometimes called words. For example, in the language of first-order logic, the terminal symbols include A and P, and a typical string is "P A Q." . Formal languages such as first-order logic and Java have strict mathematical definitions. This is in contrast to **natural languages,** such as Chinese, Danish, and English, that have no strict definition but are used by a community.

**(5) Define a grammar.**

A **grammar** is a finite set of rules that specifies a language. Formal languages always have an official grammar, specified in manuals or books. Natural languages have no official grammar, but linguists strive to discover properties of the language by a process of scientific inquiry and then to codify their discoveries in a grammar.

**(6) What are the component steps of communication? Explain with an example.**

**The component steps of communication**



Notesengine.com

A typical communication episode, in which speaker S wants to inform hearer H about proposition P using words W, is composed of seven processes:

**1) Intention.** Somehow, speaker S decides that there is some proposition P that is worth saying to hearer H. For our example, the speaker has the intention of having the hearer know that the wumpus is no longer alive.

**2) Generation.** The speaker plans how to turn the proposition P into an utterance that makes it likely that the hearer, upon perceiving the utterance in the current situation, can infer the meaning P (or something close to it). Assume that the speaker is able to come up with the words "The wumpus is dead," and call this *W*.

**3) Synthesis.** The speaker produces the physical realization *W'* of the words *W*. This can be via ink on paper, vibrations in air, or some other medium. In Figure 22.1, we show the agent synthesizing a string of sounds *W'* written in the phonetic alphabet defined on page 569: "[thaxwahmpaxsihzdehd]." The words are run together; this is typical of quickly spoken speech.

**4) Perception.** H perceives the physical realization W' as *Wi* and decodes it as the words *W2*. When the medium is speech, the perception step is called **speech recognition;** when it is printing, it is called **optical character recognition.**

**5) Analysis.** H infers that *W2* has possible meanings *PI, . . . , P,*.

We divide analysis into three main parts:

a) **syntactic interpretation (or parsing**),

b) **Semantic interpretation,** and

c)**Pragmatic interpretation**.

**Parsing** is the process of building a **parse tree** for an input string, as shown in Figure 22.1. The interior nodes of the parse tree represent phrases and the leaf nodes represent words.

**Semantic interpretation** is the process of extracting the meaning of an utterance as an expression in some representation language. Figure 22.1 shows two possible semantic interpretations: that the wumpus is not alive and that it is tired (a colloquial meaning of dead). Utterances with several possible interpretations are said to be **ambiguous.**

**Pragmatic interpretation** takes into account the fact that the same words can have different meanings in different situations.

**6) Disambiguation. H** infers that *S* intended to convey P, (where ideally *P,* = P).

Most speakers are not intentionally ambiguous, but most utterances have several feasible interpretations.. Analysis generates possible interpretations;if more than one interpretation is found, then disambiguation chooses the one that is best.

7) **Incorporation. H** decides to believe *P,* (or not). A totally naive agent might believe everything it hears, but a sophisticated agent treats the speech act as evidence for P,, not confirmation of it.

Putting it all together, we get the agent program shown in Figure 22.2. Here the agent acts as a robot slave that can be commanded by a master. On each turn, the slave will answer a question or obey a command if the master has made one, and it will believe any statements made by the master. It will also comment (once) on the current situation if it has nothing more pressing to do, and it will plan its own action if left aloine.

Here is a typical dialog:

ROBOT SLAVE MASTER
I feel a breeze. Go to 12.
Nothing is here. Go north.
I feel a breeze and I smell a stench
and I see a glitter. Grab the gold.

Notesengine.com

**Powered By www.technoscriptz.com**

Fig 22.1 shows the seven processes involved in communication,using the example sentence “The wumpus is dead”.

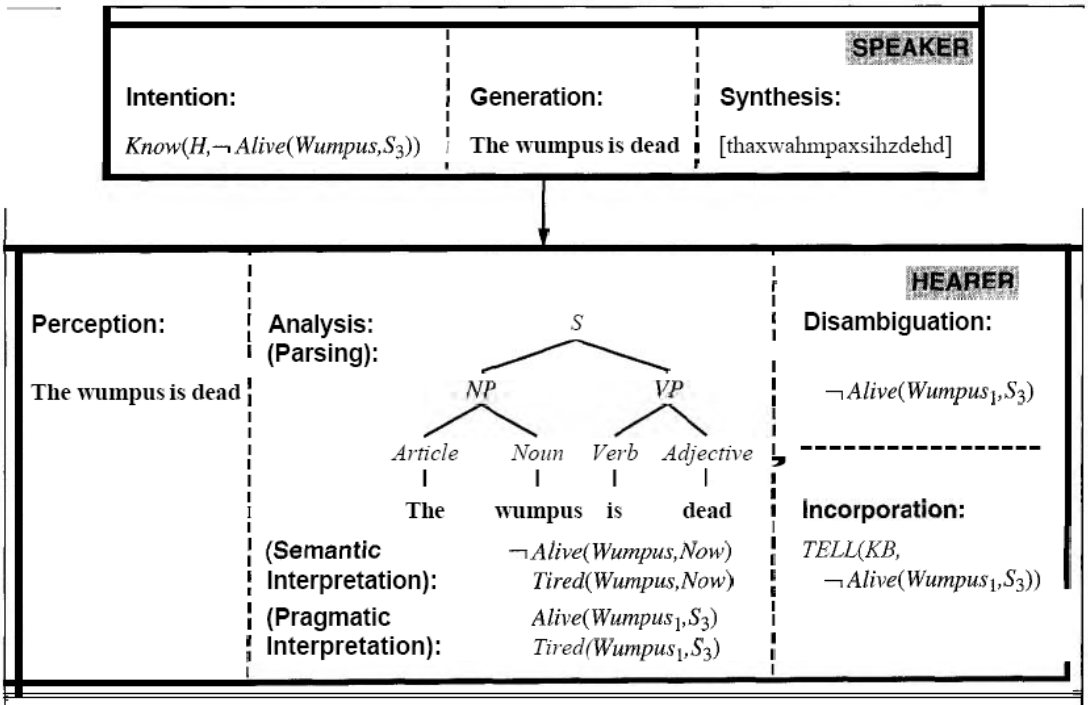



**Figure 22.1** *Seven processes involved in communication, using the example sentence "The wumpus is dead."*

**(7) Define a Lexicon and grammar for language consisting of a small fragment of English.**

### The Lexicon of $\mathcal{E}_0$

First we define the **lexicon,** or list of allowable words. The words are grouped into the categories or parts of speech familiar to dictionary users: nouns, pronouns, and names to denote

things, verbs to denote events, adjectives to modify nouns, and adverbs to modify verbs. Categories that are perhaps less familiar to some readers are articles (such as the), prepositions (in), and conjunctions (and). Figure 22.3 shows a small lexicon.

| | | |
|---|---|---|
| *Noun* | → | stench \| breeze \| glitter \| **nothing** \| agent |
| | | \| wumpus \| pit \| **pits**\| **gold** \| east \| ... |
| *Verb* | → | is \| see \| smell \| shoot \| feel \| stinks |
| | | \| go \| grab \| carry \| kill \| turn \| ... |
| *Adjective* | → | right \| left \| east \| dead \| back \| smelly \| .. |
| *Adverb* | → | here \| there \| nearby \| ahead |
| | | \| right \| left \| east \| south \| **back** \| ... |
| *Pronoun* | → | me \| you \| I \| it \| ... |
| *Name* | → | John \| Mary \| Boston \| **Aristotle** \| ... |
| *Article* | → | the \| a \| an \| ... |
| *Preposition* | → | to \| in \| on \| near \| ... |
| *Conjunction* | → | and \| or \| but \| ... |
| *Digit* | → | **0**\| **1**\| **2**\| **3**\| **4**\| **5**\| **6**\| **7**\| **8**\| **9** |

**Figure 22.3** The lexicon for $\mathcal{E}_0$.

**Powered By www.technoscriptz.com**

## The Grammar of $\mathcal{E}_0$

The next step is to combine the words into phrases. We will use five nonterminal symbols to define the different kinds of phrases: sentence *(S)*, noun phrase (NP), verb phrase (VP), prepositional phrase (PP), and relative clause e el clause).' Figure 22.4 shows a grammar for l o , with an example for each rewrite rule. lo generates good English sentences such as the following:

John is in the pit
The wumpus that stinks is in 2 2

| | | | |
|---|---|---|---|
| *S* | → | *NP VP* | *I* + feel a breeze |
| | \| | *S Conjunction S* | I feel a breeze + and + I smell a wumpus |
| *NP* | → | *Pronoun* | *I* |
| | \| | *Name* | John |
| | \| | *Noun* | pits |
| | \| | *Article Noun* | the + wumpus |
| | \| | *Digit Digit* | 3 4 |
| | \| | *NP PP* | the wumpus + to the east |
| | \| | *NP RelClause* | the wumpus + that is smelly |
| *VP* | → | *Verb* | stinks |
| | \| | *VP NP* | feel + a breeze: |
| | \| | *VP Adjective* | is + smelly |
| | \| | *VP PP* | turn + to the east |
| | \| | *VP Adverb* | go + ahead |
| *PP* | → | *Preposition NP* | to + the east |
| *RelClause* | → | that *VP* | that + is smelly |

**Figure 22.4** The grammar for $\mathcal{E}_0$, with example phrases for each rule.

**(8) What is parsing? Explain the top down parsing method.**

Parsing is defined as the process of finding a **parse tree** for a given input string. That is, a call to the parsing function PARSE, such as

PARSE("the wumpus is dead", $\mathcal{E}_0$, $S$)

should return a parse tree with root S whose leaves are "the wumpus is dead" and whose internal nodes are nonterminal symbols from the grammar Eo.

*Parsing can be seen as a process of searching for a parse tree.*

There are two extreme ways of specifying the search space (and many variants in between). First, we can start with the *S* symbol and search for a tree that has the words as its leaves. This is called **top-down parsing**

Second, we could start with the words and search for a tree with root S. This is called **bottom-up parsing.**

Top-down parsing can be precisely defined as a search problem as follows:

- The initial state is a parse tree consisting of the root S and unknown children: [S: ?I. In general, each state in the search space is a parse tree.

The successor function selects the leftmost node in the tree with unknown children. It

then looks in the grammar for rules that have the root label of the node on the left-hand side. For each such rule, it creates a successor state where the ? is replaced by a list corresponding to the right-hand side of the rule..

**(9) Formulate the bottom-up parsing as a search problem.**

The formulation of bottom-up parsing as a search is as follows:

The **initial state** is a list of the words in the input string, each viewed as a parse tree that is just a single leaf node-for example; **[the, wumlpus, is, dead].** In general, each state in the search space is a list of parse trees.

The **successor function** looks at every position ***i*** in the list of trees and at every righthand side of a rule in the grammar. If the subsequence of the list of trees starting at *i* matches the right-hand side, then the subsequence is replaced by a new tree whose category is the left-hand side of the rule and whose children are the subsequence. By "matches," we mean that the category of the node is the same as the element in the righthand side. For example, the rule *Article* + **the** matches the subsequence consisting of the first node in the list **[the, wumpus, is, dead],** so a successor state would be *[[Article:***the], wumpus, is, dead].**

The **goal test** checks for a state consisting of a single tree with root S.

See Figure 22.5 for an example of bottom-up parsing.

| *step* | *list of nodes* | *subsequence* | *rule* |
|---|---|---|---|
| INIT | **the wumpus is dead** | **the** | Article → **the** |
| 2 | *Article* **wumpus is dead** | **wumpus** | Noun → **wumpus** |
| 3 | *Article Noun* **is dead** | *Article Noun* | *NP → Article Noun* |
| 4 | *NP* **is dead** | **is** | *Verb* → **is** |
| 5 | *NP Verb* **dead** | **dead** | *Adjective* → **dead** |
| 6 | **NP** *Verb Adjective* | *Verb* | *VP → Verb* |
| 7 | *NP VP Adjective* | *VP Adjective* | *VP → VP Adjective* |
| 8 | *NP VP* | *NP VP* | *S → NP VP* |
| GOAL | S | | |

**Figure 22.5** Trace of a bottom up parse on the string "The wumpus is dead." We start with a list of nodes consisting of words. Then we replace subsequences that match the right-hand side of a rule with a new node whose root is the left-hand side. For example, in the third line the Article and Noun nodes are replaced by an NP node that has those two nodes as children. The top-down parse would produce a similar trace, but in the opposite direction.

**(10) What is dynamic programming?**

Forward Chaining on graph search problem is an example of dynamic programming. Solutions to the sub problems are constructed incrementally from those of smaller sub problems and are cached to avoid recomputation.

**(11) Construct a parse tree for "You give me the gold" showing the sub categories of the verb and verb phrase.**

**Powered By www.technoscriptz.com**

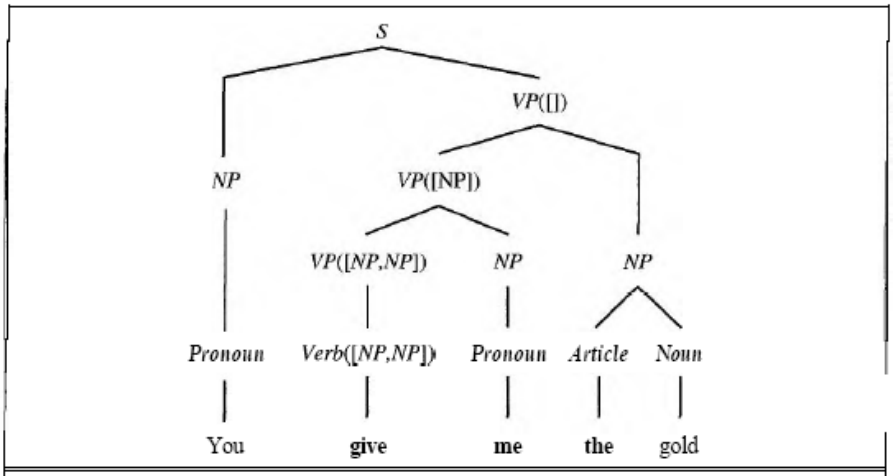



Figure 22.13 Parse tree for "You give me the gold" showing subcategorization of the verb and verb phrase.

**(12) What is semantic interpretation? Give an example.**

Semantic interpretation is the process of associating an FOL expression with a phrase.

$$
\begin{aligned}
&Exp(x) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\}\\
&Exp(x) \rightarrow (\ Exp(x)\ )\\
&Exp(x) \rightarrow Number(x)\\
&Number(x) \rightarrow Digit(x)\\
&Number(x) \rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\}\\
&Digit(x) \rightarrow x\ \{0 \le x \le 9\}\\
&Operator(x) \rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

**Figure 22.14** A grammar for arithmetic expressions, augmented with semantics. Each variable $x_i$ represents the semantics of a constituent. Note the use of the $\{test\}$ notation to define logical predicates that must be satisfied, but that are not constituents.

**(13) Construct a grammar and sentence for "John loves Mary"**

$$
\begin{aligned}
&S(rel(obj)) \rightarrow NP(obj)\ VP(rel)\\
&VP(rel(obj)) \rightarrow Verb(rel)\ NP(obj)\\
&NP(obj) \rightarrow Name(obj)\\
&\\
&Name(John) \rightarrow \textbf{John}\\
&Name(Mary) \rightarrow \textbf{Mary}\\
&Verb(\lambda y\ \lambda x\ Loves(x, y)) \rightarrow \textbf{loves}
\end{aligned}
$$

**Figure 22.16** A grammar that can derive a parse tree and semantic interpretationfor "John loves Mary" (and three other sentences). Each category is augmented with a single argument representing the semantics.

**Powered By www.technoscriptz.com**

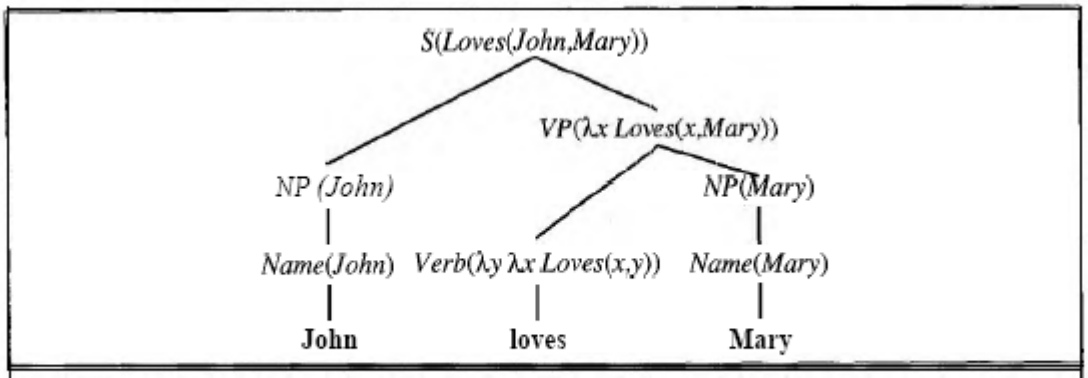



**Figure 22.17** A parse tree with semantic interpretations for the string "John loves Mary".

**(14) Construct a parse tree for the sentence "Every agent smells a Wumpus"**

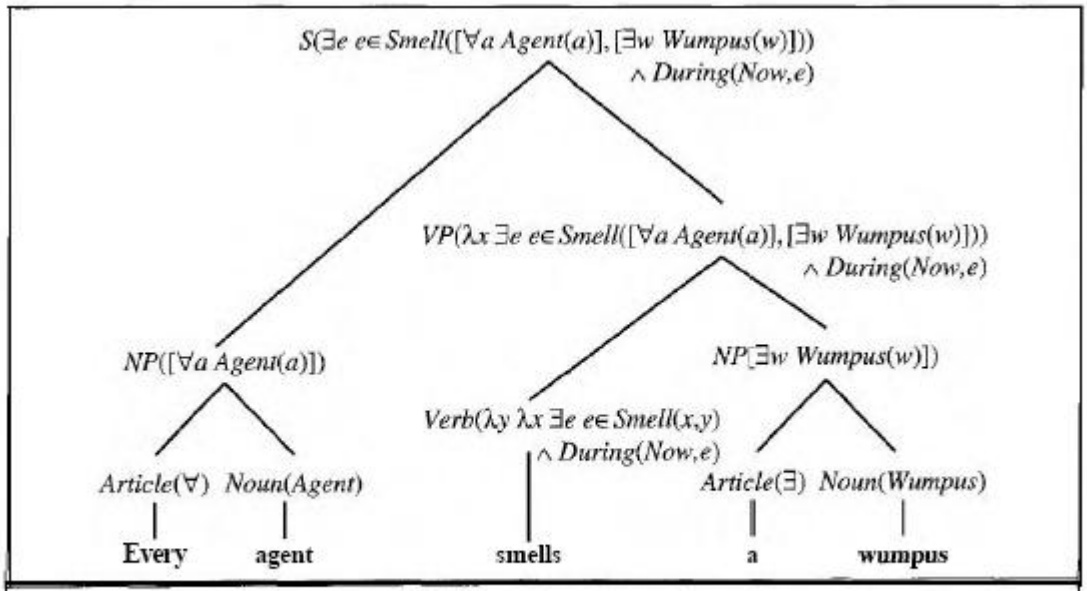



**Figure 22.19** Parse tree for the sentence "Every agent smells a wumpus," showing both syntactic structure and semantic interpretations.

**(15) Define lexical,syntactic, and semantic ambiguity.**

**Lexical ambiguity**, in which a word has more than one meaning. Lexical ambiguity is quite common; "back" can be an adverb (go back), an adjective (back door), a noun (the back of the room) or a verb (back up your files). "Jack" can be a name, a noun (a playing card, a six-pointed metal game piece, a nautical flag, a fish, a male donkey, a socket, or a device for raising heavy objects), or a verb (to jack up a car, to hunt with a light, or to hit a baseball hard).

**Syntactic ambiguity** (also known as structural ambiguity) can occur with or without lexical ambiguity. For example, the string "I smelled a wumpus in *2,2"* has two parses: one where the prepositional phrase "in 2,2" modifies the noun and one where it modifies the verb. The syntactic ambiguity leads to a semantic ambiguity, because one parse means that the wumpus is in 2,2 and the other means that a stench is in *2,2.* In this case, getting the wrong interpretation could be a deadly mistake.

**Semantic ambiguity** can occur even in phrases with no lexical or syntactic ambiguity. For example, the noun phrase "cat person" can be someone who likes felines or the lead of

the movie Attack of the Cat People. A "coast road" can be a road that follows the coast or one that leads to it.

**(16) What is disambiguation?**

**Disambiguation**

*Disambiguation is a question of diagnosis.* The speaker's intent to communicate is an unobserved cause of the words in the utterance, and the hearer's job is to work backwards from the words and from knowledge of the situation to recover the most likely intent of the speaker.. Some sort of preference is needed because syntactic and semantic interpretation rules alone cannot identify a unique correct interpretation of a phrase or sentence. So we divide the work: syntactic and semantic interpretation is responsible for enumerating a set of candidate interpretations, and the disambiguation process chooses the best one.

**(17) What is discourse?**

**A discourse** is any string of language-usually one that is more than one sentence long. Textbooks, novels, weather reports and conversations are all discourses. So far we have largely ignored the problems of discourse, preferring to dissect language into individual sentences that can be studied *in vitro.* We will look at two particular subproblems: reference resolution and coherence.

**Reference resolution**

**Reference resolution** is the interpretation of a pronoun or a definite noun phrase that refers to an object in the world. The resolution is based on knowledge of the world and of the previous parts of the discourse. Consider the passage "John flagged down the waiter. He ordered a hani sandwich."

To understand that "he" in the second sentence refers to John, we need to have understood that the first sentence mentions two people and that John is playing the role of a customer and hence is likely to order, whereas the waiter is not.

**The structure of coherent discourse**

If you open up this book to 10 random pages, and copy down the first sentence from each page. The result is bound to be incoherent. Similarly, if you take a coherent 10-sentence passage and permute the sentences, the result is incoherent. This demonstrates that sentences in natural language discourse are quite different from sentences in logic. In logic, if we TELL sentences A, *B* and *C* to a knowledge base, in any order, we endl up with Ithe conjunction *A* A *B A C.* In natural language, sentence order matters; consider th~ed ifference between "Go two blocks. Turn right." and "Turn right. Go two blocks."

**(18) What is grammar induction?**

**Grammar induction** is the task of learning a grammar from data. It is an obvious task to attempt, given that it has proven to be so difficult to construct a grammar by hand and that billions of example utterances are available for free on the Internet. It is a difficult task because the space of possible grammars is infinite and because verifying that a given grammar generates a set of sentences is computationally expensive.

**Grammar induction** can learn a grammar from examples, although there are limitations on how well the grammar will generalize.

**(19) What is information retrieval?**

**Information retrieval** is the task of finding documents that are relevant to a user's need for information. The best-known examples of information retrieval systems are search engines on the World Wide Web. A Web user can type a query such as [AI book] into a search engine and see a list of relevant pages. An information retrieval (henceforth IR) system can be characterized by:

**Powered By www.technoscriptz.com**

Notesengine.com

1) **A document collection.** Each system must decide what it wants to treat as a document: a paragraph, a page, or a multi-page text.

**2) A query posed in a query language.** The 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 specify a phrase of words that must be adjacent, as in ["AI book"]; it can contain Boolean operators as in **[AI** *AND* book]; it can include non-Boolean operators such as [AI book SITE:www.aaai.org].

**3) A result set.** This is the subset of documents that the IR system judges to be **relevant** to the query. By relevant, we mean likely to be of use to the person who asked 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 document titles or as complex as a rotating color map of the result set projected onto a three dimensional space.

**(20) What is clustering?**

Clustering is an unsupervised learning problem. Unsupervised clustering is the problem of discerning multiple categories in a collection of objects. The problem is unsupervised because the category labels are not given.

Examples

We are familiar with terms such as "red giant" and "white dwarf",but the stars do not carry these labels – astronomers had to perform unsupervised clustering to identify these categories.

**(21) What is agglomerative clustering?**

Agglomerative clustering creates a tree of clusters going all the way down to the individual documents. We begin by considering each document as a separate cluster. Then we find the two clusters that are closest to each other according to some distance measure and merge these clusters into one. The distance measure between two clusters can be the distance to the median of the cluster. Agglomerative clustering takes time $O(n^2)$,where n is the number of documents.

**(22) What is K-means clustering?**

**K-means clustering** creates a flat set of exactly k-categories.It works as follows :

**i)** Pick K documents at random to represent the K categories.

**ii)** Assign every document to the closest category.

**iii)** Compute the mean of each cluster and use K-means to represent the new value of the K categories.

iv) Repeat the steps (ii) and (iii) until convergence.

K-means takes O(n)

**(23) What is information extraction?**

**Information extraction** is the process of creating database entries by skimming a text and looking for occurrences of a particular **class of object or event** and for **relationships** among those **objects and events**.

We could be trying to extract instances of addresses from web pages, with database fields for street, city, state, and zip code; or instances of storms from weather reports, with fields for temperature, wind speed, and precipitation. Information extraction systems are mid-way between information retrieval systems and full-text parsers, in that they need to do more than consider a document as a bag of words, but less than completely analyze every sentence.

The simplest type of information extraction system is called an ***attribute-based*** **system** because it assumes that the entire text refers to a single object and the task is to extract attributes of that object.

Notesengine.com

For example, the problem of extracting from the text "17in SXGA Monitor for only $249.99" the database relations given by

*3 m m* E *ComputerMonitors A Size(m, Inches(l7)) A Price(m, $(249.99))*
*A Resolution(m, 1280 x 1024)* .

Some of this information can be handled with the help of regular expressions, which define a regular grammar in a single text string. Regular expressions are used in Unix commands such as grep, in programming languages such as Perl, and in word processors such as Microsoft Word.

**(24) What is machine translation? Explain different types.**

**Machine translation** is the automatic translation of text from one natural language (the source) to another (the target). This process has proven to be useful for a number of tasks, including the following:

**1. Rough translation,** in which the goal is just to get the gist of a passage. Ungrammatical and inelegant sentences are tolerated as long as the meaning is clear. For example, in Web surfing, a user is often happy with a rough translation of a foreign web page. Sometimes a monolingual human can post-edit the output without having to read the source. This type of machine-assisted translation saves money because such editors can be paid less than bilingual translators.

2**. Restricted-source translation**, in which the subject matter and format of the source text are severely limited. One of the most successful examples is the TAUM-METEO system, which translates weather reports from English to French. It works because the language used in weather reports is highly stylized and regular.

3. **Pre-edited translation**, in which a human preedits the source document to make it conform to a restricted subset of English (or whatever the original language is) before machine translation. This approach is particularly cost-effective when there is a need to translate one document into many languages, as is the case for legal documents in the European Community or for companies that sell the same product in many countries. Restricted languages are sometimes called "Caterpillar English," because Caterpillar Corp. was the first firm to try writing its manuals in this form. Xerox defined a language for its maintenance manuals which was simple enough that it could be translated by machine into all the languages Xerox deals with. As an added benefit, the original English manuals became clearer as well.

4**. Literary translation**, in which all the nuances of the source text are preserved. This is currently beyond the state of the art for machine translation.

**(25) Draw a schematic for a machine translation system for English to French.**

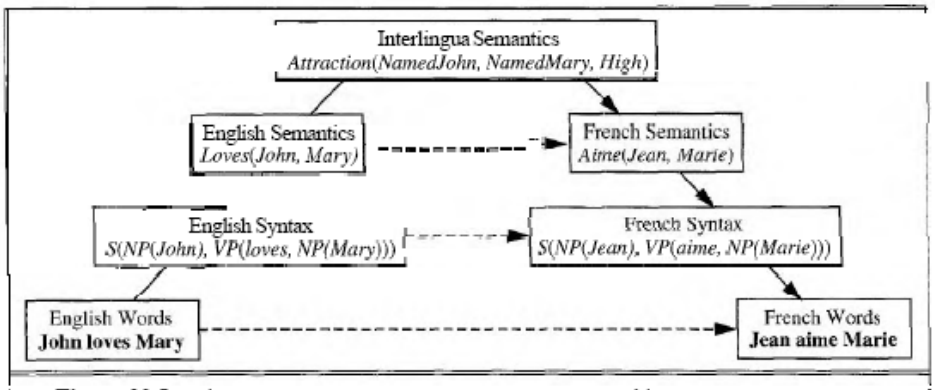



**Figure 23.5** A schematic diagram of the choices for a machine translation system. We start with English text at the top. An interlingua-based system follows the solid lines, parsing English first into a syntactic form, then into a semantic representation and an interlingua representation, and then through generation to a semantic, syntactic, and lexical form in French. A transfer-based system uses the dashed lines as a shortcut. Different systems make the transfer at different points; some make it at multiple points.

**Powered By www.technoscriptz.com**

# UNIT-V Question and Answers

**(1) Define communication.**

**Communication** is the intentional exchange of information brought about by the production and perception of **signs** drawn from a shared system of conventional signs. Most animals use signs to represent important messages: food here, predator nearby etc. In a partially observable world, communication can help agents be successful because they can learn information that is observed or inferred by others.

**(2) What is speech act?**

What sets humans apart from other animals is the complex system of structured messages known as **language** that enables us to communicate most of what we know about the world. This is known as speech act.

**Speaker, hearer,** and **utterance** are generic terms referring to any mode of communication. The term **word** is used to refer to any kind of conventional communicative sign.

(3) **What are the capabilities gained by an agent from speech act?**.

- **Query** other agents about particular aspects of the world. This is typically done by asking questions: *Have you smelled the wumpus anywhere?*
- **Inform** each other about the world. This is done by making representative statements: *There's a breeze here in 3 4.* Answering a question is another Pund of informing.
- **Request** other agents to perform actions: *Please help me carry the gold.* Sometimes **indirect speech act** (a request in the form of a statement or question) is considered more polite: *I could use some help carrying this.* An agent with authority can give commands *(Alpha go right; Bravo and Charlie go lejl),* and an agent with power can make a threat *(Give me the gold, or else).* Together, these kinds of speech acts are called **directives.**
- **Acknowledge** requests: **OK.**
- **Promise** or commit to a plan: *I'll shoot the wumpus; you grab the gold.*

**(4) Define formal language.**

**A formal language** is defined as a (possibly infinite) set of' **strings.** Each string is a concatenation of **terminal symbols,** sometimes called words. For example, in the language of first-order logic, the terminal symbols include A and P, and a typical string is "P A Q." . Formal languages such as first-order logic and Java have strict mathematical definitions. This is in contrast to **natural languages,** such as Chinese, Danish, and English, that have no strict definition but are used by a community.

**(5) Define a grammar.**

A **grammar** is a finite set of rules that specifies a language. Formal languages always have an official grammar, specified in manuals or books. Natural languages have no official grammar, but linguists strive to discover properties of the language by a process of scientific inquiry and then to codify their discoveries in a grammar.

**(6) What are the component steps of communication? Explain with an example.**

**The component steps of communication**



Notesengine.com

A typical communication episode, in which speaker S wants to inform hearer H about proposition P using words W, is composed of seven processes:

**1) Intention.** Somehow, speaker S decides that there is some proposition P that is worth saying to hearer H. For our example, the speaker has the intention of having the hearer know that the wumpus is no longer alive.

**2) Generation.** The speaker plans how to turn the proposition P into an utterance that makes it likely that the hearer, upon perceiving the utterance in the current situation, can infer the meaning P (or something close to it). Assume that the speaker is able to come up with the words "The wumpus is dead," and call this *W*.

**3) Synthesis.** The speaker produces the physical realization *W'* of the words *W*. This can be via ink on paper, vibrations in air, or some other medium. In Figure 22.1, we show the agent synthesizing a string of sounds *W'* written in the phonetic alphabet defined on page 569: "[thaxwahmpaxsihzdehd]." The words are run together; this is typical of quickly spoken speech.

**4) Perception.** H perceives the physical realization W' as *Wi* and decodes it as the words *W2*. When the medium is speech, the perception step is called **speech recognition;** when it is printing, it is called **optical character recognition.**

**5) Analysis.** H infers that *W2* has possible meanings *PI, . . . , P,*.

We divide analysis into three main parts:

a) **syntactic interpretation (or parsing**),

b) **Semantic interpretation,** and

c)**Pragmatic interpretation**.

**Parsing** is the process of building a **parse tree** for an input string, as shown in Figure 22.1. The interior nodes of the parse tree represent phrases and the leaf nodes represent words.

**Semantic interpretation** is the process of extracting the meaning of an utterance as an expression in some representation language. Figure 22.1 shows two possible semantic interpretations: that the wumpus is not alive and that it is tired (a colloquial meaning of dead). Utterances with several possible interpretations are said to be **ambiguous.**

**Pragmatic interpretation** takes into account the fact that the same words can have different meanings in different situations.

**6) Disambiguation. H** infers that *S* intended to convey P, (where ideally *P,* = P).

Most speakers are not intentionally ambiguous, but most utterances have several feasible interpretations.. Analysis generates possible interpretations;if more than one interpretation is found, then disambiguation chooses the one that is best.

7) **Incorporation. H** decides to believe *P,* (or not). A totally naive agent might believe everything it hears, but a sophisticated agent treats the speech act as evidence for P,, not confirmation of it.

Putting it all together, we get the agent program shown in Figure 22.2. Here the agent acts as a robot slave that can be commanded by a master. On each turn, the slave will answer a question or obey a command if the master has made one, and it will believe any statements made by the master. It will also comment (once) on the current situation if it has nothing more pressing to do, and it will plan its own action if left aloine.

Here is a typical dialog:

ROBOT SLAVE MASTER
I feel a breeze. Go to 12.
Nothing is here. Go north.
I feel a breeze and I smell a stench
and I see a glitter. Grab the gold.

Notesengine.com

**Powered By www.technoscriptz.com**

Fig 22.1 shows the seven processes involved in communication,using the example sentence “The wumpus is dead”.

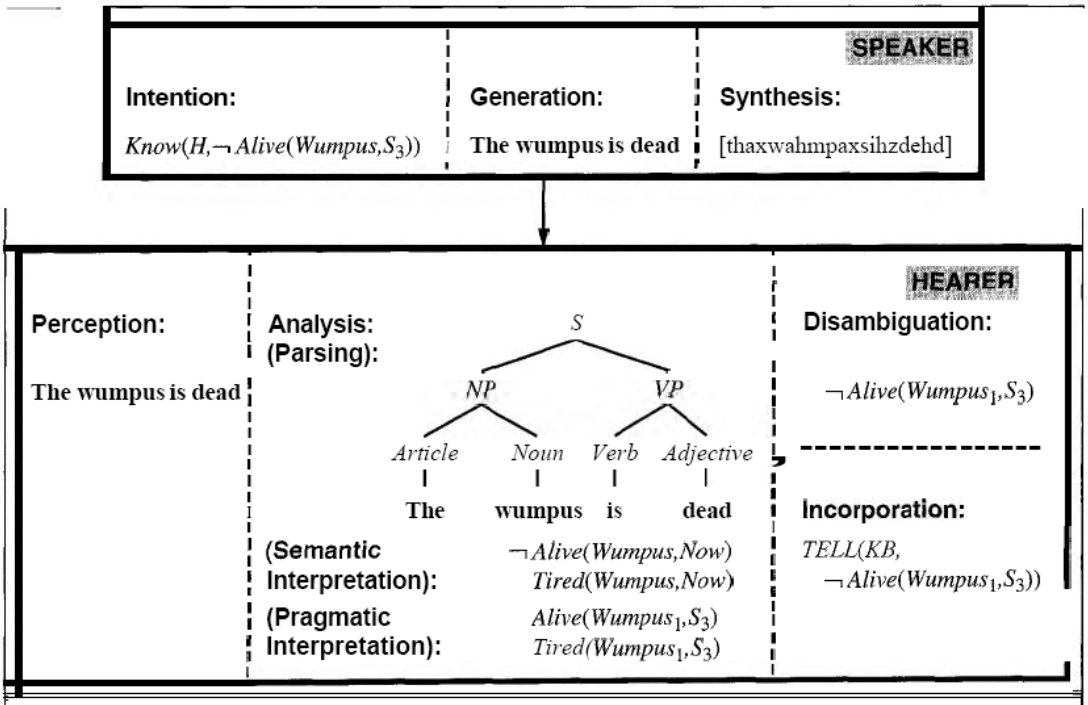



**Figure 22.1** *Seven processes involved in communication, using the example sentence "The wumpus is dead."*

**(7) Define a Lexicon and grammar for language consisting of a small fragment of English.**

### The Lexicon of $\mathcal{E}_0$

First we define the **lexicon,** or list of allowable words. The words are grouped into the categories or parts of speech familiar to dictionary users: nouns, pronouns, and names to denote

things, verbs to denote events, adjectives to modify nouns, and adverbs to modify verbs. Categories that are perhaps less familiar to some readers are articles (such as the), prepositions (in), and conjunctions (and). Figure 22.3 shows a small lexicon.

| | | |
|---|---|---|
| *Noun* | → | stench \| breeze \| glitter \| **nothing** \| agent |
| | | \| wumpus \| pit \| **pits**\| **gold** \| east \| ... |
| *Verb* | → | is \| see \| smell \| shoot \| feel \| stinks |
| | | \| go \| grab \| carry \| kill \| turn \| ... |
| *Adjective* | → | right \| left \| east \| dead \| back \| smelly \| .. |
| *Adverb* | → | here \| there \| nearby \| ahead |
| | | \| right \| left \| east \| south \| **back** \| ... |
| *Pronoun* | → | me \| you \| I \| it \| ... |
| *Name* | → | John \| Mary \| Boston \| **Aristotle** \| ... |
| *Article* | → | the \| a \| an \| ... |
| *Preposition* | → | to \| in \| on \| near \| ... |
| *Conjunction* | → | and \| or \| but \| ... |
| *Digit* | → | **0**\| **1**\| **2**\| **3**\| **4**\| **5**\| **6**\| **7**\| **8**\| **9** |

**Figure 22.3** The lexicon for $\mathcal{E}_0$.

**Powered By www.technoscriptz.com**

## The Grammar of $\mathcal{E}_0$

The next step is to combine the words into phrases. We will use five nonterminal symbols to define the different kinds of phrases: sentence *(S)*, noun phrase (NP), verb phrase (VP), prepositional phrase (PP), and relative clause e el clause).' Figure 22.4 shows a grammar for l o , with an example for each rewrite rule. lo generates good English sentences such as the following:

John is in the pit
The wumpus that stinks is in 2 2

| | | | |
|---|---|---|---|
| *S* | → | *NP VP* | *I* + feel a breeze |
| | \| | *S Conjunction S* | I feel a breeze + and + I smell a wumpus |
| *NP* | → | *Pronoun* | *I* |
| | \| | *Name* | John |
| | \| | *Noun* | pits |
| | \| | *Article Noun* | the + wumpus |
| | \| | *Digit Digit* | 3 4 |
| | \| | *NP PP* | the wumpus + to the east |
| | \| | *NP RelClause* | the wumpus + that is smelly |
| *VP* | → | *Verb* | stinks |
| | \| | *VP NP* | feel + a breeze: |
| | \| | *VP Adjective* | is + smelly |
| | \| | *VP PP* | turn + to the east |
| | \| | *VP Adverb* | go + ahead |
| *PP* | → | *Preposition NP* | to + the east |
| *RelClause* | → | that *VP* | that + is smelly |

**Figure 22.4** The grammar for $\mathcal{E}_0$, with example phrases for each rule.

**(8) What is parsing? Explain the top down parsing method.**

Parsing is defined as the process of finding a **parse tree** for a given input string. That is, a call to the parsing function PARSE, such as

PARSE("the wumpus is dead", $\mathcal{E}_0$, $S$)

should return a parse tree with root S whose leaves are "the wumpus is dead" and whose internal nodes are nonterminal symbols from the grammar Eo.

*Parsing can be seen as a process of searching for a parse tree.*

There are two extreme ways of specifying the search space (and many variants in between). First, we can start with the *S* symbol and search for a tree that has the words as its leaves. This is called **top-down parsing**

Second, we could start with the words and search for a tree with root S. This is called **bottom-up parsing.**

Top-down parsing can be precisely defined as a search problem as follows:

- The initial state is a parse tree consisting of the root S and unknown children: [S: ?I. In general, each state in the search space is a parse tree.

The successor function selects the leftmost node in the tree with unknown children. It

then looks in the grammar for rules that have the root label of the node on the left-hand side. For each such rule, it creates a successor state where the ? is replaced by a list corresponding to the right-hand side of the rule..

**(9) Formulate the bottom-up parsing as a search problem.**

The formulation of bottom-up parsing as a search is as follows:

The **initial state** is a list of the words in the input string, each viewed as a parse tree that is just a single leaf node-for example; **[the, wumlpus, is, dead].** In general, each state in the search space is a list of parse trees.

The **successor function** looks at every position ***i*** in the list of trees and at every righthand side of a rule in the grammar. If the subsequence of the list of trees starting at *i* matches the right-hand side, then the subsequence is replaced by a new tree whose category is the left-hand side of the rule and whose children are the subsequence. By "matches," we mean that the category of the node is the same as the element in the righthand side. For example, the rule *Article* + **the** matches the subsequence consisting of the first node in the list **[the, wumpus, is, dead],** so a successor state would be *[[Article:***the], wumpus, is, dead].**

The **goal test** checks for a state consisting of a single tree with root S.

See Figure 22.5 for an example of bottom-up parsing.

| *step* | *list of nodes* | *subsequence* | *rule* |
|---|---|---|---|
| INIT | **the wumpus is dead** | **the** | Article → **the** |
| 2 | *Article* **wumpus is dead** | **wumpus** | Noun → **wumpus** |
| 3 | *Article Noun* **is dead** | *Article Noun* | *NP → Article Noun* |
| 4 | *NP* **is dead** | **is** | *Verb* → **is** |
| 5 | *NP Verb* **dead** | **dead** | *Adjective* → **dead** |
| 6 | **NP** *Verb Adjective* | *Verb* | *VP → Verb* |
| 7 | *NP VP Adjective* | *VP Adjective* | *VP → VP Adjective* |
| 8 | *NP VP* | *NP VP* | *S → NP VP* |
| GOAL | S | | |

**Figure 22.5** Trace of a bottom up parse on the string "The wumpus is dead." We start with a list of nodes consisting of words. Then we replace subsequences that match the right-hand side of a rule with a new node whose root is the left-hand side. For example, in the third line the Article and Noun nodes are replaced by an NP node that has those two nodes as children. The top-down parse would produce a similar trace, but in the opposite direction.

**(10) What is dynamic programming?**

Forward Chaining on graph search problem is an example of dynamic programming. Solutions to the sub problems are constructed incrementally from those of smaller sub problems and are cached to avoid recomputation.

**(11) Construct a parse tree for "You give me the gold" showing the sub categories of the verb and verb phrase.**

**Powered By www.technoscriptz.com**

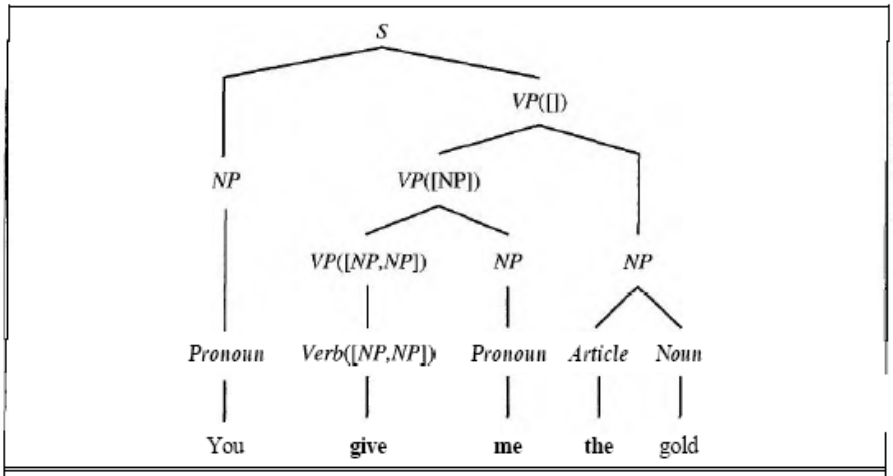



Figure 22.13 Parse tree for "You give me the gold" showing subcategorization of the verb and verb phrase.

**(12) What is semantic interpretation? Give an example.**

Semantic interpretation is the process of associating an FOL expression with a phrase.

$$
\begin{aligned}
&Exp(x) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\}\\
&Exp(x) \rightarrow (\ Exp(x)\ )\\
&Exp(x) \rightarrow Number(x)\\
&Number(x) \rightarrow Digit(x)\\
&Number(x) \rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\}\\
&Digit(x) \rightarrow x\ \{0 \le x \le 9\}\\
&Operator(x) \rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

**Figure 22.14** A grammar for arithmetic expressions, augmented with semantics. Each variable $x_i$ represents the semantics of a constituent. Note the use of the $\{test\}$ notation to define logical predicates that must be satisfied, but that are not constituents.

**(13) Construct a grammar and sentence for "John loves Mary"**

$$
\begin{aligned}
&S(rel(obj)) \rightarrow NP(obj)\ VP(rel)\\
&VP(rel(obj)) \rightarrow Verb(rel)\ NP(obj)\\
&NP(obj) \rightarrow Name(obj)\\
&\\
&Name(John) \rightarrow \textbf{John}\\
&Name(Mary) \rightarrow \textbf{Mary}\\
&Verb(\lambda y\ \lambda x\ Loves(x, y)) \rightarrow \textbf{loves}
\end{aligned}
$$

**Figure 22.16** A grammar that can derive a parse tree and semantic interpretationfor "John loves Mary" (and three other sentences). Each category is augmented with a single argument representing the semantics.

**Powered By www.technoscriptz.com**

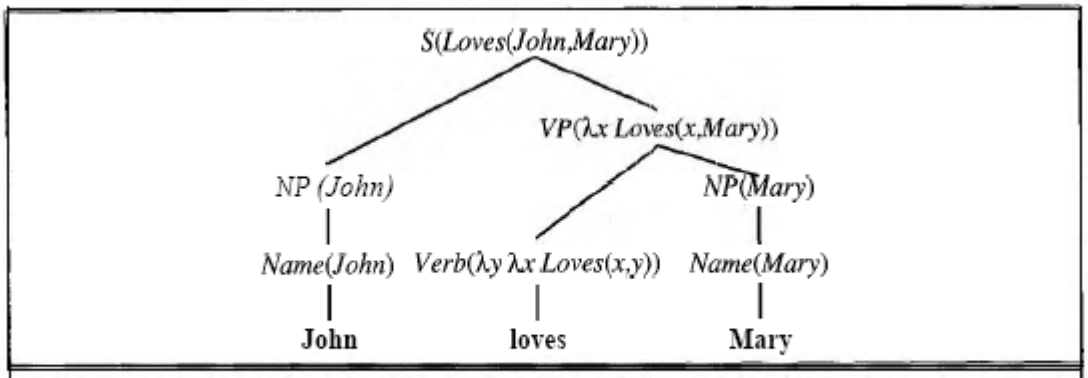



**Figure 22.17** A parse tree with semantic interpretations for the string "John loves Mary".

**(14) Construct a parse tree for the sentence "Every agent smells a Wumpus"**

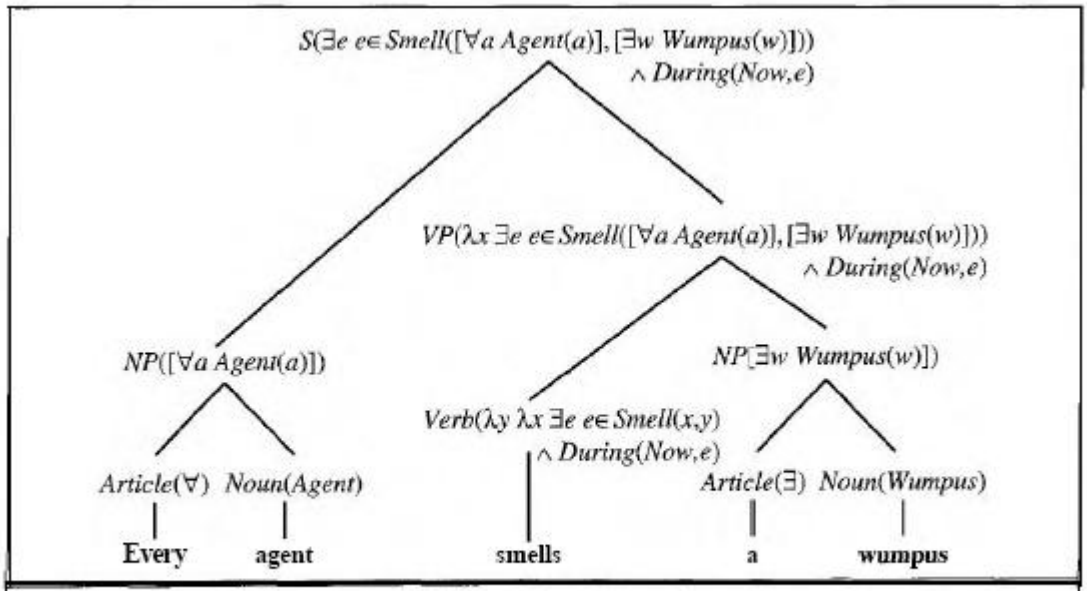



**Figure 22.19** Parse tree for the sentence "Every agent smells a wumpus," showing both syntactic structure and semantic interpretations.

**(15) Define lexical,syntactic, and semantic ambiguity.**

**Lexical ambiguity**, in which a word has more than one meaning. Lexical ambiguity is quite common; "back" can be an adverb (go back), an adjective (back door), a noun (the back of the room) or a verb (back up your files). "Jack" can be a name, a noun (a playing card, a six-pointed metal game piece, a nautical flag, a fish, a male donkey, a socket, or a device for raising heavy objects), or a verb (to jack up a car, to hunt with a light, or to hit a baseball hard).

**Syntactic ambiguity** (also known as structural ambiguity) can occur with or without lexical ambiguity. For example, the string "I smelled a wumpus in *2,2"* has two parses: one where the prepositional phrase "in 2,2" modifies the noun and one where it modifies the verb. The syntactic ambiguity leads to a semantic ambiguity, because one parse means that the wumpus is in 2,2 and the other means that a stench is in *2,2.* In this case, getting the wrong interpretation could be a deadly mistake.

**Semantic ambiguity** can occur even in phrases with no lexical or syntactic ambiguity. For example, the noun phrase "cat person" can be someone who likes felines or the lead of

the movie Attack of the Cat People. A "coast road" can be a road that follows the coast or one that leads to it.

**(16) What is disambiguation?**

**Disambiguation**

*Disambiguation is a question of diagnosis.* The speaker's intent to communicate is an unobserved cause of the words in the utterance, and the hearer's job is to work backwards from the words and from knowledge of the situation to recover the most likely intent of the speaker.. Some sort of preference is needed because syntactic and semantic interpretation rules alone cannot identify a unique correct interpretation of a phrase or sentence. So we divide the work: syntactic and semantic interpretation is responsible for enumerating a set of candidate interpretations, and the disambiguation process chooses the best one.

**(17) What is discourse?**

**A discourse** is any string of language-usually one that is more than one sentence long. Textbooks, novels, weather reports and conversations are all discourses. So far we have largely ignored the problems of discourse, preferring to dissect language into individual sentences that can be studied *in vitro.* We will look at two particular subproblems: reference resolution and coherence.

**Reference resolution**

**Reference resolution** is the interpretation of a pronoun or a definite noun phrase that refers to an object in the world. The resolution is based on knowledge of the world and of the previous parts of the discourse. Consider the passage "John flagged down the waiter. He ordered a hani sandwich."

To understand that "he" in the second sentence refers to John, we need to have understood that the first sentence mentions two people and that John is playing the role of a customer and hence is likely to order, whereas the waiter is not.

**The structure of coherent discourse**

If you open up this book to 10 random pages, and copy down the first sentence from each page. The result is bound to be incoherent. Similarly, if you take a coherent 10-sentence passage and permute the sentences, the result is incoherent. This demonstrates that sentences in natural language discourse are quite different from sentences in logic. In logic, if we TELL sentences A, *B* and *C* to a knowledge base, in any order, we endl up with Ithe conjunction *A* A *B A C.* In natural language, sentence order matters; consider th~ed ifference between "Go two blocks. Turn right." and "Turn right. Go two blocks."

**(18) What is grammar induction?**

**Grammar induction** is the task of learning a grammar from data. It is an obvious task to attempt, given that it has proven to be so difficult to construct a grammar by hand and that billions of example utterances are available for free on the Internet. It is a difficult task because the space of possible grammars is infinite and because verifying that a given grammar generates a set of sentences is computationally expensive.

**Grammar induction** can learn a grammar from examples, although there are limitations on how well the grammar will generalize.

**(19) What is information retrieval?**

**Information retrieval** is the task of finding documents that are relevant to a user's need for information. The best-known examples of information retrieval systems are search engines on the World Wide Web. A Web user can type a query such as [AI book] into a search engine and see a list of relevant pages. An information retrieval (henceforth IR) system can be characterized by:

**Powered By www.technoscriptz.com**

Notesengine.com

1) **A document collection.** Each system must decide what it wants to treat as a document: a paragraph, a page, or a multi-page text.

**2) A query posed in a query language.** The 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 specify a phrase of words that must be adjacent, as in ["AI book"]; it can contain Boolean operators as in **[AI** *AND* book]; it can include non-Boolean operators such as [AI book SITE:www.aaai.org].

**3) A result set.** This is the subset of documents that the IR system judges to be **relevant** to the query. By relevant, we mean likely to be of use to the person who asked 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 document titles or as complex as a rotating color map of the result set projected onto a three dimensional space.

**(20) What is clustering?**

Clustering is an unsupervised learning problem. Unsupervised clustering is the problem of discerning multiple categories in a collection of objects. The problem is unsupervised because the category labels are not given.

Examples

We are familiar with terms such as "red giant" and "white dwarf",but the stars do not carry these labels – astronomers had to perform unsupervised clustering to identify these categories.

**(21) What is agglomerative clustering?**

Agglomerative clustering creates a tree of clusters going all the way down to the individual documents. We begin by considering each document as a separate cluster. Then we find the two clusters that are closest to each other according to some distance measure and merge these clusters into one. The distance measure between two clusters can be the distance to the median of the cluster. Agglomerative clustering takes time $O(n^2)$,where n is the number of documents.

**(22) What is K-means clustering?**

**K-means clustering** creates a flat set of exactly k-categories.It works as follows :

**i)** Pick K documents at random to represent the K categories.

**ii)** Assign every document to the closest category.

**iii)** Compute the mean of each cluster and use K-means to represent the new value of the K categories.

iv) Repeat the steps (ii) and (iii) until convergence.

K-means takes O(n)

**(23) What is information extraction?**

**Information extraction** is the process of creating database entries by skimming a text and looking for occurrences of a particular **class of object or event** and for **relationships** among those **objects and events**.

We could be trying to extract instances of addresses from web pages, with database fields for street, city, state, and zip code; or instances of storms from weather reports, with fields for temperature, wind speed, and precipitation. Information extraction systems are mid-way between information retrieval systems and full-text parsers, in that they need to do more than consider a document as a bag of words, but less than completely analyze every sentence.

The simplest type of information extraction system is called an ***attribute-based*** **system** because it assumes that the entire text refers to a single object and the task is to extract attributes of that object.

Notesengine.com

For example, the problem of extracting from the text "17in SXGA Monitor for only $249.99" the database relations given by

*3 m m* E *ComputerMonitors A Size(m, Inches(l7)) A Price(m, $(249.99))*
*A Resolution(m, 1280 x 1024)* .

Some of this information can be handled with the help of regular expressions, which define a regular grammar in a single text string. Regular expressions are used in Unix commands such as grep, in programming languages such as Perl, and in word processors such as Microsoft Word.

**(24) What is machine translation? Explain different types.**

**Machine translation** is the automatic translation of text from one natural language (the source) to another (the target). This process has proven to be useful for a number of tasks, including the following:

**1. Rough translation,** in which the goal is just to get the gist of a passage. Ungrammatical and inelegant sentences are tolerated as long as the meaning is clear. For example, in Web surfing, a user is often happy with a rough translation of a foreign web page. Sometimes a monolingual human can post-edit the output without having to read the source. This type of machine-assisted translation saves money because such editors can be paid less than bilingual translators.

2**. Restricted-source translation**, in which the subject matter and format of the source text are severely limited. One of the most successful examples is the TAUM-METEO system, which translates weather reports from English to French. It works because the language used in weather reports is highly stylized and regular.

3. **Pre-edited translation**, in which a human preedits the source document to make it conform to a restricted subset of English (or whatever the original language is) before machine translation. This approach is particularly cost-effective when there is a need to translate one document into many languages, as is the case for legal documents in the European Community or for companies that sell the same product in many countries. Restricted languages are sometimes called "Caterpillar English," because Caterpillar Corp. was the first firm to try writing its manuals in this form. Xerox defined a language for its maintenance manuals which was simple enough that it could be translated by machine into all the languages Xerox deals with. As an added benefit, the original English manuals became clearer as well.

4**. Literary translation**, in which all the nuances of the source text are preserved. This is currently beyond the state of the art for machine translation.

**(25) Draw a schematic for a machine translation system for English to French.**

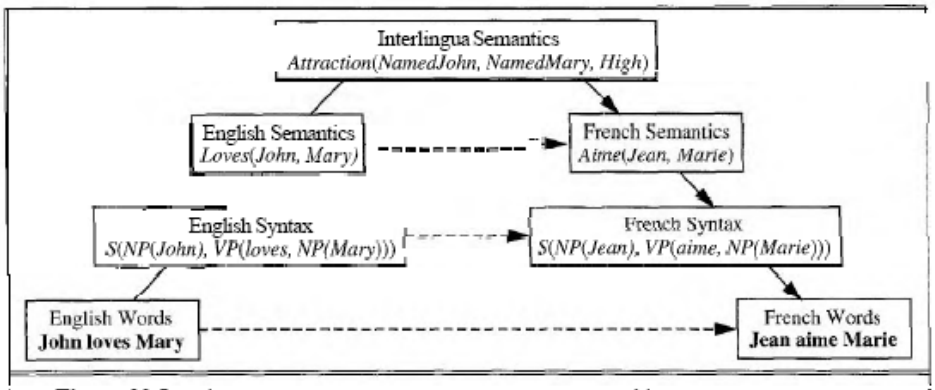



**Figure 23.5** A schematic diagram of the choices for a machine translation system. We start with English text at the top. An interlingua-based system follows the solid lines, parsing English first into a syntactic form, then into a semantic representation and an interlingua representation, and then through generation to a semantic, syntactic, and lexical form in French. A transfer-based system uses the dashed lines as a shortcut. Different systems make the transfer at different points; some make it at multiple points.

**Powered By www.technoscriptz.com**