BCS-404 ARTIFICIAL INTELLIGENCE (3-1-0) Cr.-04

Module - I

Formalized symbolic logic: Propositional logic-first order predicate logic, wff conversion to clausal form, inference rules, the resolution principle, Dealing with inconsistencies and uncertainties, fuzzy logic.

Module - II

Probabilistic Reasoning Structured knowledge, graphs, frames and related structures, Knowledge organization and manipulation.

Module – III

Matching Techniques, Knowledge organizations, Management.

Module - IV

Natural Language processing, Pattern recognition, expert systems.

Text Book:

1. Artificial Intelligence, Dan W Patterson, Prentice Hall of India (1999) Chapter-4, 5,7,9,10,11,12,13,15.

Reference Books:

1. Artificial Intelligence, Nils J.Nilsson, ELSEVIER.
2. E.Rich and K.Knight, Artificial Intelligence, - TMH

Overview of Artificial Intelligence

What is AI ?

- **A**rtificial Intelligence (AI) is a branch of *Science* which deals with helping machines find solutions to complex problems in a more human-like fashion.
- This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way.
- A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behavior appears
- Artificial intelligence can be viewed from a variety of perspectives.
    - ✓ From the perspective of **intelligence** artificial intelligence is making machines "intelligent" -- acting as we would expect people to act.
        - o The inability to distinguish computer responses from human responses is called the Turing test.
        - o Intelligence requires knowledge
        - o Expert problem solving - restricting domain to allow including significant relevant knowledge
    - ✓ From a **business** perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.
    - ✓ From a **programming** perspective, AI includes the study of symbolic programming, problem solving, and search.
        - o Typically AI programs focus on symbols rather than numeric processing.
        - o Problem solving - achieve goals.
        - o Search - seldom access a solution directly. Search may include a variety of techniques.
        - o AI programming languages include:

– LISP, developed in the 1950s, is the early programming language strongly associated with AI. LISP is a functional programming language with procedural extensions. LISP (LISt Processor) was specifically designed for processing heterogeneous lists -- typically a list of symbols. Features of LISP are run- time type checking, higher order functions (functions that have other functions as parameters), automatic memory management (garbage collection) and an interactive environment.

– The second language strongly associated with AI is PROLOG. PROLOG was developed in the 1970s. PROLOG is based on first order logic. PROLOG is declarative in nature and has facilities for explicitly limiting the search space.

– Object-oriented languages are a class of languages more recently used for AI programming. Important features of object-oriented languages include: concepts of objects and messages, objects bundle data and methods for manipulating the data, sender specifies what is to be done receiver decides how to do it, inheritance (object hierarchy where objects inherit the attributes of the more general class of objects). Examples of object-oriented languages are Smalltalk, Objective C, C++. Object oriented extensions to LISP (CLOS - Common LISP Object System) and PROLOG (L&O - Logic & Objects) are also used.

- Artificial Intelligence is a new electronic machine that stores large amount of information and process it at very high speed
- The computer is interrogated by a human via a teletype It passes if the human cannot tell if there is a computer or human at the other end
- The ability to solve problems
- It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence

Importance of AI

- **Game Playing**

  You can buy machines that can play master level chess for a few hundred dollars. There is some AI in them, but they play well against people mainly through brute force computation--looking at hundreds of thousands of positions. To beat a world champion by brute force and known reliable heuristics requires being able to look at 200 million positions per second.

- **Speech Recognition**

In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names. It is quite convenient. On the other hand, while it is possible to instruct some computers using speech, most users have gone back to the keyboard and the mouse as still more convenient.

- **Understanding Natural Language**

  Just getting a sequence of words into a computer is not enough. Parsing sentences is not enough either. The computer has to be provided with an understanding of the domain the text is about, and this is presently possible only for very limited domains.

- **Computer Vision**

  The world is composed of three-dimensional objects, but the inputs to the human eye and computers' TV cameras are two dimensional. Some useful programs can work solely in two dimensions, but full computer vision requires partial three-dimensional information that is not just a set of two-dimensional views. At present there are only limited ways of representing three-dimensional information directly, and they are not as good as what humans evidently use.

- **Expert Systems**

  A ``knowledge engineer'' interviews experts in a certain domain and tries to embody their knowledge in a computer program for carrying out some task. How well this works depends on whether the intellectual mechanisms required for the task are within the present state of AI. When this turned out not to be so, there were many disappointing results. One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students or practicing doctors, provided its limitations were observed. Namely, its ontology included bacteria, symptoms, and treatments and did not include patients, doctors, hospitals, death, recovery, and events occurring in time. Its interactions depended on a single patient being considered. Since the experts consulted by the knowledge engineers knew about patients, doctors, death, recovery, etc., it is clear that the knowledge engineers forced what the experts told them into a predetermined framework. The usefulness of current expert systems depends on their users having common sense.

- **Heuristic Classification**

  One of the most feasible kinds of expert system given the present knowledge of AI is to put some information in one of a fixed set of categories using several sources of information. An example is advising whether to accept a proposed credit card purchase. Information is available about the owner of the credit card, his record of payment and also about the item he is buying and about the establishment from which he is buying it (e.g., about whether there have been previous credit card frauds at this establishment).

- **The applications of AI are shown in Fig 1.1:**

  ✓ Consumer Marketing

  - Have you ever used any kind of credit/ATM/store card while shopping?
  - if so, you have very likely been “input” to an AI algorithm
  - All of this information is recorded digitally
  - Companies like Nielsen gather this information weekly and search for patterns
    - general changes in consumer behavior
    - tracking responses to new products
    - identifying customer segments: targeted marketing, e.g., they find out that consumers with sports cars who buy textbooks respond well to offers of new credit cards.
  - Algorithms (“data mining”) search data for patterns based on mathematical theories of learning

  ✓ Identification Technologies

  - ID cards e.g., ATM cards
  - can be a nuisance and security risk: cards can be lost, stolen, passwords forgotten, etc
  - Biometric Identification, walk up to a locked door
    - Camera
    - Fingerprint device
    - Microphone
    - Computer uses biometric signature for identification
    - Face, eyes, fingerprints, voice pattern

    - This works by comparing data from person at door with stored library
    - Learning algorithms can learn the matching process by analyzing a large library database off-line, can improve its performance.

- ✓ Intrusion Detection
  - o Computer security - we each have specific patterns of computer use times of day, lengths of sessions, command used, sequence of commands, etc
    - would like to learn the "signature" of each authorized user
    - can identify non-authorized users
  - o How can the program automatically identify users?
    - record user's commands and time intervals
    - characterize the patterns for each user
    - model the variability in these patterns
    - classify (online) any new user by similarity to stored patterns
- ✓ Machine Translation
  - o Language problems in international business
    - e.g., at a meeting of Japanese, Korean, Vietnamese and Swedish investors, no common language
    - If you are shipping your software manuals to 127 countries, the solution is ; hire translators to translate
    - would be much cheaper if a machine could do this!
  - o How hard is automated translation
    - very difficult!
    - e.g., English to Russian
    - not only must the words be translated, but their meaning also!

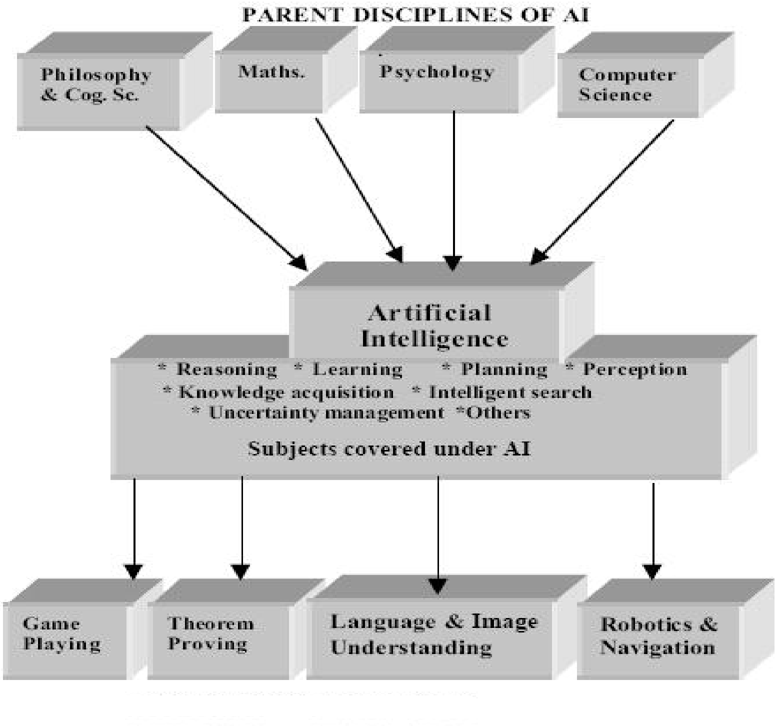

PARENT DISCIPLINES OF AI
Philosophy & Cog. Sc.
Maths.
Psychology
Computer Science
Artificial Intelligence
* Reasoning * Learning * Planning * Perception
* Knowledge acquisition * Intelligent search
* Uncertainty management *Others
Subjects covered under AI
Game Playing
Theorem Proving
Language & Image Understanding
Robotics & Navigation

Fig : Application areas of AI

Early work in AI

- "Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on."
- Scientific Goal To determine which ideas about knowledge representation, learning, rule systems, search, and so on, explain various sorts of real intelligence.
- Engineering Goal To solve real world problems using AI techniques such as knowledge representation, learning, rule systems, search, and so on.
- Traditionally, computer scientists and engineers have been more interested in the engineering goal, while psychologists, philosophers and cognitive scientists have been more interested in the scientific goal.
- The Roots - Artificial Intelligence has identifiable roots in a number of older disciplines, particularly:
  - ✓ Philosophy
  - ✓ Logic/Mathematics
  - ✓ Computation

- ✓ Psychology/Cognitive Science
- ✓ Biology/Neuroscience
- ✓ Evolution

- There is inevitably much overlap, e.g. between philosophy and logic, or between mathematics and computation. By looking at each of these in turn, we can gain a better understanding of their role in AI, and how these underlying disciplines have developed to play that role.
- Philosophy
  - ✓ ~400 BC Socrates asks for an algorithm to distinguish piety from non-piety.
  - ✓ ~350 BC Aristotle formulated different styles of deductive reasoning, which could mechanically generate conclusions from initial premises, e.g. Modus Ponens

    If A ? B and A then B

    If A implies B and A is true then B is true when it's raining you

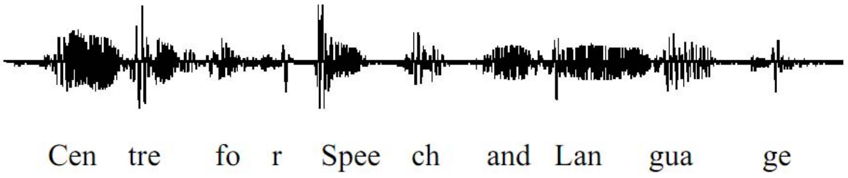
    get wet and it's raining then you get wet
  - ✓ 1596 – 1650 Rene Descartes idea of mind-body dualism – part of the mind is exempt from physical laws.
  - ✓ 1646 – 1716 Wilhelm Leibnitz was one of the first to take the materialist position which holds that the mind operates by ordinary physical processes – this has the implication that mental processes can potentially be carried out by machines.
- Logic/Mathematics
  - ✓ Earl Stanhope's Logic Demonstrator was a machine that was able to solve syllogisms, numerical problems in a logical form, and elementary questions of probability.
  - ✓ 1815 – 1864 George Boole introduced his formal language for making logical inference in 1847 – Boolean algebra.
  - ✓ 1848 – 1925 Gottlob Frege produced a logic that is essentially the first-order logic that today forms the most basic knowledge representation system.
  - ✓ 1906 – 1978 Kurt Gödel showed in 1931 that there are limits to what logic can do. His Incompleteness Theorem showed that in any formal logic powerful enough to describe the properties of natural numbers, there are true statements whose truth cannot be established by any algorithm.

- ✓ 1995 Roger Penrose tries to prove the human mind has non-computable capabilities.

- Computation
  - ✓ 1869 William Jevon’s Logic Machine could handle Boolean Algebra and Venn Diagrams, and was able to solve logical problems faster than human beings.
  - ✓ 1912 – 1954 Alan Turing tried to characterise exactly which functions are capable of being computed. Unfortunately it is difficult to give the notion of computation a formal definition. However, the Church-Turing thesis, which states that a Turing machine is capable of computing any computable function, is generally accepted as providing a sufficient definition. Turing also showed that there were some functions which no Turing machine can compute (e.g. Halting Problem).
  - ✓ 1903 – 1957 John von Neumann proposed the von Neuman architecture which allows a description of computation that is independent of the particular realisation of the computer.
  - ✓ 1960s Two important concepts emerged: Intractability (when solution time grows atleast exponentially) and Reduction (to ‘easier’ problems).
- Psychology / Cognitive Science
  - ✓ Modern Psychology / Cognitive Psychology / Cognitive Science is the science which studies how the mind operates, how we behave, and how our brains process information.
  - ✓ Language is an important part of human intelligence. Much of the early work on knowledge representation was tied to language and informed by research into linguistics.
  - ✓ It is natural for us to try to use our understanding of how human (and other animal) brains lead to intelligent behavior in our quest to build artificial intelligent systems. Conversely, it makes sense to explore the properties of artificial systems (computer models/simulations) to test our hypotheses concerning human systems.
  - ✓ Many sub-fields of AI are simultaneously building models of how the human system operates, and artificial systems for solving real world problems, and are allowing useful ideas to transfer between them.
- Biology / Neuroscience

  - ✓ Our brains (which give rise to our intelligence) are made up of tens of billions of neurons, each connected to hundreds or thousands of other neurons.
  - ✓ Each neuron is a simple processing device (e.g. just firing or not firing depending on the total amount of activity feeding into it). However, large networks of neurons are extremely powerful computational devices that can learn how best to operate.
  - ✓ The field of Connectionism or Neural Networks attempts to build artificial systems based on simplified networks of simplified artificial neurons.
  - ✓ The aim is to build powerful AI systems, as well as models of various human abilities.
  - ✓ Neural networks work at a sub-symbolic level, whereas much of conscious human reasoning appears to operate at a symbolic level.
  - ✓ Artificial neural networks perform well at many simple tasks, and provide good models of many human abilities. However, there are many tasks that they are not so good at, and other approaches seem more promising in those areas.
- Evolution
  - ✓ One advantage humans have over current machines/computers is that they have a long evolutionary history.
  - ✓ Charles Darwin (1809 – 1882) is famous for his work on evolution by natural selection. The idea is that fitter individuals will naturally tend to live longer and produce more children, and hence after many generations a population will automatically emerge with good innate properties.
  - ✓ This has resulted in brains that have much structure, or even knowledge, built in at birth.
  - ✓ This gives them at the advantage over simple artificial neural network systems that have to learn everything.
  - ✓ Computers are finally becoming powerful enough that we can simulate evolution and evolve good AI systems.
  - ✓ We can now even evolve systems (e.g. neural networks) so that they are good at learning.
  - ✓ A related field called genetic programming has had some success in evolving programs, rather than programming them by hand.
- Sub-fields of Artificial Intelligence

  - Neural Networks – e.g. brain modelling, time series prediction, classification
  - Evolutionary Computation – e.g. genetic algorithms, genetic programming
  - Vision – e.g. object recognition, image understanding
  - Robotics – e.g. intelligent control, autonomous exploration
  - Expert Systems – e.g. decision support systems, teaching systems
  - Speech Processing– e.g. speech recognition and production
  - Natural Language Processing – e.g. machine translation
  - Planning – e.g. scheduling, game playing
  - Machine Learning – e.g. decision tree learning, version space learning
- Speech Processing
  - As well as trying to understand human systems, there are also numerous real world applications: speech recognition for dictation systems and voice activated control; speech production for automated announcements and computer interfaces.
  - How do we get from sound waves to text streams and vice-versa?



- Natural Language Processing
  - For example, machine understanding and translation of simple sentences:
- Planning
  - Planning refers to the process of choosing/computing the correct sequence of steps to solve a given problem.
  - To do this we need some convenient representation of the problem domain. We can define states in some formal language, such as a subset of predicate logic, or a series of rules.
  - A plan can then be seen as a sequence of operations that transform the initial state into the goal state, i.e. the problem solution. Typically we will use some kind of search algorithm to find a good plan.
- Common Techniques
  - Even apparently radically different AI systems (such as rule based expert systems and neural networks) have many common techniques.
  - Four important ones are:

- Knowledge Representation: Knowledge needs to be represented somehow – perhaps as a series of if-then rules, as a frame based system, as a semantic network, or in the connection weights of an artificial neural network.
- Learning: Automatically building up knowledge from the environment – such as acquiring the rules for a rule based expert system, or determining the appropriate connection weights in an artificial neural network.
- Rule Systems: These could be explicitly built into an expert system by a knowledge engineer, or implicit in the connection weights learnt by a neural network.
- Search: This can take many forms – perhaps searching for a sequence of states that leads quickly to a problem solution, or searching for a good set of connection weights for a neural network by minimizing a fitness function.

AI and related fields

- **Logical AI**

  What a program knows about the world in general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals.

- **Search**

  AI programs often examine large numbers of possibilities, e.g. moves in a chess game or inferences by a theorem proving program. Discoveries are continually made about how to do this more efficiently in various domains.

- **Pattern Recognition**

  When a program makes observations of some kind, it is often programmed to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns, e.g. in a natural language text, in a chess position, or in the history of some event are also studied.

- **Representation**

  Facts about the world have to be represented in some way. Usually languages of mathematical logic are used.

- **Inference**

  From some facts, others can be inferred. Mathematical logical deduction is adequate for some purposes, but new methods of *non-monotonic* inference have been added to logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default, but the conclusion can be withdrawn if there is evidence to the contrary. For example, when we hear of a bird, we man infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Ordinary logical reasoning is monotonic in that the set of conclusions that can the drawn from a set of premises is a monotonic increasing function of the premises.

- **Common sense knowledge and reasoning**

  This is the area in which AI is farthest from human-level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress, e.g. in developing systems of *non-monotonic reasoning* and theories of action, yet more new ideas are needed.

- **Learning from experience**

  Programs do that. The approaches to AI based on *connectionism* and *neural nets* specialize in that. There is also learning of laws expressed in logic. Programs can only learn what facts or behaviors their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

- **Planning**

  Planning programs start with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, they generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

- **Epistemology**

This is a study of the kinds of knowledge that are required for solving problems in the world.

- **Ontology**

  Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are. Emphasis on ontology begins in the 1990s.

- **Heuristics**

  A heuristic is a way of trying to discover something or an idea imbedded in a program. The term is used variously in AI. *Heuristic functions* are used in some approaches to search to measure how far a node in a search tree seems to be from a goal. *Heuristic predicates* that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, may be more useful.

- **Genetic Programming**

  Genetic programming is a technique for getting programs to solve a task by mating random Lisp programs and selecting fittest in millions of generations.

Search and Control Strategies:

Problem solving is an important aspect of Artificial Intelligence. A problem can be considered to consist of a goal and a set of actions that can be taken to lead to the goal. At any given time, we consider the state of the search space to represent where we have reached as a result of the actions we have applied so far. For example, consider the problem of looking for a contact lens on a football field. The initial state is how we start out, which is to say we know that the lens is somewhere on the field, but we don’t know where. If we use the representation where we examine the field in units of one square foot, then our first action might be to examine the square in the top-left corner of the field. If we do not find the lens there, we could consider the state now to be that we have examined the top-left square and have not found the lens. After a number of actions, the state might be that we have examined 500 squares, and we have now just found the lens in the last square we examined. This is a goal state because it satisfies the goal that we had of finding a contact lens.

Search is a method that can be used by computers to examine a problem space like this in order to find a goal. Often, we want to find the goal as quickly as possible or without using too many resources. A problem space can also be considered to be a search space because in order to solve the problem, we will search the space for a goal state.We will continue to use the term search space to describe this concept. In this chapter, we will look at a number of methods for examining a search space. These methods are called search methods.

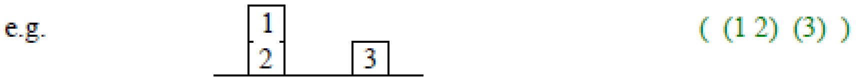
- The Importance of Search in AI
    - It has already become clear that many of the tasks underlying AI can be phrased in terms of a search for the solution to the problem at hand.
    - Many goal based agents are essentially problem solving agents which must decide what to do by searching for a sequence of actions that lead to their solutions.
    - For production systems, we have seen the need to search for a sequence of rule applications that lead to the required fact or action.
    - For neural network systems, we need to search for the set of connection weights that will result in the required input to output mapping.
- Which search algorithm one should use will generally depend on the problem domain? There are four important factors to consider:
    - Completeness – Is a solution guaranteed to be found if at least one solution exists?
    - Optimality – Is the solution found guaranteed to be the best (or lowest cost) solution if there exists more than one solution?
    - Time Complexity – The upper bound on the time required to find a solution, as a function of the complexity of the problem.
    - Space Complexity – The upper bound on the storage space (memory) required at any point during the search, as a function of the complexity of the problem.

Preliminary concepts

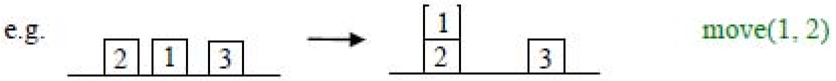
- Two varieties of **space-for-time** algorithms:
    - Input enhancement — preprocess the input (or its part) to store some info to be used later in solving the problem
        - Counting for sorting
        - String searching algorithms
    - Prestructuring — preprocess the input to make accessing its elements easier
        - Hashing

- Indexing schemes (e.g., B-trees)

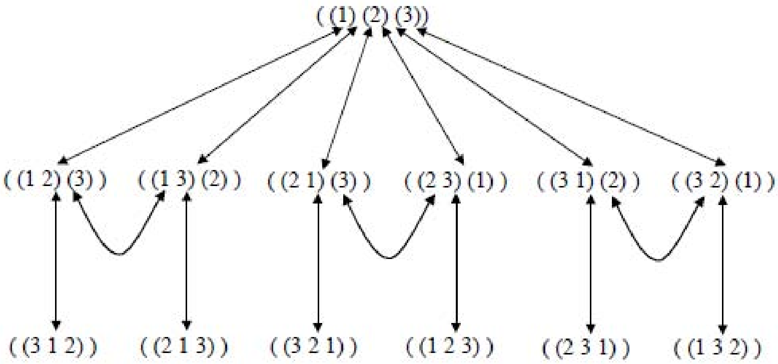
- **State Space Representations:** The state space is simply the space of all possible states, or configurations, that our system may be in. Generally, of course, we prefer to work with some convenient representation of that search space.
- There are two components to the representation of state spaces:
  - Static States



  - Transitions between States

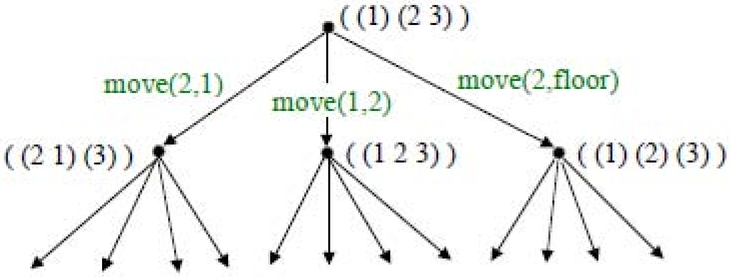


- **State Space Graphs:** If the number of possible states of the system is small enough, we can represent all of them, along with the transitions between them, in a state space graph, e.g.



- **Routes through State Space:** Our general aim is to search for a route, or sequence of transitions, through the state space graph from our initial state to a goal state.
- Sometimes there will be more than one possible goal state. We define a goal test to determine if a goal state has been achieved.
- The solution can be represented as a sequence of link labels (or transitions) on the state space graph. Note that the labels depend on the direction moved along the link.
- Sometimes there may be more than one path to a goal state, and we may want to find the optimal (best possible) path. We can define link costs and path costs for measuring the cost of going along a particular path, e.g. the path cost may just equal the number of links, or could be the sum of individual link costs.

- For most realistic problems, the state space graph will be too large for us to hold all of it explicitly in memory at any one time.
- **Search Trees:** It is helpful to think of the search process as building up a search tree of routes through the state space graph. The root of the search tree is the search node corresponding to the initial state.
- The leaf nodes correspond either to states that have not yet been expanded, or to states that generated no further nodes when expanded.



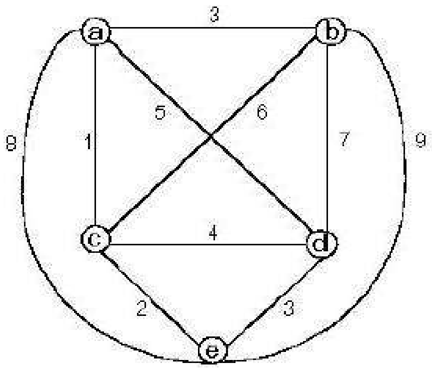
- At each step, the search algorithm chooses a new unexpanded leaf node to expand. The different search strategies essentially correspond to the different algorithms one can use to select which is the next mode to be expanded at each stage.

Examples of search problems

- **Traveling Salesman Problem:** Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city.
- A lower bound on the length l of any tour can be computed as follows
  - ✓ For each city i, $1 \leq i \leq n$, find the sum s_i of the distances from city i to the two nearest cities.
  - ✓ Compute the sum s of these n numbers.
  - ✓ Divide the result by 2 and round up the result to the nearest integer

$$lb = s / 2$$

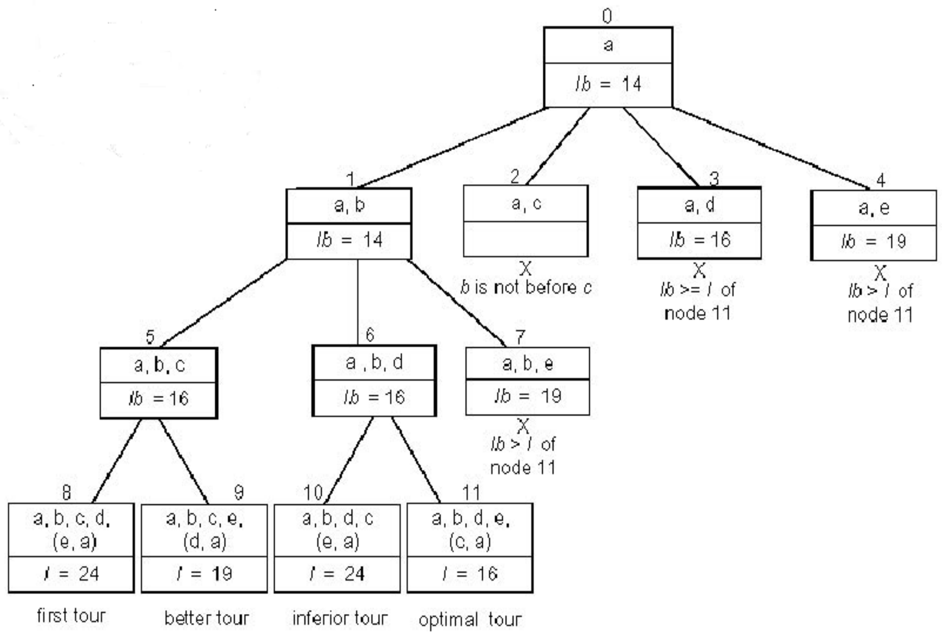
- The lower bound for the graph shown in the Fig 5.1 can be computed as follows:



lb = [(1 + 3) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 3)] / 2 = 14.

- For any subset of tours that must include particular edges of a given graph, the lower bound can be modified accordingly. E.g.: For all the Hamiltonian circuits of the graph that must include edge (a, d), the lower bound can be computed as follows:
  lb = [(1 + 5) + (3 + 6) + (1 + 2) + (3 + 5) + (2 + 3)] / 2 = 16.

- Applying the branch-and-bound algorithm, with the bounding function lb = s / 2, to find the shortest Hamiltonian circuit for the given graph, we obtain the state-space tree as shown below:
- To reduce the amount of potential work, we take advantage of the following two observations:
  - ✓ We can consider only tours that start with a.
  - ✓ Since the graph is undirected, we can generate only tours in which b is visited before c.
- In addition, after visiting n – 1 cities, a tour has no choice but to visit the remaining unvisited city and return to the starting one is shown in the Fig 5.2



Root node includes only the starting vertex a with a lower bound of

lb = [(1 + 3) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 3)] / 2 = 14.

- Node 1 represents the inclusion of edge (a, b)
  lb = [(1 + 3) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 3)] / 2 = 14.
- Node 2 represents the inclusion of edge (a, c). Since b is not visited before c, this node is terminated.
- Node 3 represents the inclusion of edge (a, d)
  lb = [(1 + 5) + (3 + 6) + (1 + 2) + (3 + 5) + (2 + 3)] / 2 = 16.
- Node 1 represents the inclusion of edge (a, e)
  lb = [(1 + 8) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 8)] / 2 = 19.
- Among all the four live nodes of the root, node 1 has a better lower bound. Hence we branch from node 1.
- Node 5 represents the inclusion of edge (b, c)
  lb = [(1 + 3) + (3 + 6) + (1 + 6) + (3 + 4) + (2 + 3)] / 2 = 16.
- Node 6 represents the inclusion of edge (b, d)
  lb = [(1 + 3) + (3 + 7) + (1 + 2) + (3 + 7) + (2 + 3)] / 2 = 16.
- Node 7 represents the inclusion of edge (b, e)
  lb = [(1 + 3) + (3 + 9) + (1 + 2) + (3 + 4) + (2 + 9)] / 2 = 19.

- Since nodes 5 and 6 both have the same lower bound, we branch out from each of them.
- Node 8 represents the inclusion of the edges (c, d), (d, e) and (e, a). Hence, the length of the tour,
  l = 3 + 6 + 4 + 3 + 8 = 24.

- Node 9 represents the inclusion of the edges (c, e), (e, d) and (d, a). Hence, the length of the tour,
  l = 3 + 6 + 2 + 3 + 5 = 19.

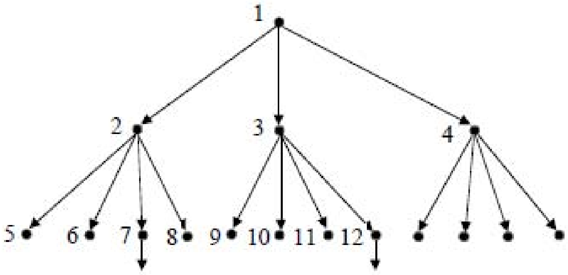
- Node 10 represents the inclusion of the edges (d, c), (c, e) and (e, a). Hence, the length of the tour,
  l = 3 + 7 + 4 + 2 + 8 = 24.

- Node 11 represents the inclusion of the edges (d, e), (e, c) and (c, a). Hence, the length of the tour,
  l = 3 + 7 + 3 + 2 + 1 = 16.

- Node 11 represents an optimal tour since its tour length is better than or equal to the other live nodes, 8, 9, 10, 3 and 4.
- The optimal tour is a → b → d → e → c → a with a tour length of 16.

Uniformed or Blind search

- **Breadth First Search (BFS):** BFS expands the leaf node with the lowest path cost so far, and keeps going until a goal node is generated. If the path cost simply equals the number of links, we can implement this as a simple queue ("first in, first out").

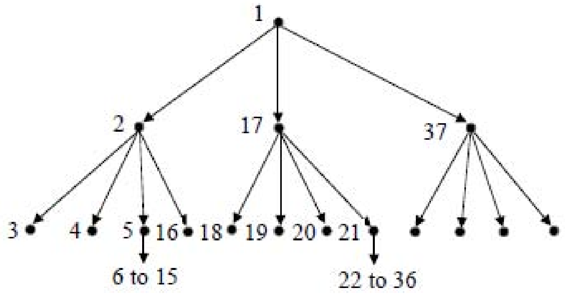


- This is guaranteed to find an optimal path to a goal state. It is memory intensive if the state space is large. If the typical branching factor is b, and the depth of the shallowest goal state is d – the space complexity is O(b^d), and the time complexity is O(b^d).
- BFS is an easy search technique to understand. The algorithm is presented below.

```
breadth_first_search ()

{

        store initial state in queue Q

        set state in the front of the Q as current state ;

        while (goal state is reached OR Q is empty)

        {

                apply rule to generate a new state from the current

                state ;

                if (new state is goal state) quit ;

                        else if (all states generated from current states are

                        exhausted)

                        {

                                delete the current state from the Q ;

                                set front element of Q as the current state ;

                        }

                        else continue ;

                }

}
```

- The algorithm is illustrated using the bridge components configuration problem. The initial state is PDFG, which is not a goal state; and hence set it as the current state. Generate another state DPFG (by swapping 1st and 2nd position values) and add it to the list. That is not a goal state, hence; generate next successor state, which is FDPG (by swapping 1st and 3rd position values). This is also not a goal state; hence add it to the list and generate the next successor state GDFP.

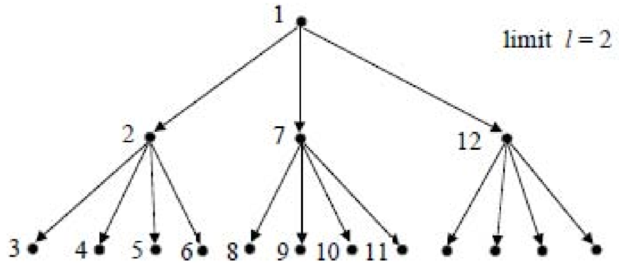
- Only three states can be generated from the initial state. Now the queue Q will have three elements in it, viz., DPFG, FDPG and GDFP. Now take DPFG (first state in the list) as the current state and continue the process, until all the states generated from this are evaluated. Continue this process, until the goal state DGPF is reached.
- The 14th evaluation gives the goal state. It may be noted that, all the states at one level in the tree are evaluated before the states in the next level are taken up; i.e., the evaluations are carried out breadth-wise. Hence, the search strategy is called breadth-first search.
- **Depth First Search (DFS):** DFS expands the leaf node with the highest path cost so far, and keeps going until a goal node is generated. If the path cost simply equals the number of links, we can implement this as a simple stack ("last in, first out").



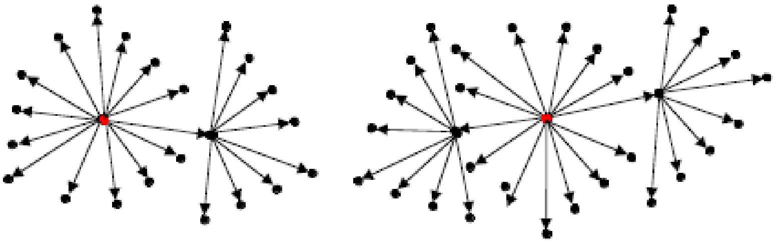
- This is not guaranteed to find any path to a goal state. It is memory efficient even if the state space is large. If the typical branching factor is b, and the maximum depth of the tree is m – the space complexity is O(bm), and the time complexity is O(b^m).
- In DFS, instead of generating all the states below the current level, only the first state below the current level is generated and evaluated recursively. The search continues till a further successor cannot be generated.
- Then it goes back to the parent and explores the next successor. The algorithm is given below.

```
depth_first_search ()

{

        set initial state to current state ;

        if (initial state is current state) quit ;
        else
        {
            if (a successor for current state exists)
            {
                generate a successor of the current state and
                set it as current state ;
            }
            else return ;
            depth_first_search (current_state) ;
            if (goal state is achieved) return ;
            else continue ;
        }
    }
```

- Since DFS stores only the states in the current path, it uses much less memory during the search compared to BFS.
- The probability of arriving at goal state with a fewer number of evaluations is higher with DFS compared to BFS. This is because, in BFS, all the states in a level have to be evaluated before states in the lower level are considered. DFS is very efficient when more acceptable solutions exist, so that the search can be terminated once the first acceptable solution is obtained.
- BFS is advantageous in cases where the tree is very deep.
- An ideal search mechanism is to combine the advantages of BFS and DFS.
- **Depth Limited Search (DLS):** DLS is a variation of DFS. If we put a limit l on how deep a depth first search can go, we can guarantee that the search will terminate (either in success or failure).



- If there is at least one goal state at a depth less than l, this algorithm is guaranteed to find a goal state, but it is not guaranteed to find an optimal path. The space complexity is O(bl), and the time complexity is O(b^l).
- **Depth First Iterative Deepening Search (DFIDS):** DFIDS is a variation of DLS. If the lowest depth of a goal state is not known, we can always find the best limit l for DLS by trying all possible depths l = 0, 1, 2, 3, … in turn, and stopping once we have achieved a goal state.
- This appears wasteful because all the DLS for l less than the goal level are useless, and many states are expanded many times. However, in practice, most of the time is spent at the deepest part of the search tree, so the algorithm actually combines the benefits of DFS and BFS.
- Because all the nodes are expanded at each level, the algorithm is complete and optimal like BFS, but has the modest memory requirements of DFS. Exercise: if we had plenty of memory, could/should we avoid expanding the top level states many times?
- The space complexity is O(bd) as in DLS with l = d, which is better than BFS.
- The time complexity is O(b^d) as in BFS, which is better than DFS.
- **Bi-Directional Search (BDS):** The idea behind bi-directional search is to search simultaneously both forward from the initial state and backwards from the goal state, and stop when the two BFS searches meet in the middle.



- This is not always going to be possible, but is likely to be feasible if the state transitions are reversible. The algorithm is complete and optimal, and since the two search depths are ~d/2, it has space complexity O($b^{d/2}$), and time complexity O($b^{d/2}$). However, if there is more than one possible goal state, this must be factored into the complexity.

- **Repeated States:** In the above discussion we have ignored an important complication that often arises in search processes – the possibility that we will waste time by expanding states that have already been expanded before somewhere else on the search tree.
- For some problems this possibility can never arise, because each state can only be reached in one way.
- For many problems, however, repeated states are unavoidable. This will include all problems where the transitions are reversible, e.g.

((1 2 3)) → ((1) (2 3)) → ((1 2 3)) → ((1) (2 3)) → ((1 2 3)) → ...

- The search trees for these problems are infinite, but if we can prune out the repeated states, we can cut the search tree down to a finite size, We effectively only generate a portion of the search tree that matches the state space graph.
- **Avoiding Repeated States:** There are three principal approaches for dealing with repeated states:
  - Never return to the state you have just come from

    The node expansion function must be prevented from generating any node successor that is the same state as the node's parent.

  - Never create search paths with cycles in them

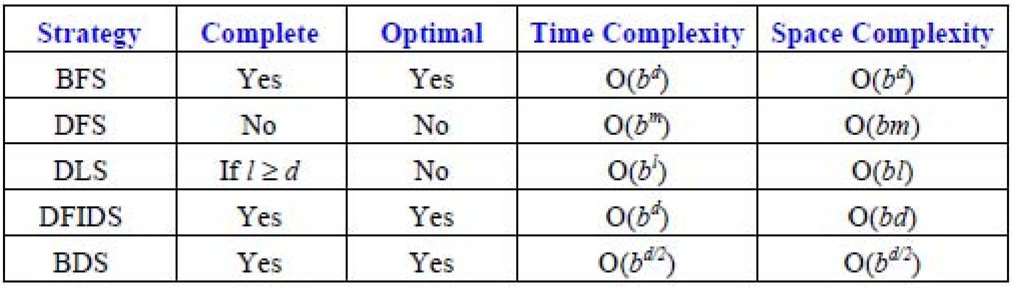
    The node expansion function must be prevented from generating

    any node successor that is the same state as any of the node's ancestors.

  - Never generate states that have already been generated before

    This requires that every state ever generated is remembered, potentially resulting in space complexity of O(b^d).

- **Comparing the Uninformed Search Algorithms:** We can now summarize the properties of our five uninformed search strategies:



Strategy	Complete	Optimal	Time Complexity	Space Complexity
BFS	Yes	Yes	O(b^d)	O(b^d)
DFS	No	No	O(b^m)	O(bm)
DLS	If $l \geq d$	No	O(b^l)	O(bl)
DFIDS	Yes	Yes	O(b^d)	O(bd)
BDS	Yes	Yes	O($b^{d/2}$)	O($b^{d/2}$)

- Simple BFS and BDS are complete and optimal but expensive with respect to space and time.
- DFS requires much less memory if the maximum tree depth is limited, but has no guarantee of finding any solution, let alone an optimal one. DLS offers an improvement over DFS if we have some idea how deep the goal is.
- The best overall is DFID which is complete, optimal and has low memory requirements, but still exponential time.

Informed search

- Informed search uses some kind of evaluation function to tell us how far each expanded state is from a goal state, and/or some kind of heuristic function to help us decide which state is likely to be the best one to expand next.
- The hard part is to come up with good evaluation and/or heuristic functions. Often there is a natural evaluation function, such as distance in miles or number objects in the wrong position.
- Sometimes we can learn heuristic functions by analyzing what has worked well in similar previous searches.
- The simplest idea, known as greedy best first search, is to expand the node that is already closest to the goal, as that is most likely to lead quickly to a solution. This is like DFS in that it attempts to follow a single route to the goal, only attempting to try a different route when it reaches a dead end. As with DFS, it is not complete, not optimal, and has time and complexity of O(b^m). However, with good heuristics, the time complexity can be reduced substantially.
- **Branch and Bound:** An enhancement of backtracking.
- Applicable to optimization problems.

- For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution).
- Uses the bound for:
  - ➢ Ruling out certain nodes as “nonpromising” to prune the tree – if a node’s bound is not better than the best solution seen so far.
  - ➢ Guiding the search through state-space.
- The search path at the current node in a state-space tree can be terminated for any one of the following three reasons:
  - ➢ The value of the node’s bound is not better than the value of the best solution seen so far.
  - ➢ The node represents no feasible solutions because the constraints of the problem are already violated.
  - ➢ The subset of feasible solutions represented by the node consists of a single point and hence we compare the value of the objective function for this feasible solution with that of the best solution seen so far and update the latter with the former if the new solution is better.
- **Best-First branch-and-bound:**
  - ➢ A variation of backtracking.
  - ➢ Among all the nonterminated leaves, called as the live nodes, in the current tree, generate all the children of the most promising node, instead of generation a single child of the last promising node as it is done in backtracking.
  - ➢ Consider the node with the best bound as the most promising node.
- **A* Search:** Suppose that, for each node n in a search tree, an evaluation function f(n) is defined as the sum of the cost g(n) to reach that node from the start state, plus an estimated cost h(n) to get from that state to the goal state. That f(n) is then the estimated cost of the cheapest solution through n.
- A* search, which is the most popular form of best-first search, repeatedly picks the node with the lowest f(n) to expand next. It turns out that if the heuristic function h(n) satisfies certain conditions, then this strategy is both complete and optimal.
- In particular, if h(n) is an admissible heuristic, i.e. is always optimistic and never overestimates the cost to reach the goal, then A* is optimal.

- The classic example is finding the route by road between two cities given the straight line distances from each road intersection to the goal city. In this case, the nodes are the intersections, and we can simply use the straight line distances as h(n).
- **Hill Climbing / Gradient Descent:** The basic idea of hill climbing is simple: at each current state we select a transition, evaluate the resulting state, and if the resulting state is an improvement we move there, otherwise we try a new transition from where we are.
- We repeat this until we reach a goal state, or have no more transitions to try. The transitions explored can be selected at random, or according to some problem specific heuristics.
- In some cases, it is possible to define evaluation functions such that we can compute the gradients with respect to the possible transitions, and thus compute which transition direction to take to produce the best improvement in the evaluation function.
- Following the evaluation gradients in this way is known as gradient descent.
- In neural networks, for example, we can define the total error of the output activations as a function of the connection weights, and compute the gradients of how the error changes as we change the weights. By changing the weights in small steps against those gradients, we systematically minimize the network's output errors.

Searching And-Or graphs

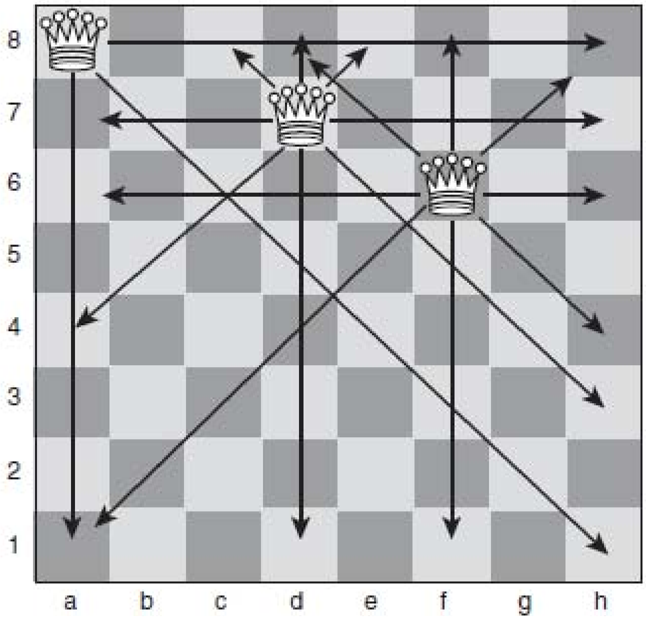
- The DFS and BFS strategies for OR trees and graphs can be adapted for And-Or trees
- The main difference lies in the way termination conditions are determined, since all goals following an And node must be realized, whereas a single goal node following an Or node will do
- A more general optimal strategy is AO* (O for ordered) algorithm
- As in the case of the A* algorithm, we use the open list to hold nodes that have been generated but not expanded and the closed list to hold nodes that have been expanded
- The algorithm is a variation of the original given by Nilsson
- It requires that nodes traversed in the tree be labeled as solved or unsolved in the solution process to account for And node solutions which require solutions to all successors nodes.
- A solution is found when the start node is labeled as solved

- The AO* algorithm
    - Step 1: Place the start node s on open
    - Step 2: Using the search tree constructed thus far, compute the most promising solution tree T_0
    - Step 3:Select a node n that is both on open and a part of $T_{0.}$ Remove n from open and place it on closed
    - Step 4: If n ia terminal goal node, label n as solved. If the solution of n results in any of n's ancestors being solved, label all the ancestors as solved. If the start node s is solved, exit with success where T_0 is the solution tree. Remove from open all nodes with a solved ancestor
    - Step 5: If n is not a solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. If any of n's ancestors become unsolvable because n is, label them unsolvable as well. Remove from open all nodes with unsolvable ancestors
    - Otherwise, expand node n generating all of its successors. For each such successor node that contains more than one subproblem, generate their successors to give individual subproblems. Attach to each newly generated node a back pointer to its predecessor. Compute the cost estimate h* for each newly generated node and place all such nodes thst do not yet have descendents on open. Next recomputed the values oh h* at n and each ancestors of n
    - Step 7: Return to step 2
- It can be shown that AO* will always find a minimum-cost solution tree if one exists, provided only that $h^*(n) \leq h(n)$, and all arc costs are positive. Like A*, the efficiency depends on how closely h* approximates h

Constraint Satisfaction Search

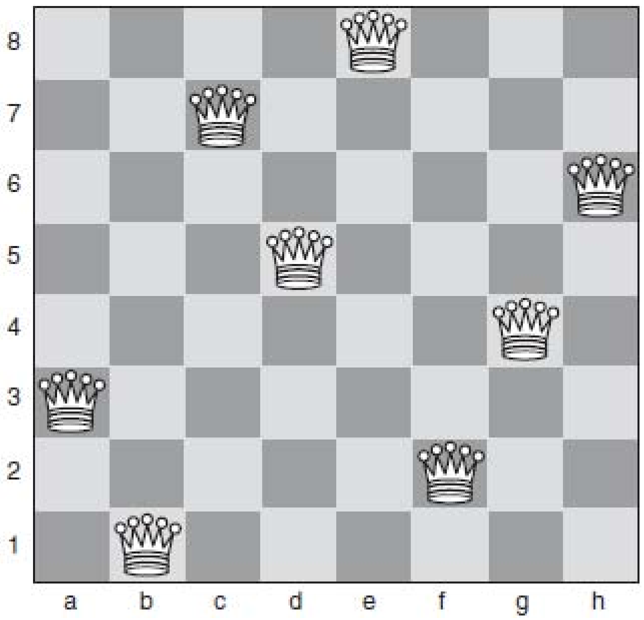
- Search can be used to solve problems that are limited by constraints, such as the eight-queens problem. Such problems are often known as Constraint Satisfaction Problems, or CSPs. I

- n this problem, eight queens must be placed on a chess board in such a way that no two queens are on the same diagonal, row, or column. If we use traditional chess board notation, we mark the columns with letters from a to g and the rows with numbers from 1 to 8. So, a square can be referred to by a letter and a number, such as a4 or g7.
- This kind of problem is known as a constraint satisfaction problem (CSP) because a solution must be found that satisfies the constraints.
- In the case of the eight-queens problem, a search tree can be built that represents the possible positions of queens on the board. One way to represent this is to have a tree that is 8-ply deep, with a branching factor of 64 for the first level, 63 for the next level, and so on, down to 57 for the eighth level.
- A goal node in this tree is one that satisfies the constraints that no two queens can be on the same diagonal, row, or column.
- An extremely simplistic approach to solving this problem would be to analyze every possible configuration until one was found that matched the constraints.
- A more suitable approach to solving the eight-queens problem would be to use depth-first search on a search tree that represents the problem in the following manner:
    - The first branch from the root node would represent the first choice of a square for a queen. The next branch from these nodes would represent choices of where to place the second queen.
    - The first level would have a branching factor of 64 because there are 64 possible squares on which to place the first queen. The next level would have a somewhat lower branching factor because once a queen has been placed, the constraints can be used to determine possible squares upon which the next queen can be placed.
    - The branching factor will decrease as the algorithm searches down the tree. At some point, the tree will terminate because the path being followed will lead to a position where no more queens can be placed on legal squares on the board, and there are still some queens remaining.



In fact, because each row and each column must contain exactly one queen, the branching factor can be significantly reduced by assuming that the first queen must be placed in row 1, the second in row 2, and so on. In this way, the first level will have a branching factor of 8 (a choice of eight squares on which the first queen can be placed), the next 7, the next 6, and so on.

- The search tree can be further simplified as each queen placed on the board "uses up" a diagonal, meaning that the branching factor is only 5 or 6 after the first choice has been made, depending on whether the first queen is placed on an edge of the board (columns a or h) or not.
- The next level will have a branching factor of about 4, and the next may have a branching factor of just 2, as shown in Fig 6.1.
- The arrows in Fig 6.1 show the squares to which each queen can move.
- Note that no queen can move to a square that is already occupied by another queen.



- In Fig 6.1, the first queen was placed in column a of row 8, leaving six choices for the next row. The second queen was placed in column d of row 7, leaving four choices for row 6. The third queen was placed in column f in row 6, leaving just two choices (column c or column h) for row 5.
- Using knowledge like this about the problem that is being solved can help to significantly reduce the size of the search tree and thus improve the efficiency of the search solution.
- A solution will be found when the algorithm reaches depth 8 and successfully places the final queen on a legal square on the board.
- A goal node would be a path containing eight squares such that no two squares shared a diagonal, row, or column.
- One solution to the eight-queens problem is shown in above Fig .
- Note that in this solution, if we start by placing queens on squares e8, c7, h6, and then d5, once the fourth queen has been placed, there are only two choices for placing the fifth queen (b4 or g4). If b4 is chosen, then this leaves no squares that could be chosen for the final three queens to satisfy the constraints. If g4 is chosen for the fifth queen, as has been done in Fig 6.2, only one square is available for the sixth queen (a3), and the final two choices are similarly constrained. So, it can be seen that by applying the

constraints appropriately, the search tree can be significantly reduced for this problem.

- Using chronological backtracking in solving the eight-queens problem might not be the most efficient way to identify a solution because it will backtrack over moves that did not necessarily directly lead to an error, as well as ones that did. In this case, nonchronological backtracking, or dependency-directed backtracking could be more useful because it could identify the steps earlier in the search tree that caused the problem further down the tree.

Forward Checking

- In fact, backtracking can be augmented in solving problems like the eightqueens problem by using a method called forward checking.
- As each queen is placed on the board, a forward-checking mechanism is used to delete from the set of possible future choices any that have been rendered impossible by placing the queen on that square.
- For example, if a queen is placed on square a1, forward checking will remove all squares in row 1, all squares in column a, and also squares b2, c3, d4, e5, f6, g7, and h8.
- In this way, if placing a queen on the board results in removing all remaining squares, the system can immediately backtrack, without having to attempt to place any more queens.
- This can often significantly improve the performance of solutions for CSPs such as the eight-queens problem.

Most-Constrained Variables

- A further improvement in performance can be achieved by using the most-constrained variable heuristic.
- At each stage of the search, this heuristic involves working with the variable that has the least possible number of valid choices.

- In the case of the eight-queens problem, this might be achieved by considering the problem to be one of assigning a value to eight variables, a through h. Assigning value 1 to variable a means placing a queen in square a1.
- To use the most constrained variable heuristic with this representation means that at each move we assign a value to the variable that has the least choices available to it. Hence, after assigning a = 1, b = 3, and c = 5, this leaves three choices for d, three choices for e, one choice for f, three choices for g, and three choices for h. Hence, our next move is to place a queen in column f.
- This heuristic is perhaps more clearly understood in relation to the mapcoloring problem. It makes sense that, in a situation where a particular country can be given only one color due to the colors that have been assigned to its neighbors, that country be colored next.
- The most-constraining variable heuristic is similar in that it involves assigning a value next to the variable that places the greatest number of constraints on future variables.
- The least-constraining value heuristic is perhaps more intuitive than the two already presented in this section.
- This heuristic involves assigning a value to a variable that leaves the greatest number of choices for other variables.
- This heuristic can be used to make n-queens problems with extremely large values of n quite solvable.

Example: Cryptographic Problems

- The constraint satisfaction procedure is also a useful way to solve problems such as cryptographic problems. For example:

  FORTY

  + TEN

  + TEN

  SIXTY

  Solution:

  29786

+ 850

+ 850

31486

- This cryptographic problem can be solved by using a Generate and Test method, applying the following constraints:
  - Each letter represents exactly one number.
  - No two letters represent the same number.
- Generate and Test is a brute-force method, which in this case involves cycling through all possible assignments of numbers to letters until a set is found that meets the constraints and solves the problem.
- Without using constraints, the method would first start by attempting to assign 0 to all letters, resulting in the following sum:

  00000

  + 000

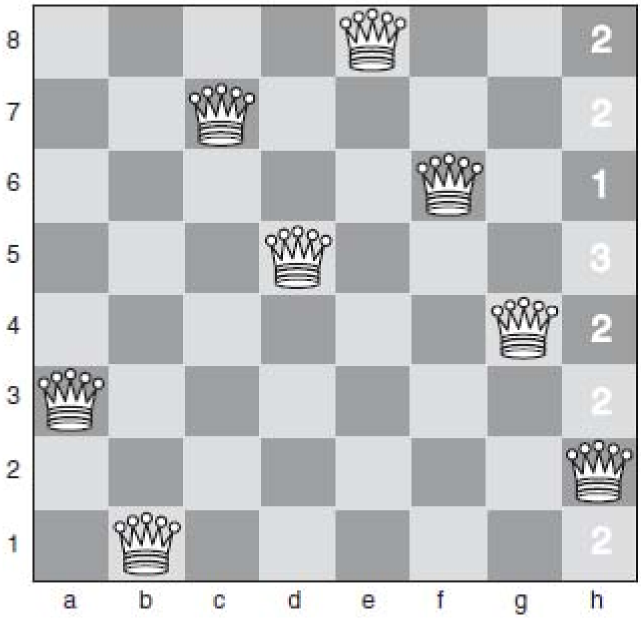
  + 000

  00000

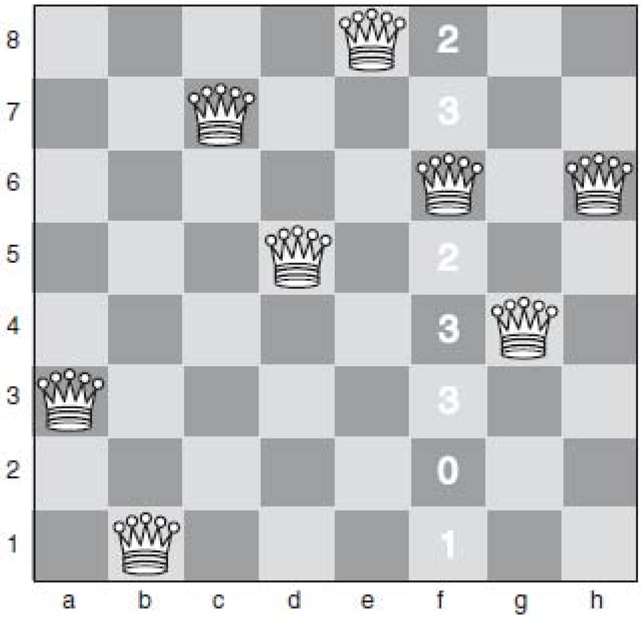
- Although this may appear to be a valid solution to the problem, it does not meet the constraints laid down that specify that each letter can be assigned only one number, and each number can be assigned only to one letter.
- Hence, constraints are necessary simply to find the correct solution to the problem. They also enable us to reduce the size of the search tree.
- In this case, for example, it is not necessary to examine possible solutions where two letters have been assigned the same number, which dramatically reduces the possible solutions to be examined.

Heuristic Repair

- Heuristics can be used to improve performance of solutions to constraint satisfaction problems.
- One way to do this is to use a heuristic repair method, which involves generating a possible solution (randomly, or using a heuristic to generate a position that is close to a solution) and then making changes that reduce the distance of the state from the goal.
- In the case of the eight-queens problem, this could be done using the minconflicts heuristic.
- To move from one state to another state that is likely to be closer to a solution using the min-conflicts heuristic, select one queen that conflicts with another queen (in other words, it is on the same row, column, or diagonal as another queen).
- Now move that queen to a square where it conflicts with as few queens as possible. Continue with another queen. To see how this method would work, consider the starting position shown in Fig 6.3.



- This starting position has been generated by placing the queens such that there are no conflicts on rows or columns. The only conflict here is that the queen in column 3 (on c7) is on a diagonal with the queen in column h (on h2).
- To move toward a solution, we choose to move the queen that is on column h.
- We will only ever apply a move that keeps a queen on the same column because we already know that we need to have one queen on each column.
- Each square in column h has been marked with a number to show how many other queens that square conflicts with. Our first move will be to move the queen on column h up to row 6, where it will conflict only with one queen. Then we arrive at the position shown in below Fig
- Because we have created a new conflict with the queen on row 6 (on f6), our next move must be to move this queen. In fact, we can move it to a square where it has zero conflicts. This means the problem has been solved, and there are no remaining conflicts.
- This method can be used not only to solve the eight-queens problem but also has been successfully applied to the n-queens problem for extremely large values of n. It has been shown that, using this method, the 1,000,000 queens problem can be solved in an average of around 50 steps.
- Solving the 1,000,000-queens problem using traditional search techniques would be impossible because it would involve searching a tree with a branching factor of 10^{12}.



Local Search and Metaheuristics

- Local search methods work by starting from some initial configuration (usually random) and making small changes to the configuration until a state is reached from which no better state can be achieved.
- Hill climbing is a good example of a local search technique.
- Local search techniques, used in this way, suffer from the same problems as hill climbing and, in particular, are prone to finding local maxima that are not the best solution possible.
- The methods used by local search techniques are known as metaheuristics.
- Examples of metaheuristics include simulated annealing, tabu search, genetic algorithms, ant colony optimization, and neural networks.
- This kind of search method is also known as local optimization because it is attempting to optimize a set of values but will often find local maxima rather than a global maximum.
- A local search technique applied to the problem of allocating teachers to classrooms would start from a random position and make small changes until a configuration was reached where no inappropriate allocations were made.
- **Exchanging Heuristics**
  - The simplest form of local search is to use an exchanging heuristic.

- An exchanging heuristic moves from one state to another by exchanging one or more variables by giving them different values. We saw this in solving the eight-queens problem as heuristic repair.
- A k-exchange is considered to be a method where k variables have their values changed at each step.
- The heuristic repair method we applied to the eight-queens problem was 2-exchange.
- A k-exchange can be used to solve the traveling salesman problem. A tour (a route through the cities that visits each city once, and returns to the start) is generated at random. Then, if we use 2-exchange, we remove two edges from the tour and substitute them for two other edges. If this pro duces a valid tour that is shorter than the previous one, we move on from here. Otherwise, we go back to the previous tour and try a different set of substitutions.
- In fact, using k = 2 does not work well for the traveling salesman problem, whereas using k = 3 produces good results.
- Using larger numbers of k will give better and better results but will also require more and more iterations.
- Using k = 3 gives reasonable results and can be implemented efficiently. It does, of course, risk finding local maxima, as is often the case with local search methods.

- **Iterated Local Search**
    - Iterated local search techniques attempt to overcome the problem of local maxima by running the optimization procedure repeatedly, from different initial states.
    - If used with sufficient iterations, this kind of method will almost always find a global maximum.
    - The aim, of course, in running methods like this is to provide a very good solution without needing to exhaustively search the entire problem space.
    - In problems such as the traveling salesman problem, where the search space grows extremely quickly as the number of cities increases, results can be generated that are good enough (i.e., a local maximum) without using many iterations, where a perfect solution would be impossible to find (or at least it would be impossible to guarantee a perfect solution even one iteration of local search may happen upon the global maximum).

- **Tabu Search**
    - ➢ Tabu search is a metaheuristic that uses a list of states that have already been visited to attempt to avoid repeating paths.
    - ➢ The tabu search metaheuristic is used in combination with another heuristic and operates on the principle that it is worth going down a path that appears to be poor if it avoids following a path that has already been visited.
    - ➢ In this way, tabu search is able to avoid local maxima.

Simulated Annealing

- Annealing is a process of producing very strong glass or metal, which involves heating the material to a very high temperature and then allowing it to cool very slowly.
- In this way, the atoms are able to form the most stable structures, giving the material great strength.
- Simulated annealing is a local search metaheuristic based on this method and is an extension of a process called metropolisMonteCarlo simulation.
- Simulated annealing is applied to a multi-value combinatorial problem where values need to be chosen for many variables to produce a particular value for some global function, dependent on all the variables in the system.
- This value is thought of as the energy of the system, and in general the aim of simulated annealing is to find a minimum energy for a system.
- Simple Monte Carlo simulation is a method of learning information (such as shape) about the shape of a search space. The process involves randomly selecting points within the search space.
- An example of its use is as follows: A square is partially contained within a circle. Simple Monte Carlo simulation can be used to identify what proportion of the square is within the circle and what proportion is outside the circle. This is done by randomly sampling points within the square and checking which ones are within the circle and which are not.
- Metropolis Monte Carlo simulation extends this simple method as follows: Rather than selecting new states from the search space at random, a new state is chosen by making a small change to the current state.

- If the new state means that the system as a whole has a lower energy than it did in the previous state, then it is accepted.
- If the energy is higher than for the previous state, then a probability is applied to determine whether the new state is accepted or not. This probability is called a Boltzmann acceptance criterion and is calculated as follows: e(_dE/T) where T is the current temperature of the system, and dE is the increase in energy that has been produced by moving from the previous state to the new state.
- The temperature in this context refers to the percentage of steps that can be taken that lead to a rise in energy: At a higher temperature, more steps will be accepted that lead to a rise in energy than at low temperature.
- To determine whether to move to a higher energy state or not, the probability e(_dE/T) is calculated, and a random number is generated between 0 and 1. If this random number is lower than the probability function, the new state is accepted. In cases where the increase in energy is very high, or the temperature is very low, this means that very few states will be accepted that involve an increase in energy, as e(_dE/T) approaches zero.
- The fact that some steps are allowed that increase the energy of the system enables the process to escape from local minima, which means that simulated annealing often can be an extremely powerful method for solving complex problems with many local maxima.
- Some systems use e(_dE/kT) as the probability that the search will progress to a state with a higher energy, where k is Boltzmann's constant (Boltzmann's constant is approximately 1.3807 _ 10_23 Joules per Kelvin).
- Simulated annealing usesMonte Carlo simulation to identify the most stable state (the state with the lowest energy) for a system.
- This is done by running successive iterations of metropolis Monte Carlo simulation, using progressively lower temperatures. Hence, in successive iterations, fewer and fewer steps are allowed that lead to an overall increase in energy for the system.
- A cooling schedule (or annealing schedule) is applied, which determines the manner in which the temperature will be lowered for successive iterations.
- Two popular cooling schedules are as follows:

  Tnew = Told _ dT

  Tnew = C _ Told (where C < 1.0)

- The cooling schedule is extremely important, as is the choice of the number of steps of metropolis Monte Carlo simulation that are applied in each iteration.
- These help to determine whether the system will be trapped by local minima (known as quenching). The number of times the metropolis Monte Carlo simulation is applied per iteration is for later iterations.
- Also important in determining the success of simulated annealing are the choice of the initial temperature of the system and the amount by which the temperature is decreased for each iteration.
- These values need to be chosen carefully according to the nature of the problem being solved. When the temperature, T, has reached zero, the system is frozen, and if the simulated annealing process has been successful, it will have identified a minimum for the total energy of the system.
- Simulated annealing has a number of practical applications in solving problems with large numbers of interdependent variables, such as circuit design.
- It has also been successfully applied to the traveling salesman problem.
- **Uses of Simulated Annealing**
  - Simulated annealing was invented in 1983 by Kirkpatrick, Gelatt, and Vecchi.
  - It was first used for placing VLSI* components on a circuit board.
  - Simulated annealing has also been used to solve the traveling salesman problem, although this approach has proved to be less efficient than using heuristic methods that know more about the problem.
  - It has been used much more successfully in scheduling problems and other large combinatorial problems where values need to be assigned to a large number of variables to maximize (or minimize) some function of those variables.

Real-Time A*

- Real-time A* is a variation of A*.
- Search continues on the basis of choosing paths that have minimum values of f(node) = g(node) + h(node). However, g(node) is the distance of the node from the current node, rather than from the root node.
- Hence, the algorithm will backtrack if the cost of doing so plus the estimated cost of solving the problem from the new node is less than the estimated cost of solving the problem from the current node.

- Implementing real-time A* means maintaining a hash table of previously visited states with their h(node) values.

Iterative-Deepening A* (IDA*)

- By combining iterative-deepening with A*, we produce an algorithm that is optimal and complete (like A*) and that has the low memory requirements of depth-first search.
- IDA* is a form of iterative-deepening search where successive iterations impose a greater limit on f(node) rather than on the depth of a node.
- IDA* performs well in problems where the heuristic value f (node) has relatively few possible values.
- For example, using the Manhattan distance as a heuristic in solving the eight-queens problem, the value of f (node) can only have values 1, 2, 3, or 4.
- In this case, the IDA* algorithm only needs to run through a maximum of four iterations, and it has a time complexity not dissimilar from that of A*, but with a significantly improved space complexity because it is effectively running depth-first search.
- In cases such as the traveling salesman problem where the value of f (node) is different for every state, the IDA* method has to expand 1 + 2 + 3 + . . . + n nodes = O(n2) where A* would expand n nodes.

Propositional and Predicate Logic

Logic is concerned with reasoning and the validity of arguments. In general, in logic, we are not concerned with the truth of statements, but rather with their validity. That is to say, although the following argument is clearly logical, it is not something that we would consider to be true:

All lemons are blue

Mary is a lemon

Therefore, Mary is blue

This set of statements is considered to be valid because the conclusion (Mary is blue) follows logically from the other two statements, which we often call the premises. The reason that validity and truth can be separated in this way is simple: a piece of a reasoning is considered to be valid if its conclusion is true in cases where its premises are also true. Hence, a valid set of statements such as the ones above can give a false conclusion, provided one or more of the premises are also false.

We can say: a piece of reasoning is valid if it leads to a true conclusion in every situation where the premises are true.

Logic is concerned with truth values. The possible truth values are true and false. These can be considered to be the fundamental units of logic, and almost all logic is ultimately concerned with these truth values.

Logic is widely used in computer science, and particularly in Artificial Intelligence. Logic is widely used as a representational method for Artificial Intelligence. Unlike some other representations, logic allows us to easily reason about negatives (such as, “this book is not red”) and disjunctions (“or”—such as, “He’s either a soldier or a sailor”).

Logic is also often used as a representational method for communicating concepts and theories within the Artificial Intelligence community. In addition, logic is used to represent language in systems that are able to understand and analyze human language.

As we will see, one of the main weaknesses of traditional logic is its inability to deal with uncertainty. Logical statements must be expressed in terms of truth or falsehood—it is not possible to reason, in classical logic, about possibilities. We will see different versions of logic such as modal logics that provide some ability to reason about possibilities, and also probabilistic methods and fuzzy logic that provide much more rigorous ways to reason in uncertain situations.

Logical Operators

- In reasoning about truth values, we need to use a number of operators, which can be applied to truth values.
- We are familiar with several of these operators from everyday language:

  I like apples and oranges.

  You can have an ice cream or a cake.

If you come from France, then you speak French.

- Here we see the four most basic logical operators being used in everyday language. The operators are:
  - and
  - or
  - not
  - if . . . then . . . (usually called implies)
- One important point to note is that or is slightly different from the way we usually use it. In the sentence, “You can have an icecream or a cake,” the mother is usually suggesting to her child that he can only have one of the items, but not both. This is referred to as an exclusive-or in logic because the case where both are allowed is excluded.
- The version of or that is used in logic is called inclusive-or and allows the case with both options.
- The operators are usually written using the following symbols, although other symbols are sometimes used, according to the context:

  and $\wedge$

  or $\vee$

  not $\neg$

  implies $\rightarrow$

  iff $\leftrightarrow$

- Iff is an abbreviation that is commonly used to mean “if and only if.”
- We see later that this is a stronger form of implies that holds true if one thing implies another, and also the second thing implies the first.
- For example, “you can have an ice-cream if and only if you eat your dinner.” It may not be immediately apparent why this is different from “you can have an icecream if you eat your dinner.” This is because most mothers really mean iff when they use if in this way.

Translating between English and Logic Notation

- To use logic, it is first necessary to convert facts and rules about the real world into logical expressions using the logical operators
- Without a reasonable amount of experience at this translation, it can seem quite a daunting task in some cases.
- Let us examine some examples. First, we will consider the simple operators, $\wedge$, $\vee$, and $\neg$.
- Sentences that use the word and in English to express more than one concept, all of which is true at once, can be easily translated into logic using the AND operator, $\wedge$.
- For example: “It is raining and it is Tuesday.” might be expressed as: $R \wedge T$, Where R means “it is raining” and T means “it is Tuesday.”
- For example, if it is not necessary to discuss where it is raining, R is probably enough.
- If we need to write expressions such as “it is raining in New York” or “it is raining heavily” or even “it rained for 30 minutes on Thursday,” then R will probably not suffice. To express more complex concepts like these, we usually use predicates. Hence, for example, we might translate “it is raining in New York” as: N(R) We might equally well choose to write it as: R(N)
- This depends on whether we consider the rain to be a property of New York, or vice versa. In other words, when we write N(R), we are saying that a property of the rain is that it is in New York, whereas with R(N) we are saying that a property of New York is that it is raining. Which we use depends on the problem we are solving. It is likely that if we are solving a problem about New York, we would use R(N), whereas if we are solving a problem about the location of various types of weather, we might use N(R).
- Let us return nowto the logical operators. The expression “it is raining inNew York, and I’meither getting sick or just very tired”can be expressed as follows: $R(N) \wedge (S(I) \vee T(I))$
- Here we have used both the $\wedge$ operator, and the $\vee$ operator to express a collection of statements. The statement can be broken down into two sections, which is indicated by the use of parentheses.

- The section in the parentheses is $S(I) \vee T(I)$, which means “I’m either getting sick OR I’m very tired”. This expression is “AND’ed”with the part outside the parentheses, which is R(N).
- Finally, the $\neg$ operator is applied exactly as you would expect—to express negation.
- For example, It is not raining in New York, might be expressed as $\neg R(N)$
- It is important to get the $\neg$ in the right place. For example: “I’m either not well or just very tired” would be translated as $\neg W(I) \vee T(I)$
- The position of the $\neg$ here indicates that it is bound to W(I) and does not play any role in affecting T(I).
- Now let us see how the →□operator is used. Often when dealing with logic we are discussing rules, which express concepts such as “if it is raining then I will get wet.”
- This sentence might be translated into logic as $R \rightarrow W(I)$
- This is read “R implies W(I)” or “IF R THEN W(I)”. By replacing the symbols R and W(I) with their respective English language equivalents, we can see that this sentence can be read as “raining implies I’ll get wet” or “IF it’s raining THEN I’ll get wet.”
- Implication can be used to express much more complex concepts than this.
- For example, “Whenever he eats sandwiches that have pickles in them, he ends up either asleep at his desk or singing loud songs” might be translated as

$$S(y) \wedge E(x, y) \wedge P(y) \rightarrow A(x) \vee (S(x, z) \wedge L(z))$$

- Here we have used the following symbol translations: S(y) means that y is a sandwich. E(x, y) means that x (the man) eats y (the sandwich).

  P(y) means that y (the sandwich) has pickles in it.

  A(x) means that x ends up asleep at his desk.

  S(x, z) means that x (the man) sings z (songs).

  L(z) means that z (the songs) are loud.

- The important thing to realize is that the choice of variables and predicates is important, but that you can choose any variables and predicates that map well to your problem and that help you to solve the problem.
- For example, in the example we have just looked at, we could perfectly well have used instead $S \rightarrow A \vee L$ where S means “he eats a sandwich which has pickles in it,” A means “he ends up asleep at his desk,” and L means “he sings loud songs.”
- The choice of granularity is important, but there is no right or wrong way to make this choice. In this simpler logical expression, we have chosen to express a simple relationship between three variables, which makes sense if those variables are all that we care about—in other words, we don’t need to know anything else about the sandwich, or the songs, or the man, and the facts we examine are simply whether or not he eats a sandwich with pickles, sleeps at his desk, and sings loud songs.
- The first translation we gave is more appropriate if we need to examine these concepts in more detail and reason more deeply about the entities involved.
- Note that we have thus far tended to use single letters to represent logical variables. It is also perfectly acceptable to use longer variable names, and thus to write expressions such as the following:

$$\text{Fish}(x) \wedge \text{living}(x) \rightarrow \text{has\_scales}(x)$$

- This kind of notation is obviously more useful when writing logical expressions that are intended to be read by humans but when manipulated by a computer do not add any value.

Truth Tables

- We can use variables to represent possible truth values, in much the same way that variables are used in algebra to represent possible numerical values.
- We can then apply logical operators to these variables and can reason about the way in which they behave.
- It is usual to represent the behavior of these logical operators using truth tables.
- A truth table shows the possible values that can be generated by applying an operator to truth values.
- **Not**

- First of all, we will look at the truth table for not, ¬ .
- Not is a unary operator, which means it is applied only to one variable.
- Its behavior is very simple:

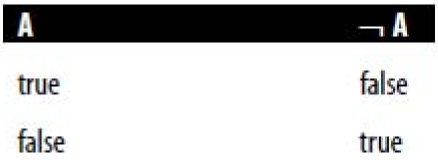
¬ true is equal to false

¬ false is equal to true

If variable A has value true, then ¬ A has value false.

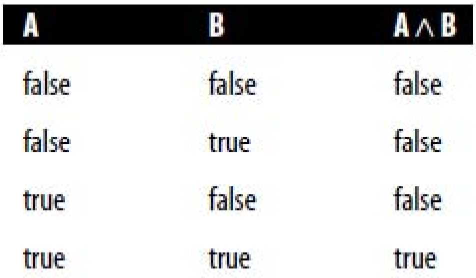
If variable B has value false, then ¬ B has value true.

- These can be represented by a truth table,



A	¬ A
true	false
false	true

- **And**
  - Now, let us examine the truth table for our first binary operator—one which acts on two variables:

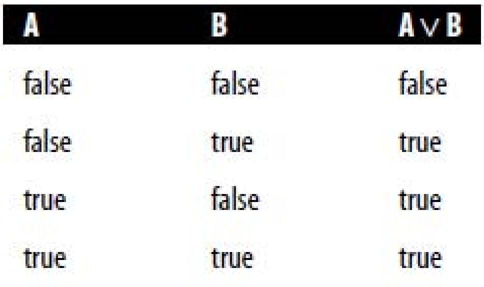


A	B	A ∧ B
false	false	false
false	true	false
true	false	false
true	true	true

  - ∧ is also called the conjunctive operator.
  - A ∧ B is the conjunction of A and B.
  - You can see that the only entry in the truth table for which A ∧ B is true is the one where A is true and B is true. If A is false, or if B is false, then A ∧ B is false. If both A and B are false, then A ∧ B is also false.
  - What do A and B mean? They can represent any statement, or proposition, that can take on a truth value.

  - For example, A might represent “It’s sunny,” and B might represent “It’s warm outside.” In this case, A ∧ B would mean “It is sunny and it’s warm outside,” which clearly is true only if the two component parts are true (i.e., if it is true that it is sunny and it is true that it is warm outside).

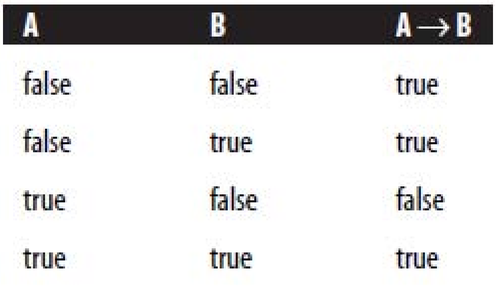
- **Or**
  - The truth table for the or operator, ∨



A	B	A ∨ B
false	false	false
false	true	true
true	false	true
true	true	true

  - ∨ is also called the disjunctive operator.
  - A ∨ B is the disjunction of A and B.
  - Clearly A ∨ B is true for any situation except when both A and B are false.
  - If A is true, or if B is true, or if both A and B are true, A ∨ B is true.
  - This table represents the inclusive-or operator.
  - A table to represent exclusive-or would have false in the final row. In other words, while A ∨ B is true if A and B are both true, A EOR B (A exclusive-or B) is false if A and B are both true.
  - You may also notice a pleasing symmetry between the truth tables for ∧ and ∨. This will become useful later, as will a number of other symmetrical relationships.

- **Implies**
  - The truth table for implies (→) is a little less intuitive.



A	B	A → B
false	false	true
false	true	true
true	false	false
true	true	true

  - This form of implication is also known as material implication
  - In the statement A→B, A is the antecedent, and B is the consequent. The bottom two lines of the table should be obvious. If A is true and B is true, then A →□B seems to be a reasonable thing to believe.
  - For example, if A means “you live in France” and B means “You speak French,” then A→B corresponds to the statement “if you live in France, then you speak French.”
  - Clearly, this statement is true (A→B is true) if I live in France and I speak French (A is true and B is true).
  - Similarly, if I live in France, but I don’t speak French (A is true, but B is false), then it is clear that A→B is not true.
  - The situations where A is false are a little less clear. If I do not live in France (A is not true), then the truth table tells us that regardless of whether I speak French or not (the value of B), the statement A→B is true. A→B is usually read as “A implies B” but can also be read as “If A then B” or “If A is true then B is true.”
  - Hence, if A is false, the statement is not really saying anything about the value of B, so B is free to take on any value (as long as it is true or false, of course!).
  - All of the following statements are valid:

    52 = 25 →4 = 4 (true →true)

    9 _ 9 = 123 →8 > 3 (false →true)

    52 = 25 →0 = 2 (false →false)

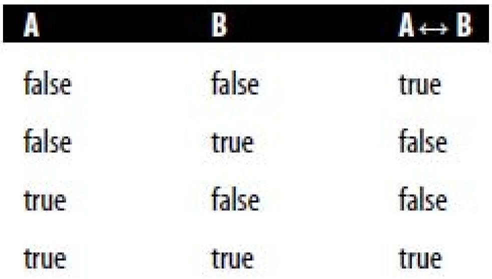
  - In fact, in the second and third examples, the consequent could be given any meaning, and the statement would still be true. For example, the following statement is valid:

    52 = 25 →Logic is weird

  - Notice that when looking at simple logical statements like these, there does not need to be any real-world relationship between the antecedent and the consequent.
  - For logic to be useful, though, we tend to want the relationships being expressed to be meaningful as well as being logically true.

- iff

- The truth table for iff (if and only if {↔}) is as follows:

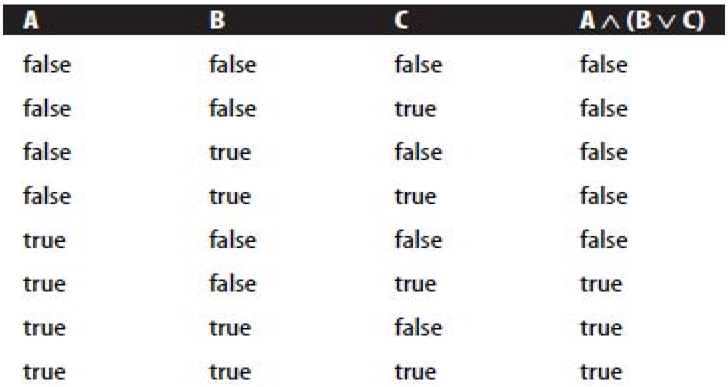


A	B	A↔B
false	false	true
false	true	false
true	false	false
true	true	true

- It can be seen that A ↔ B is true as long as A and B have the same value.
- In other words, if one is true and the other false, then A ↔ B is false. Otherwise, if A and B have the same value, A↔ B is true.

Complex Truth Tables

- Truth tables are not limited to showing the values for single operators.
- For example, a truth table can be used to display the possible values for A ∧(B ∨C).



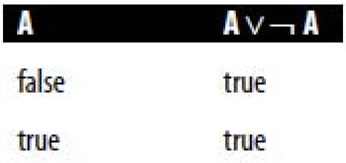
A	B	C	A ∧ (B ∨ C)
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	false
true	false	true	true
true	true	false	true
true	true	true	true

- Note that for two variables, the truth table has four lines, and for three variables, it has eight. In general, a truth table for n variables will have 2^n lines.
- The use of brackets in this expression is important. A ∧(B ∨C) is not the same as (A ∧B) ∨C.
- To avoid ambiguity, the logical operators are assigned precedence, as with mathematical operators.
- The order of precedence that is used is as follows: ¬ , ∧, ∨,→,↔

- Hence, in a statement such as $\neg A \lor \neg B \land C$, the $\neg$ operator has the greatest precedence, meaning that it is most closely tied to its symbols. $\land$ has a greater precedence than $\lor$, which means that the sentence above can be expressed as $(\neg A) \lor ((\neg B) \land C)$
- Similarly, when we write $\neg A \lor B$ this is the same as $(\neg A) \lor B$ rather than $\neg (A \lor B)$
- In general, it is a good idea to use brackets whenever an expression might otherwise be ambiguous.

Tautology

- Consider the following truth table:



A	$A \lor \neg A$
false	true
true	true

- This truth table has a property that we have not seen before: the value of the expression $A \lor \neg A$ is true regardless of the value of A.
- An expression like this that is always true is called a tautology.
- If A is a tautology, we write: $\models A$
- A logical expression that is a tautology is often described as being valid.
- A valid expression is defined as being one that is true under any interpretation.
- In other words, no matter what meanings and values we assign to the variables in a valid expression, it will still be true.
- For example, the following sentences are all valid:

  If wibble is true, then wibble is true.

  Either wibble is true, or wibble is not true.

- In the language of logic, we can replace wibble with the symbol A, in which case these two statements can be rewritten as

  $A \rightarrow A$

A ∨ ¬ A

- If an expression is false in any interpretation, it is described as being contradictory.
- The following expressions are contradictory:

A ∧ ¬ A

(A ∨ ¬ A)→(A ∧ ¬ A)

Equivalence

- Consider the following two expressions:

A ∧ B

B ∧ A

- It should be fairly clear that these two expressions will always have the same value for a given pair of values for A and B.
- In otherwords, we say that the first expression is logically equivalent to the second expression.
- We write this as A ∧ B _ B ∧ A. This means that the ∧ operator is commutative.
- Note that this is not the same as implication: A ∧ B→B ∧ A, although this second statement is also true.
- The difference is that if for two expressions e1 and e2: e1 _ e2, then e1 will always have the same value as e2 for a given set of variables.
- On the other hand, as we have seen, e1→e2 is true if e1 is false and e2 is true.
- There are a number of logical equivalences that are extremely useful.
- The following is a list of a few of the most common:

A ∨ A _ A

A ∧ A _ A

A ∧ (B ∧ C) _ (A ∧ B) ∧ C (∧ is associative)

A ∨ (B ∨ C) _ (A ∨ B) ∨ C (∨ is associative)

A ∧ (B ∨ C) _ (A ∧ B) ∨ (A ∧ C) (∧ is distributive over ∨)

$A \land (A \lor B) \_ A$

$A \lor (A \land B) \_ A$

$A \land \text{true} \_ A$

$A \land \text{false} \_ \text{false}$

$A \lor \text{true} \_ \text{true}$

$A \lor \text{false} \_ A$

- All of these equivalences can be proved by drawing up the truth tables for each side of the equivalence and seeing if the two tables are the same.
- The following is a very important equivalence: $A \rightarrow B \_ \neg A \lor B$
- We do not need to use the →□ symbol at all—we can replace it with a combination of ¬ and ∨.
- Similarly, the following equivalences mean we do not need to use ∧ or↔:

  $A \land B \_ \neg(\neg A \lor \neg B)$

  $A \leftrightarrow B \_ \neg(\neg(\neg A \lor B) \lor \neg(\neg B \lor A))$

- In fact, any binary logical operator can be expressed using ¬ and ∨. This is a fact that is employed in electronic circuits, where nor gates, based on an operator called nor, are used. Nor is represented by ↓, and is defined as follows:

  $A \downarrow B \_ \neg(A \lor B)$

- Finally, the following equivalences are known as DeMorgan's Laws:

  $A \land B \_ \neg(\neg A \lor \neg B)$

  $A \lor B \_ \neg(\neg A \land \neg B)$

- By using these and other equivalences, logical expressions can be simplified.
- For example, $(C \land D) \lor ((C \land D) \land E)$ can be simplified using the following rule: $A \lor (A \land B) \_ A$ hence, $(C \land D) \lor ((C \land D) \land E) \_ C \land D$

- In this way, it is possible to eliminate subexpressions that do not contribute to the overall value of the expression.

Propositional Logic

- There are a number of possible systems of logic.
- The system we have been examining so far is called propositional logic.
- The language that is used to express propositional logic is called the propositional calculus.
- A logical system can be defined in terms of its syntax (the alphabet of symbols and how they can be combined), its semantics (what the symbols mean), and a set of rules of deduction that enable us to derive one expression from a set of other expressions and thus make arguments and proofs.
- **Syntax**
  - We have already examined the syntax of propositional calculus. The alphabet of symbols, _ is defined as follows

    $\sum$ = {true, false, ¬ ,→, (,), ∧ , ∨ ,↔, p1, p2, p3, . . . , pn, . . . }

  - Here we have used set notation to define the possible values that are contained within the alphabet $\sum$.
  - Note that we allow an infinite number of proposition letters, or propositional symbols, p1, p2, p3, . . . , and so on.
  - More usually, we will represent these by capital letters P, Q, R, and so on,
  - If we need to represent a very large number of them, we will use the subscript notation (e.g., p1).
  - An expression is referred to as a well-formed formula (often abbreviated as wff) or a sentence if it is constructed correctly, according to the rules of the syntax of propositional calculus, which are defined as follows.
  - In these rules, we use A, B, C to represent sentences. In other words, we define a sentence recursively, in terms of other sentences.
  - The following are wellformed sentences:

    P,Q,R. . .

    true, false

(A)

$\neg A$

$A \wedge B$

$A \vee B$

$A \rightarrow B$

$A \leftrightarrow B$

- Hence, we can see that the following is an example of a wff:

  $P \wedge Q \vee (B \wedge \neg C) \rightarrow A \wedge B \vee D \wedge (\neg E)$

- **Semantics**
  - The semantics of the operators of propositional calculus can be defined in terms of truth tables.
  - The meaning of $P \wedge Q$ is defined as “true when P is true and Q is also true.”
  - The meaning of symbols such as P and Q is arbitrary and could be ignored altogether if we were reasoning about pure logic.
  - In other words, reasoning about sentences such as $P \vee Q \wedge \neg R$ is possible without considering what P, Q, and R mean.
  - Because we are using logic as a representational method for artificial intelligence, however, it is often the case that when using propositional logic, the meanings of these symbols are very important.
  - The beauty of this representation is that it is possible for a computer to reason about them in a very general way, without needing to know much about the real world.
  - In other words, if we tell a computer, “I like ice cream, and I like chocolate,” it might represent this statement as $A \wedge B$, which it could then use to reason with, and, as we will see, it can use this to make deductions.

Predicate Calculus

- **Syntax**

- Predicate calculus allows us to reason about properties of objects and relationships between objects.
- In propositional calculus, we could express the English statement "I like cheese" by A. This enables us to create constructs such as ¬ A, which means "I do not like cheese," but it does not allow us to extract any information about the cheese, or me, or other things that I like.
- In predicate calculus, we use predicates to express properties of objects. So the sentence "I like cheese" might be expressed as L(me, cheese) where L is a predicate that represents the idea of "liking." Note that as well as expressing a property of me, this statement also expresses a relationship between me and cheese. This can be useful, as we will see, in describing environments for robots and other agents.
- For example, a simple agent may be concerned with the location of various blocks, and a statement about the world might be T(A,B), which could mean: Block A is on top of Block B.
- It is also possible to make more general statements using the predicate calculus.
- For example, to express the idea that everyone likes cheese, we might say $(\forall x)(P(x) \rightarrow L(x, C))$
- The symbol ∀ is read "for all," so the statement above could be read as "for every x it is true that if property P holds for x, then the relationship L holds between x and C," or in plainer English: "every x that is a person likes cheese." (Here we are interpreting P(x) as meaning "x is a person" or, more precisely, "x has property P.")
- Note that we have used brackets rather carefully in the statement above.
- This statement can also be written with fewer brackets: $\forall x\ P(x) \rightarrow L(x, C)$, ∀ is called the universal quantifier.
- The quantifier ∃ can be used to express the notion that some values do have a certain property, but not necessarily all of them: $(\exists x)(L(x,C))$
- This statement can be read "there exists an x such that x likes cheese."
- This does not make any claims about the possible values of x, so x could be a person, or a dog, or an item of furniture. When we use the existential

quantifier in this way, we are simply saying that there is at least one value of x for which L(x,C) holds.

- The following is true: $(\forall x)(L(x,C)) \rightarrow (\exists x)(L(x,C))$, but the following is not: $(\exists x)(L(x,C)) \rightarrow (\forall x)(L(x,C))$

- Relationships between ∀ and ∃
  - It is also possible to combine the universal and existential quantifiers, such as in the following statement: $(\forall x)\ (\exists y)\ (L(x,y))$.
  - This statement can be read "for all x, there exists a y such that L holds for x and y," which we might interpret as "everyone likes something."
  - A useful relationship exists between ∀ and ∃. Consider the statement "not everyone likes cheese." We could write this as

    $\neg(\forall x)(P(x) \rightarrow L(x,C))$ -------------- (1)

  - As we have already seen, $A \rightarrow B$ is equivalent to $\neg A \vee B$. Using DeMorgan's laws, we can see that this is equivalent to $\neg(A \wedge \neg B)$. Hence, the statement (1) above, can be rewritten:

    $\neg(\forall x)\neg(P(x) \wedge \neg L(x,C))$ ------------- (2)

  - This can be read as "It is not true that for all x the following is not true: x is a person and x does not like cheese." If you examine this rather convoluted sentence carefully, you will see that it is in fact the same as "there exists an x such that x is a person and x does not like cheese." Hence we can rewrite it as

    $(\exists x)(P(x) \wedge \neg L(x,C))$ ------------- (3)

  - In making this transition from statement (2) to statement (3), we have utilized the following equivalence: $\exists x \cong \neg(\forall x)\neg$
  - In an expression of the form $(\forall x)(P(x, y))$, the variable x is said to be bound, whereas y is said to be free. This can be understood as meaning that the variable y could be replaced by any other variable because it is free, and the expression would still have the same meaning, whereas if the variable x were to be replaced by some other variable in P(x,y), then the meaning of the

expression would be changed: $(\forall x)(P(y, z))$ is not equivalent to $(\forall x)(P(x, y))$, whereas $(\forall x)(P(x, z))$ is.

- Note that a variable can occur both bound and free in an expression, as in $(\forall x)(P(x,y,z) \rightarrow (\exists y)(Q(y,z)))$
- In this expression, x is bound throughout, and z is free throughout; y is free in its first occurrence but is bound in $(\exists y)(Q(y,z))$. (Note that both occurrences of y are bound here.)
- Making this kind of change is known as substitution.
- Substitution is allowed of any free variable for another free variable.

- **Functions**
  - In much the same way that functions can be used in mathematics, we can express an object that relates to another object in a specific way using functions.
  - For example, to represent the statement “my mother likes cheese,” we might use L(m(me),cheese)
  - Here the function m(x) means the mother of x. Functions can take more than one argument, and in general a function with n arguments is represented as $f(x1, x2, x3, . . . , x_n)$

First-Order Predicate Logic

- The type of predicate calculus that we have been referring to is also called firstorder predicate logic (FOPL).
- A first-order logic is one in which the quantifiers ∀ and ∃ can be applied to objects or terms, but not to predicates or functions.
- So we can define the syntax of FOPL as follows. First,we define a term:
- A constant is a term.
- A variable is a term. f(x1, x2, x3, . . . , xn) is a term if x1, x2, x3, . . . , xn are all terms.
- Anything that does not meet the above description cannot be a term.
- For example, the following is not a term: $\forall x\ P(x)$. This kind of construction we call a sentence or a well-formed formula (wff), which is defined as follows.

- In these definitions, P is a predicate, x1, x2, x3, . . . , xn are terms, and A,B are wff ’s. The following are the acceptable forms for wff ’s:

  P(x1, x2, x3, . . . , xn)

  $\neg A$

  $A \wedge B$

  $A \vee B$

  $A \rightarrow B$

  $A \leftrightarrow B$

  $(\forall x)A$

  $(\exists x)A$

- An atomic formula is a wff of the form P(x1, x2, x3, . . . , xn).
- Higher order logics exist in which quantifiers can be applied to predicates and functions, and where the following expression is an example of a wff:

  $(\forall P)(\exists x)P(x)$

Soundness

- We have seen that a logical system such as propositional logic consists of a syntax, a semantics, and a set of rules of deduction.
- A logical system also has a set of fundamental truths, which are known as axioms.
- The axioms are the basic rules that are known to be true and from which all other theorems within the system can be proved.
- An axiom of propositional logic, for example, is $A \rightarrow (B \rightarrow A)$
- A theorem of a logical system is a statement that can be proved by applying the rules of deduction to the axioms in the system.
- If A is a theorem, then we write $\vdash A$
- A logical system is described as being sound if every theorem is logically valid, or a tautology.
- It can be proved by induction that both propositional logic and FOPL are sound.
- **Completeness**

    - A logical system is complete if every tautology is a theorem—in other words, if every valid statement in the logic can be proved by applying the rules of deduction to the axioms. Both propositional logic and FOPL are complete.

- **Decidability**
    - A logical system is decidable if it is possible to produce an algorithm that will determine whether any wff is a theorem. In other words, if a logical system is decidable, then a computer can be used to determine whether logical expressions in that system are valid or not.
    - We can prove that propositional logic is decidable by using the fact that it is complete.
    - We can prove that a wff A is a theorem by showing that it is a tautology. To show if a wff is a tautology, we simply need to draw up a truth table for that wff and show that all the lines have true as the result. This can clearly be done algorithmically because we know that a truth table for n values has 2n lines and is therefore finite, for a finite number of variables.
    - FOPL, on the other hand, is not decidable. This is due to the fact that it is not possible to develop an algorithm that will determine whether an arbitrary wff in FOPL is logically valid.
- **Monotonicity**
    - A logical system is described as being monotonic if a valid proof in the system cannot be made invalid by adding additional premises or assumptions.
    - In other words, if we find that we can prove a conclusion C by applying rules of deduction to a premise B with assumptions A, then adding additional assumptions A¬ and B¬ will not stop us from being able to deduce C.
    - Monotonicity of a logical system can be expressed as follows:

      If we can prove {A, B} ├ C,

      then we can also prove: {A, B, A_, B_} ├ C.
    - In other words, even adding contradictory assumptions does not stop us from making the proof in a monotonic system.
    - In fact, it turns out that adding contradictory assumptions allows us to prove anything, including invalid conclusions. This makes sense if we recall the line in

the truth table for →, which shows that false → true. By adding a contradictory assumption, we make our assumptions false and can thus prove any conclusion.

Modal Logics and Possible Worlds

- The forms of logic that we have dealt with so far deal with facts and properties of objects that are either true or false.
- In these classical logics, we do not consider the possibility that things change or that things might not always be as they are now.
- Modal logics are an extension of classical logic that allow us to reason about possibilities and certainties.
- In other words, using a modal logic, we can express ideas such as "although the sky is usually blue, it isn't always" (for example, at night). In this way, we can reason about possible worlds.
- A possible world is a universe or scenario that could logically come about.
- The following statements may not be true in our world, but they are possible, in the sense that they are not illogical, and could be true in a possible world:

  Trees are all blue.

  Dogs can fly.

  People have no legs.

- It is possible that some of these statements will become true in the future, or even that they were true in the past.
- It is also possible to imagine an alternative universe in which these statements are true now.
- The following statements, on the other hand, cannot be true in any possible world:

  $A \land \neg A$

  $(x > y) \land (y > z) \land (z > x)$

- The first of these illustrates the law of the excluded middle, which simply states that a fact must be either true or false: it cannot be both true and false.
- It also cannot be the case that a fact is neither true nor false. This is a law of classical logic, it is possible to have a logical system without the law of the excluded middle, and in which a fact can be both true and false.

- The second statement cannot be true by the laws of mathematics. We are not interested in possible worlds in which the laws of logic and mathematics do not hold.
- A statement that may be true or false, depending on the situation, is called contingent.
- A statement that must always have the same truth value, regardless of which possible world we consider, is noncontingent.
- Hence, the following statements are contingent:

  $A \wedge B$

  $A \vee B$

  I like ice cream.

  The sky is blue.

- The following statements are noncontingent:

  $A \vee \neg A$

  $A \wedge \neg A$

  If you like all ice cream, then you like this ice cream.

- Clearly, a noncontingent statement can be either true or false, but the fact that it is noncontingent means it will always have that same truth value.
- If a statement A is contingent, then we say that A is possibly true, which is written ◊ A
- If A is noncontingent, then it is necessarily true, which is written □ A
- **Reasoning in Modal Logic**
  - It is not possible to draw up a truth table for the operators ◊ and □
  - The following rules are examples of the axioms that can be used to reason in this kind of modal logic:

    $\Box A \rightarrow \Diamond A$

    $\Box \neg A \rightarrow \neg \Diamond A$

    $\Diamond A \rightarrow \neg \Box A$

- ➢ Although truth tables cannot be drawn up to prove these rules, you should be able to reason about them using your understanding of the meaning of the ◊ and □ operators.

Possible world representations

- It describes method proposed by Nilsson which generalizes firtst order logic in the modeling of uncertain beliefs
- The method assigns truth values ranging from 0 to 1 to possible worlds
- Each set of possible worlds corresponds to a different interpretation of sentences contained in a knowledge base denoted as KB
- Consider the simple case where a KB contains only the single sentence S, S may be either true or false. We envision S as being true in one set of possible worlds W_1 and false in another set W_2 . The actual world , the one we are in, must be in one of the two sets, but we are uncertain which one. Uncertainty is expressed by assigning a probability P to W_1 and 1 – P to $W_{2.}$ We can say then that the probability of S being true is P
- When KB contains L sentences, $S_{1,...}$ S_L , more sets of possible worlds are required to represent all consistent truth value assignments. There are 2^L possible truth assignments for L sentences.
- Truth Value assignments for the set {P. P→Q, Q}

Consistent			Inconsistent		
P	Q	P → Q	P	Q	P → Q
True	True	True	True	True	False
True	False	False	True	False	True
False	True	True	False	True	False
False	False	True	False	False	False

- They are based on the use of the probability constraints

  $0 \leq p_i \leq 1$, and $\sum_i p_i = 1$

- The consistent probability assignments are bounded by the hyperplanes of a certain convex hull

Dempster- Shafer theory

- The Dempster-Shafer theory, also known as the theory of belief functions, is a generalization of the Bayesian theory of subjective probability.
- Whereas the Bayesian theory requires probabilities for each question of interest, belief functions allow us to base degrees of belief for one question on probabilities for a related question. These degrees of belief may or may not have the mathematical properties of probabilities;
- The Dempster-Shafer theory owes its name to work by A. P. Dempster (1968) and Glenn Shafer (1976), but the theory came to the attention of AI researchers in the early 1980s, when they were trying to adapt probability theory to expert systems.
- Dempster-Shafer degrees of belief resemble the certainty factors in MYCIN, and this resemblance suggested that they might combine the rigor of probability theory with the flexibility of rule-based systems.
- The Dempster-Shafer theory remains attractive because of its relative flexibility.
- The Dempster-Shafer theory is based on two ideas:
    - the idea of obtaining degrees of belief for one question from subjective probabilities for a related question,
    - Dempster's rule for combining such degrees of belief when they are based on independent items of evidence.
- To illustrate the idea of obtaining degrees of belief for one question from subjective probabilities for another, suppose I have subjective probabilities for the reliability of my friend Betty. My probability that she is reliable is 0.9, and my probability that she is unreliable is 0.1. Suppose she tells me a limb fell on my car. This statement, which must true if she is reliable, is not necessarily false if she is unreliable. So her testimony alone justifies a 0.9 degree of belief that a limb fell on my car, but only a zero degree of belief (not a 0.1 degree of belief) that no limb fell on my car. This zero does not mean that I am sure that no limb fell on my car, as a zero probability would; it merely means that Betty's testimony gives me no reason to believe that no limb fell on my car. The 0.9 and the zero together constitute a belief function.
- To illustrate Dempster's rule for combining degrees of belief, suppose I also have a 0.9 subjective probability for the reliability of Sally, and suppose she too testifies,

independently of Betty, that a limb fell on my car. The event that Betty is reliable is independent of the event that Sally is reliable, and we may multiply the probabilities of these events; the probability that both are reliable is 0.9x0.9 = 0.81, the probability that neither is reliable is 0.1x0.1 = 0.01, and the probability that at least one is reliable is 1 - 0.01 = 0.99. Since they both said that a limb fell on my car, at least of them being reliable implies that a limb did fall on my car, and hence I may assign this event a degree of belief of 0.99. Suppose, on the other hand, that Betty and Sally contradict each other—Betty says that a limb fell on my car, and Sally says no limb fell on my car. In this case, they cannot both be right and hence cannot both be reliable—only one is reliable, or neither is reliable. The prior probabilities that only Betty is reliable, only Sally is reliable, and that neither is reliable are 0.09, 0.09, and 0.01, respectively, and the posterior probabilities (given that not both are reliable) are 9 19 , 9 19 , and 1 19 , respectively. Hence we have a 9 19 degree of belief that a limb did fall on my car (because Betty is reliable) and a 9 19 degree of belief that no limb fell on my car (because Sally is reliable).

- In summary, we obtain degrees of belief for one question (Did a limb fall on my car?) from probabilities for another question (Is the witness reliable?). Dempster's rule begins with the assumption that the questions for which we have probabilities are independent with respect to our subjective probability judgments, but this independence is only a priori; it disappears when conflict is discerned between the different items of evidence.
- Implementing the Dempster-Shafer theory in a specific problem generally involves solving two related problems.
  - ➢ First, we must sort the uncertainties in the problem into a priori independent items of evidence.
  - ➢ Second, we must carry out Dempster's rule computationally. These two problems and their solutions are closely related.
- Sorting the uncertainties into independent items leads to a structure involving items of evidence that bear on different but related questions, and this structure can be used to make computations
- This can be regarded as a more general approach to representing uncertainty than the Bayesian approach.
- The basic idea in representing uncertainty in this model is:

  - Set up a confidence interval -- an interval of probabilities within which the true probability lies with a certain confidence -- based on the Belief *B* and plausibility *PL* provided by some evidence *E* for a proposition *P*.
  - The belief brings together all the evidence that would lead us to believe in *P* with some certainty.
  - The plausibility brings together the evidence that is compatible with *P* and is not inconsistent with it.
  - This method allows for further additions to the set of knowledge and does not assume disjoint outcomes.

- If $\Omega$is the set of possible outcomes, then a *mass probability*, *M*, is defined for each member of the set $2^{\Omega}$and takes values in the range [0,1].

  The Null set, $\emptyset$, is also a member of 2^{Ω}.

- **NOTE:** This deals wit set theory terminology that will be dealt with in a tutorial shortly. Also see exercises to get experience of problem solving in this important subject matter.

  M is a *probability density function* defined not just for Ωbut for **em all** subsets.

  So if $\Omega$is the set { *Flu (F), Cold (C), Pneumonia (P)* } then $2^{\Omega}$is the set { $\emptyset$, {*F*}, {*C*}, {*P*}, {*F, C*}, {*F, P*}, {*C, P*}, {*F, C, P*} }

- The confidence interval is then defined as [*B*(*E*),*PL*(*E*)] where

$$B(E) = \sum_{A} M$$

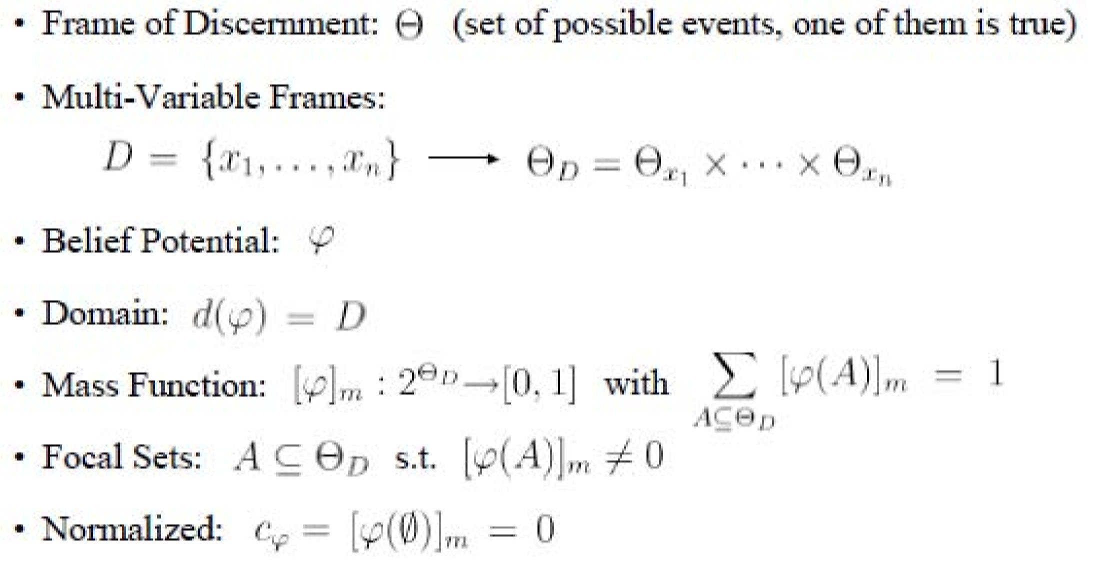
where $A \subseteq E$ *i.e.* all the evidence that makes us believe in the correctness of *P*, and

$$\begin{aligned} PL(E) &= 1 - B(\neg E) \\ &= 1 - \sum_{\neg A} M \end{aligned}$$

where $\neg A \subseteq \neg E$ *i.e.* all the evidence that contradicts *P*.

- Let X be the universal set: the set of all states under consideration. The power set is the set of all possible sub-sets of X, including the empty set . For example, if: X = {a,b}then 2^{x} = {Ǿ, {a},{b}, X}
- The elements of the power set can be taken to represent propositions that one might be interested in, by containing all and only the states in which this proposition is true.

- The theory of evidence assigns a belief mass to each element of the power set. Formally, a function m: $2^x \rightarrow [0, 1]$ is called a *basic belief assignment* (BBA), when it has two properties.
- First, the mass of the empty set is zero: m (Ǿ) = 0
- Second, the masses of the remaining members of the power set add up to a total of 1:
  $\sum$ m(A) = 1
  A€ 2^x
- The mass *m*(*A*) of a given member of the power set, *A*, expresses the proportion of all relevant and available evidence that supports the claim that the actual state belongs to *A* but to no particular subset of *A*. The value of *m*(*A*) pertains *only* to the set *A* and makes no additional claims about any subsets of *A*, each of which have, by definition, their own mass.
- From the mass assignments, the upper and lower bounds of a probability interval can be defined. This interval contains the precise probability of a set of interest (in the classical sense), and is bounded by two non-additive continuous measures called belief (or support) and plausibility:
  bel(A) ≤ P(A) ≤ pl(A)



- Frame of Discernment: Θ (set of possible events, one of them is true)
- Multi-Variable Frames:

$$D = \{x_1, \ldots, x_n\} \longrightarrow \Theta_D = \Theta_{x_1} \times \cdots \times \Theta_{x_n}$$

- Belief Potential: φ
- Domain: $d(\varphi) = D$
- Mass Function: $[\varphi]_m : 2^{\Theta_D} \rightarrow [0, 1]$ with $\sum_{A \subseteq \Theta_D} [\varphi(A)]_m = 1$
- Focal Sets: $A \subseteq \Theta_D$ s.t. $[\varphi(A)]_m \neq 0$
- Normalized: $c_\varphi = [\varphi(\emptyset)]_m = 0$

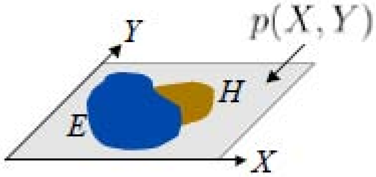
- Normalized Mass Function: $[\varphi(A)]_M \overset{def}{=} \begin{cases} 0, \text{ if } A = \emptyset, \\ \dfrac{[\varphi(A)]_m}{1 - c_\varphi}, \text{ otherwise,} \end{cases}$
- Belief Function: $[\varphi(A)]_B \overset{def}{=} \sum_{B \subseteq A} [\varphi(B)]_M = \sum_{\substack{B \subseteq A \\ B \in FS(\varphi)}} [\varphi(B)]_M$
- Plausibility Function: $[\varphi(A)]_P \overset{def}{=} \sum_{B \cap A \neq \emptyset} [\varphi(B)]_M = \sum_{\substack{B \cap A \neq \emptyset \\ B \in FS(\varphi)}} [\varphi(B)]_M$
- Properties: $[\varphi(\emptyset)]_B = [\varphi(\emptyset)]_P = 0$

  $[\varphi(\Theta_D)]_B = [\varphi(\Theta_D)]_P = 1$

  $[\varphi(A)]_B \leq [\varphi(A)]_P$ for all $A \subseteq \Theta_D$

  $[\varphi]_P(H) = 1 - [\varphi]_B(H)$

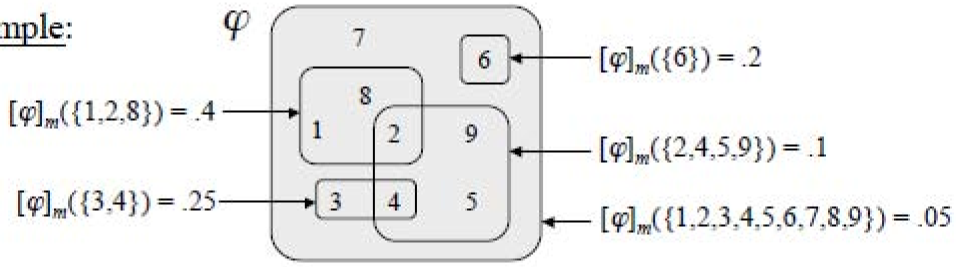
- Belief is Sub-Additive: $[\varphi]_B(H) + [\varphi]_B(H^c) \leq 1$
- Plausibility is Super-Additive: $[\varphi]_P(H) + [\varphi]_P(H^c) \geq 1$



Additive: $p(H|E) + p(H^c|E) = 1$

Sub-Additive: $p(H|E) + p(H^c|E) \leq 1$

- Example:


φ
7
6
8
1
2
9
3
4
5
$[\varphi]_m(\{6\}) = .2$
$[\varphi]_m(\{1,2,8\}) = .4$
$[\varphi]_m(\{2,4,5,9\}) = .1$
$[\varphi]_m(\{3,4\}) = .25$
$[\varphi]_m(\{1,2,3,4,5,6,7,8,9\}) = .05$

$H = \{1,2,3,4,7,8\}$ $\quad$ $H^c = \{5,6,9\}$

$[\varphi]_B(H) = .4+.25 = .65$ $\quad$ $[\varphi]_B(H^c) = .2$

$[\varphi]_P(H) = .4+.25+.1+.05 = .8$ $\quad$ $[\varphi]_P(H^c) = .2+.1+.05 = .35$

- Ignorance of φ relative to H: $\quad 0 \leq [\varphi]_P(H) - [\varphi]_B(H) \leq 1$

- Total ignorance: $\begin{cases} [\varphi]_B(H) = 0 \\ [\varphi]_P(H) = 1 \end{cases}$

- Dempster's Rule of Combination:

$$[\varphi_1 \otimes \varphi_2(A)]_m \overset{def}{=} \sum_{B \cap C = A} [\varphi_1(B)]_m \cdot [\varphi_2(C)]_m$$

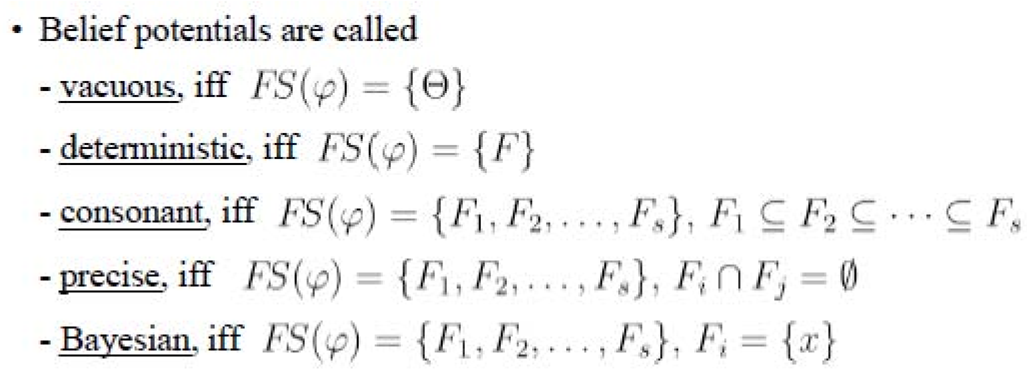
$$= \sum_{\substack{B \cap C = A \\ B \in FS(\varphi_1), C \in FS(\varphi_2)}} [\varphi_1(B)]_m \cdot [\varphi_2(C)]_m$$

- Non-Monotonicity: $\quad [\varphi \otimes \varphi']_B(H) \quad \begin{matrix} < \\ = \\ > \end{matrix} \quad [\varphi]_B(H)$

$$[\varphi \otimes \varphi']_P(H) \quad \begin{matrix} < \\ = \\ > \end{matrix} \quad [\varphi]_P(H)$$

- Marginalization: $\quad D' \subseteq D$

$$[\varphi^{\downarrow D'}(A)]_m \overset{def}{=} \sum_{B^{\downarrow D'} = A} [\varphi(B)]_m = \sum_{\substack{B^{\downarrow D'} = A \\ B \in FS(\varphi)}} [\varphi(B)]_m$$



- Belief potentials are called
  - vacuous, iff $FS(\varphi) = \{\Theta\}$
  - deterministic, iff $FS(\varphi) = \{F\}$
  - consonant, iff $FS(\varphi) = \{F_1, F_2, \ldots, F_s\}, F_1 \subseteq F_2 \subseteq \cdots \subseteq F_s$
  - precise, iff $FS(\varphi) = \{F_1, F_2, \ldots, F_s\}, F_i \cap F_j = \emptyset$
  - Bayesian, iff $FS(\varphi) = \{F_1, F_2, \ldots, F_s\}, F_i = \{x\}$

- Benefits of Dempster-Shafer Theory:
  - Allows proper distinction between reasoning and decision taking
  - No modeling restrictions (e.g. DAGs)
  - It represents properly partial and total ignorance
  - Ignorance is quantified:
    - low degree of ignorance means
      - high confidence in results
      - enough information available for taking decisions
    - high degree of ignorance means
      - low confidence in results
      - gather more information (if possible) before taking decisions
  - Conflict is quantified:
    - low conflict indicates the presence of confirming information sources
    - high conflict indicates the presence of contradicting sources
  - Simplicity: Dempster’s rule of combination covers
    - combination of evidence
    - Bayes’ rule
    - Bayesian updating (conditioning)
    - belief revision (results from non-monotonicity)
- DS-Theory is not very successful because:
  - Inference is less efficient than Bayesian inference
  - Pearl is the better speaker than Dempster (and Shafer, Kohlas, etc.)
  - Microsoft supports Bayesian Networks
  - The UAI community does not like „outsiders“

Fuzzy Set Theory

What is Fuzzy Set ?

- The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut.
- Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.
- Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the set.

  Fuzzy set theory permits the gradual assessment of the membership of elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].
- **Example:**

  Words like young, tall, good, or high are fuzzy.

  - There is no single quantitative value which defines the term young.
  - For some people, age 25 is young, and for others, age 35 is young.
  - The concept young has no clean boundary.
  - Age 1 is definitely young and age 100 is definitely not young;
  - Age 35 has some possibility of being young and usually depends on the context in which it is being considered.

Introduction

In real world, there exists much fuzzy knowledge;
Knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature.

Human thinking and reasoning frequently involve fuzzy information, originating from inherently inexact human concepts. Humans, can give satisfactory answers, which are probably true.

However, our systems are unable to answer many questions. The reason is, most systems are designed based upon classical set theory and two-valued logic which is unable to cope with unreliable and incomplete information and give expert opinions.

- ## Classical Set Theory

  A Set is any well defined collection of objects. An object in a set is called an element or member of that set.

  - Sets are defined by a simple statement describing whether a particular element having a certain property belongs to that particular set.
  - Classical set theory enumerates all its elements using

    $$A = \{ a_1 , a_2 , a_3 , a_4 , \ldots . a_n \}$$

    If the elements a_i $(i = 1, 2, 3, \ldots n)$ of a set A are subset of universal set X, then set A can be represented for all elements $x \in X$ by its **characteristic function**

    $$\mu_A (x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases}$$

  - A set A is well described by a function called characteristic function.

    This function, defined on the universal space X, assumes :

    a value of 1 for those elements x that belong to set A, and

    a value of 0 for those elements x that do not belong to set A.

    The notations used to express these mathematically are

    $$\left.\begin{array}{l} A : X \rightarrow [0, 1] \\ A(x) = 1 , \ x \text{ is a member of } A \\ A(x) = 0 , \ x \text{ is not a member of } A \end{array}\right\} \quad \textbf{Eq.(1)}$$

    Alternatively, the set A can be represented for all elements $x \in X$ by its characteristic function $\mu_A (x)$ defined as

    $$\mu_A (x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases} \qquad \textbf{Eq.(2)}$$

  - Thus in classical set theory $\mu_A (x)$ has only the values 0 ('false') and 1 ('true''). Such sets are called **crisp sets.**

- **Fuzzy Set Theory**

  Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, *True* and *False*, is sometimes inadequate when describing human reasoning. Fuzzy logic uses the whole interval between **0** (false) and **1** (true) to describe human reasoning.

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

  - The **degree of membership** or truth is not same as probability;
    - fuzzy truth is not likelihood of some event or condition.
    - fuzzy truth represents membership in vaguely defined sets;

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

  - Fuzzy logic is capable of handling inherently imprecise concepts.

  - Fuzzy logic allows in linguistic form the set membership values to imprecise concepts like **"slightly", "quite" and "very".**

  - Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets.

- **Crisp and Non-Crisp Set**

  - As said before, in classical set theory, the **characteristic function** $\mu_A(x)$ of Eq.(2) has only values **0** ('false') and **1** ('true'').

    Such sets are **crisp sets.**

  - For Non-crisp sets the characteristic function $\mu_A(x)$ can be defined.
    - The characteristic function $\mu_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
    - This generalized characteristic function $\mu_A(x)$ of Eq.(2) is called **membership function**.

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

  - Crisp set theory is not capable of representing descriptions and classifications in many cases; In fact, Crisp set does not provide adequate representation for most cases.

- **Representation of Crisp and Non-Crisp Set**

  Example : Classify students for a basketball team

  This example explains the grade of truth value.

  - **tall students** qualify and **not tall students** do not qualify
  - if students 1.8 m tall are to be qualified, then should we exclude a student who is $^1/_{10}$" less? or should we exclude a student who is 1" shorter?

  - Non-Crisp Representation to represent the notion of a tall person.

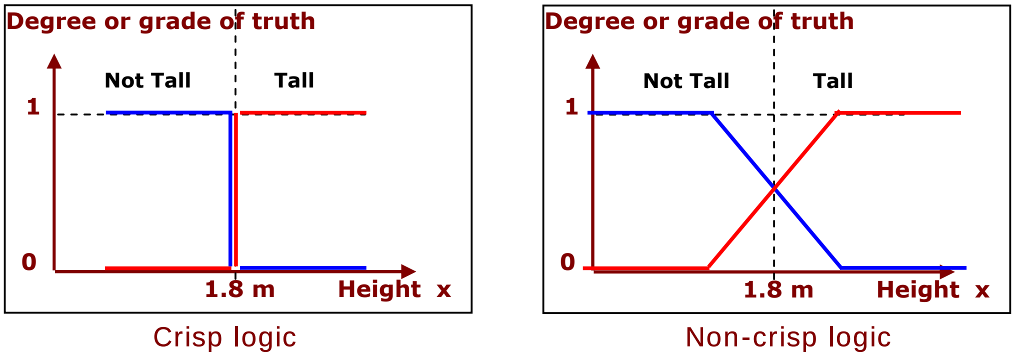


Fig. 1 Set Representation – Degree or grade of truth

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

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

- **Capturing Uncertainty**

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

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

    The number $\mu_A(\mathbf{x}_\sigma)$ represents the **degree of membership** of $\mathbf{x}_\sigma$ in **A**.
  - The notation used for membership function μ_A **(x)** of a fuzzy set **A** is

    $\mathbf{A : X \rightarrow [0, 1]}$
  - Each membership function maps elements of a given universal base set **X** , which is itself a crisp set, into real numbers in **[0, 1]** .

- Example

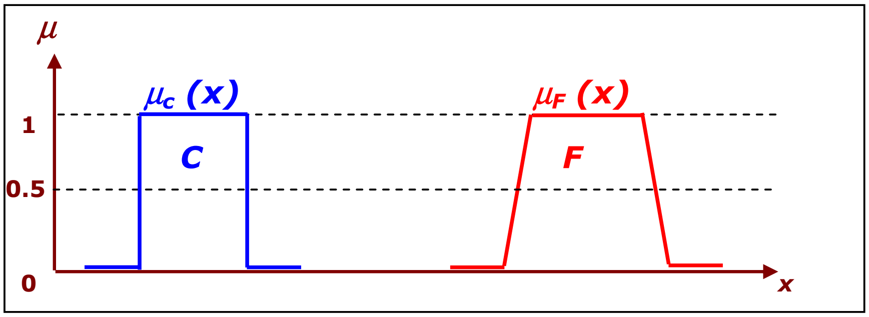


Fig. 2 Membership function of a Crisp set C and Fuzzy set F

- In the case of Crisp Sets the members of a set are :

  either out of the set, with membership of degree " **0** ",

  or in the set, with membership of degree " **1** ",

  Therefore, **Crisp Sets** ⊆ **Fuzzy Sets**

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

- **Examples of Crisp and Non-Crisp Set**

Example 1: Set of prime numbers (a crisp set)

If we consider space **X** consisting of natural numbers ≤ **12**

ie **X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}**

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

PRIME = {x contained in X | x is a prime number} = {2, 3, 5, 6, 7, 11}

Example 2: Set of SMALL (as non-crisp set)

A Set **X** that consists of SMALL cannot be described;

for example **1** is a member of SMALL and **12** is not a member of SMALL.

Set **A**, as SMALL, has un-sharp boundaries, can be characterized by a function that assigns a real number from the closed interval from **0** to **1** to each element **x** in the set **X**.

Fuzzy Set

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

- ### Definition of Fuzzy set

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

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

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

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

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

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

  Example : Set **SMALL** in set **X** consisting of natural numbers **≤ to 12**.

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

  Then, following the notations described in the definition above :

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

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

- ### Definition of Universal Space

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

  The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements. This gives the user more flexibility in choosing the universal space.

  Example : The fuzzy set of numbers, defined in the universal space

  X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

  SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

Fuzzy Membership

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

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

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

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

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

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

The Graphic Interpretation of fuzzy membership for the fuzzy sets : Small, Prime Numbers, Universal-space, Finite and Infinite UniversalSpace, and Empty are illustrated in the next few slides.

- ### Graphic Interpretation of Fuzzy Sets SMALL

The fuzzy set SMALL of small numbers, defined in the universal space **$X = \{ x_i \} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$** is presented as
SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **SMALL** in set **X** is :

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

Therefore **SetSmall** is represented as

**SetSmall = FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2},
{8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

FuzzyPlot [SMALL, AxesLable → {"X", "SMALL"}]

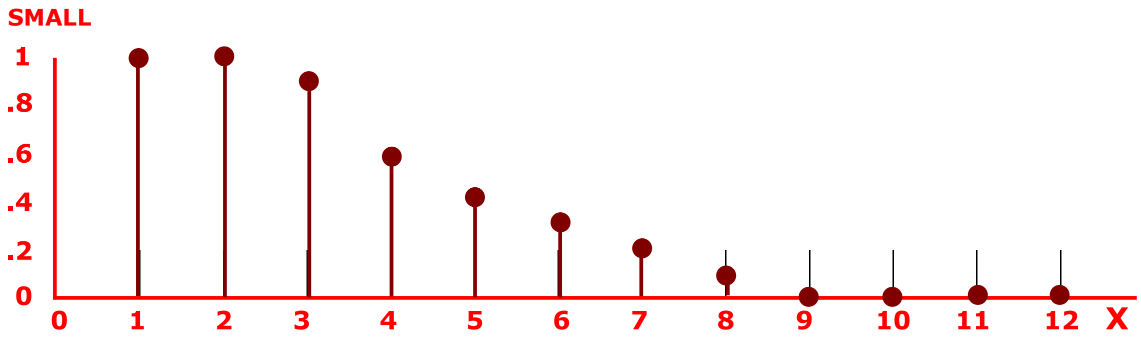


Fig Graphic Interpretation of Fuzzy Sets SMALL

Graphic Interpretation of Fuzzy Sets PRIME Numbers

The fuzzy set PRIME numbers, defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **PRIME** in set **X** is :

**PRIME = FuzzySet {{1, 0}, {2, 1}, {3, 1}, {4, 0}, {5, 1}, {6, 0}, {7, 1}, {8, 0},
{9, 0}, {10, 0}, {11, 1}, {12, 0}}**

Therefore **SetPrime** is represented as

SetPrime = **FuzzySet [{{1,0},{2,1}, {3,1}, {4,0}, {5,1},{6,0}, {7,1},
{8, 0}, {9, 0}, {10, 0}, {11, 1}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**

FuzzyPlot [PRIME, AxesLable → {"X", "PRIME"}]

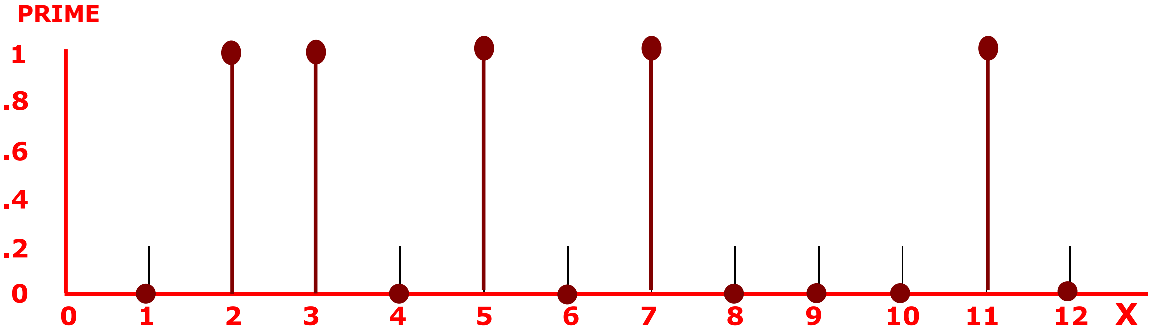


Fig Graphic Interpretation of Fuzzy Sets PRIME

Graphic Interpretation of Fuzzy Sets UNIVERSALSPACE

In any application of sets or fuzzy sets theory, all sets are subsets of a fixed set called universal space or universe of discourse denoted by **X**. Universal space **X** as a fuzzy set is a function equal to **1** for all elements.

The fuzzy set **UNIVERSALSPACE** numbers, defined in the universal space **X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **UNIVERSALSPACE** in set **X** is :

**UNIVERSALSPACE = FuzzySet {{1, 1}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1},
{7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}**

Therefore **SetUniversal** is represented as

SetUniversal = FuzzySet [{{1,1},{2,1}, {3,1}, {4,1}, {5,1},{6,1}, {7,1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}} , UniversalSpace → {1, 12, 1}]

FuzzyPlot [UNIVERSALSPACE, AxesLable → {"X", " UNIVERSAL SPACE "}]

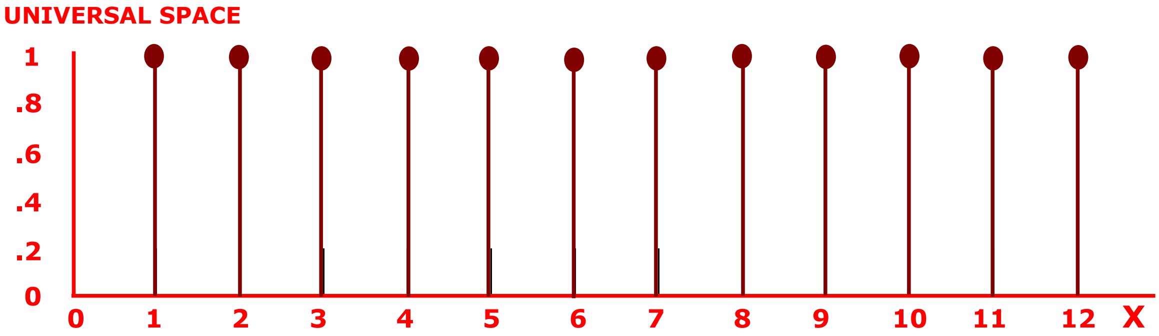


Fig Graphic Interpretation of Fuzzy Set UNIVERSALSPACE

- **Finite and Infinite Universal Space**

Universal sets can be finite or infinite.

Any universal set is finite if it consists of a specific number of different elements, that is, if in counting the different elements of the set, the counting can come to an end, else the set is infinite.

Examples:

1. Let **N** be the universal space of the days of the week.

   N = {Mo, Tu, We, Th, Fr, Sa, Su}. **N** is finite.

2. Let **M = {1, 3, 5, 7, 9, ...}.** **M** is infinite.

3. Let **L = {u | u is a lake in a city }.** **L** is finite.

   (Although it may be difficult to count the number of lakes in a city, but **L** is still a finite universal set.)

- ## Graphic Interpretation of Fuzzy Sets EMPTY

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

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

The Empty set is also called the Null set.

The fuzzy set **EMPTY** , defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **EMPTY** in set **X** is :

**EMPTY = FuzzySet {{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0},
{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}**

Therefore **SetEmpty** is represented as

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

FuzzyPlot [EMPTY, AxesLable → {"X", " UNIVERSAL SPACE "}]

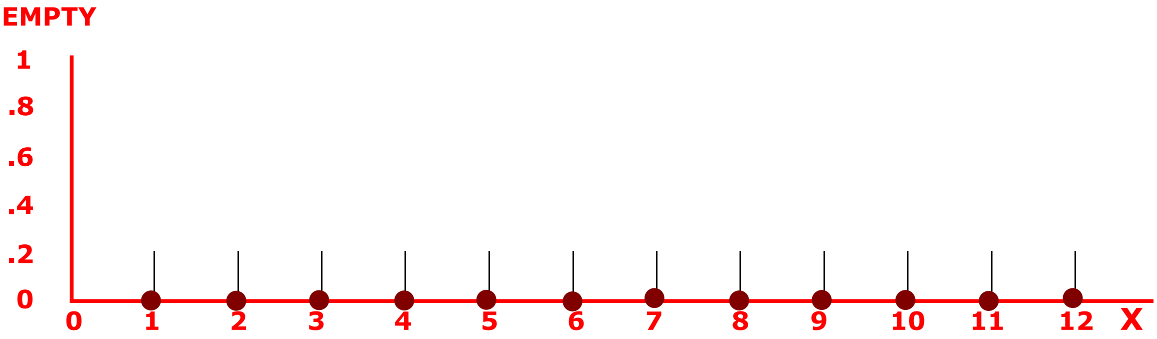


Fig Graphic Interpretation of Fuzzy Set EMPTY

Fuzzy Operations

A fuzzy set operations are the operations on fuzzy sets. The fuzzy set operations are generalization of crisp set operations. Zadeh [1965] formulated the fuzzy set theory in the terms of standard operations: Complement, Union, Intersection, and Difference.

In this section, the graphical interpretation of the following standard fuzzy set terms and the Fuzzy Logic operations are illustrated:

Inclusion :	**FuzzyInclude [VERYSMALL, SMALL]**
Equality :	**FuzzyEQUALITY [SMALL, STILLSMALL]**
Complement :	**FuzzyNOTSMALL = FuzzyCompliment [Small]**
Union :	**FuzzyUNION = [SMALL ∪ MEDIUM]**
Intersection :	**FUZZYINTERSECTON = [SMALL ∩ MEDIUM]**

- ### Inclusion

Let **A** and **B** be fuzzy sets defined in the same universal space **X**.
The fuzzy set **A** is included in the fuzzy set **B** if and only if for every **x** in the set **X** we have **A(x)** ≤ **B(x)**

Example :

The fuzzy set **UNIVERSALSPACE** numbers, defined in the universal space $\mathbf{X = \{ x_i \} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}}$ is presented as
SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The fuzzy set B SMALL

The Set **SMALL** in set **X** is :

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

Therefore **SetSmall** is represented as

**SetSmall = FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2},
{8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

The fuzzy set A VERYSMALL

The Set **VERYSMALL** in set **X** is :

**VERYSMALL = FuzzySet {{1, 1 }, {2, 0.8 }, {3, 0.7}, {4, 0.4}, {5, 0.2},
{6, 0.1}, {7, 0 }, {8, 0 }, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}**

Therefore **SetVerySmall** is represented as

SetVerySmall = **FuzzySet [{{1,1},{2,0.8}, {3,0.7}, {4,0.4}, {5,0.2},{6,0.1}, {7,0}, {8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**

The Fuzzy Operation : Inclusion

Include [VERYSMALL, SMALL]

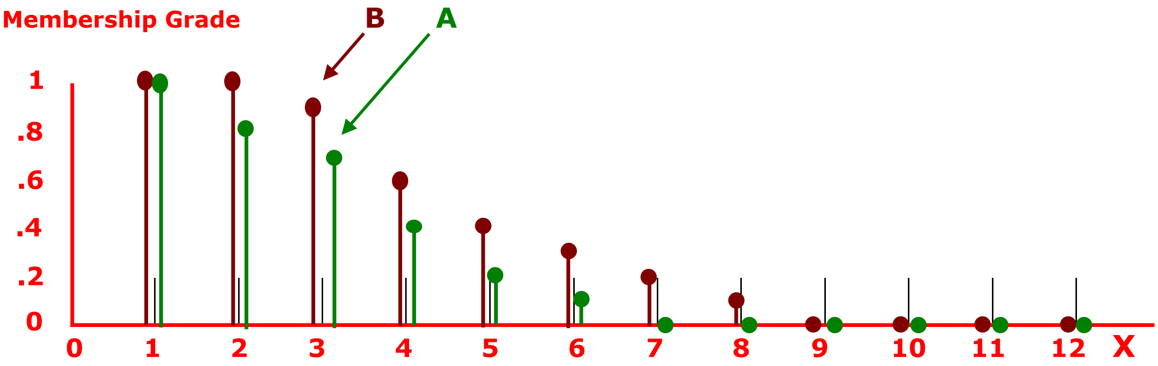


Fig Graphic Interpretation of Fuzzy Inclusion
FuzzyPlot [SMALL, VERYSMALL]

- **Comparability**

  Two fuzzy sets **A** and **B** are comparable

  if the condition $\mathbf{A} \subset \mathbf{B}$ **or** $\mathbf{B} \subset \mathbf{A}$ holds, ie,

  if one of the fuzzy sets is a subset of the other set, they are comparable.

  Two fuzzy sets **A** and **B** are incomparable

  If the condition $\mathbf{A} \not\subset \mathbf{B}$ **or** $\mathbf{B} \not\subset \mathbf{A}$ holds.

  Example 1:

  Let **A = {{a, 1}, {b, 1}, {c, 0}} and**
  B = {{a, 1}, {b, 1}, {c, 1}}.

  Then **A** is comparable to **B**, since **A** is a subset of **B**.

  Example 2 :

  Let **C = {{a, 1}, {b, 1}, {c, 0.5}} and**
  D = {{a, 1}, {b, 0.9}, {c, 0.6}}.

  Then **C** and **D** are not comparable since

  C is not a subset of **D** and

  D is not a subset of **C**.

Property Related to Inclusion :

for all **x** in the set **X**, if **A(x)** ⊂ **B(x)** ⊂

- **Equality**

Let **A** and **B** be fuzzy sets defined in the same space **X**.

Then **A** and **B** are equal, which is denoted **X = Y**

if and only if for all **x** in the set **X**, **A(x) = B(x).**

Example.

The fuzzy set B SMALL

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

The fuzzy set A STILLSMALL

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

The Fuzzy Operation : Equality

Equality [SMALL, STILLSMALL]

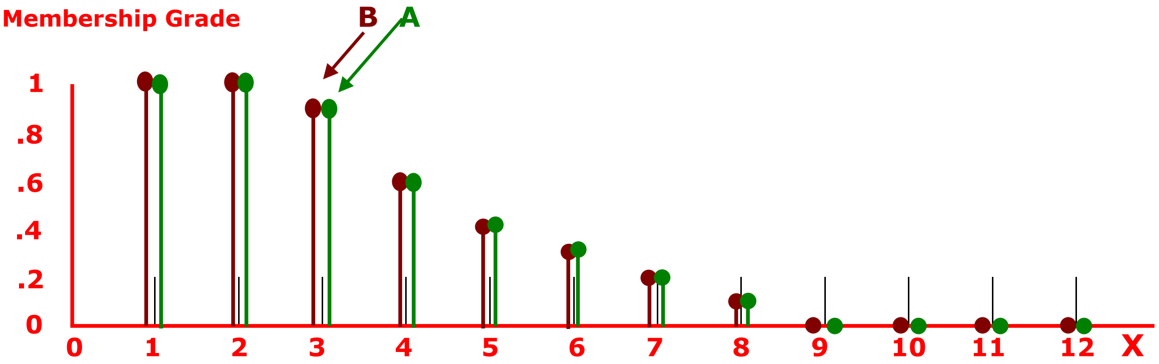


Fig Graphic Interpretation of Fuzzy Equality
FuzzyPlot [SMALL, STILLSMALL]

Note : If equality **A(x) = B(x)** is not satisfied even for one element **x** in the set **X**, then we say that **A** is not equal to **B**.

- **Complement**

  Let **A** be a fuzzy set defined in the space **X**.

  Then the fuzzy set **B** is a complement of the fuzzy set **A**, if and only if, for all **x** in the set **X**, **B(x) = 1 - A(x).**

  The complement of the fuzzy set **A** is often denoted by **A'** or **Ac** or $\bar{A}$

  Fuzzy Complement : **Ac(x) = 1 – A(x)**

  Example 1.

  The fuzzy set A SMALL

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

  The fuzzy set Ac NOTSMALL

  NOTSMALL = FuzzySet {{1, 0 }, {2, 0 }, {3, 0.1}, {4, 0.4}, {5, 0.6}, {6, 0.7}, {7, 0.8}, {8, 0.9}, {9, 1 }, {10, 1 }, {11, 1}, {12, 1}}

  The Fuzzy Operation : Compliment

  NOTSMALL = Compliment [SMALL]

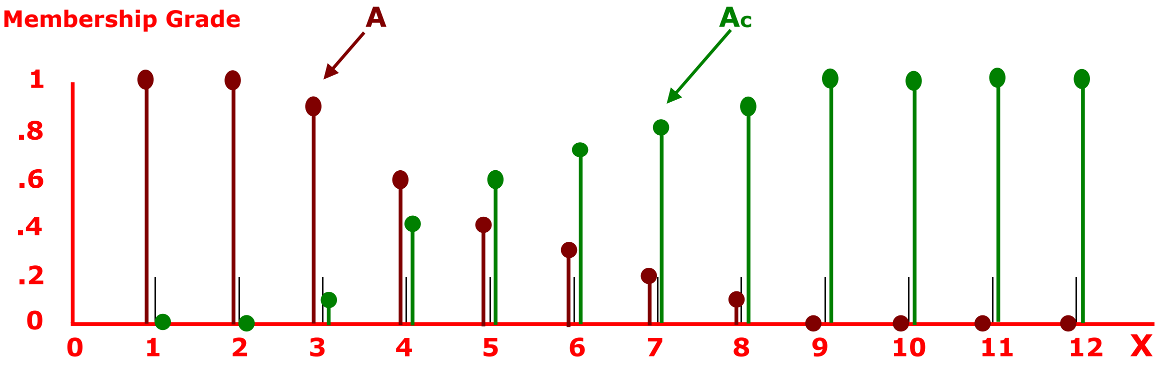


Fig Graphic Interpretation of Fuzzy Compliment
FuzzyPlot [SMALL, NOTSMALL]

Example 2.

The empty set Φ and the universal set **X**, as fuzzy sets, are complements of one another.

Φ' = X , X' = Φ

The fuzzy set B EMPTY

**Empty = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0}, {5, 0}, {6, 0},
{7, 0}, {8, 0}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}**

The fuzzy set A UNIVERSAL

**Universal = FuzzySet {{1, 1 }, {2, 1 }, {3, 1}, {4, 1}, {5, 1}, {6, 1},
{7, 1}, {8, 1}, {9, 1 }, {10, 1 }, {11, 1}, {12, 1}}**

The fuzzy operation : Compliment

EMPTY = Compliment [UNIVERSALSPACE]

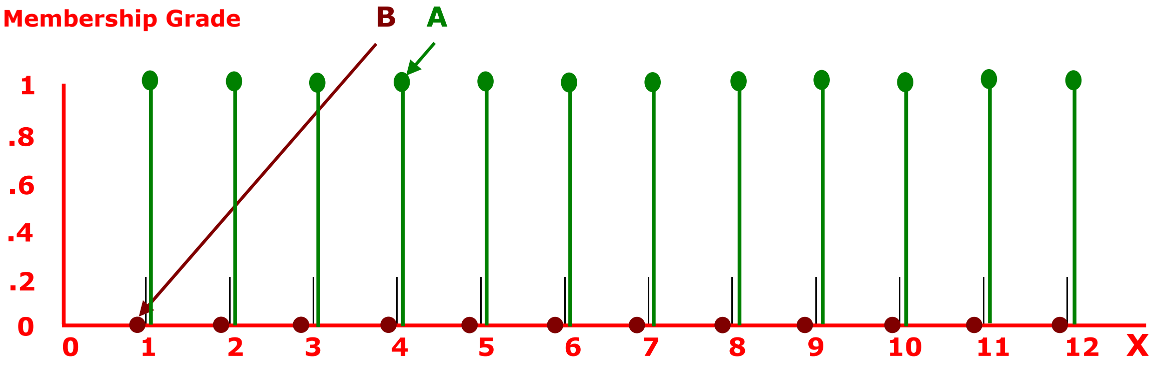


**Fig Graphic Interpretation of Fuzzy Compliment
FuzzyPlot [EMPTY, UNIVERSALSPACE]**

- **Union**

Let **A** and **B** be fuzzy sets defined in the space **X**.

The union is defined as the smallest fuzzy set that contains both **A** and **B**.

The union of **A** and **B** is denoted by **A ∪ B.**

The following relation must be satisfied for the union operation :

for all x in the set X, (A ∪ B)(x) = Max (A(x), B(x)).

Fuzzy Union : **(A ∪ B)(x) = max [A(x), B(x)] for all x ∈ X**

Example 1 : Union of Fuzzy **A** and **B**

A(x) = 0.6 and B(x) = 0.4 ∴ (A ∪ B)(x) = max [0.6, 0.4] = 0.6

Example 2 : Union of **SMALL** and **MEDIUM**

The fuzzy set A SMALL

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

The fuzzy set B MEDIUM

MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8}, {7, 1}, {8, 1}, {9, 0.7 }, {10, 0.4 }, {11, 0.1}, {12, 0}}

The fuzzy operation : Union

FUZZYUNION = [SMALL ∪ MEDIUM]

SetSmallUNIONMedium = **FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.7}, {10, 0.4}, {11, 0.1}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

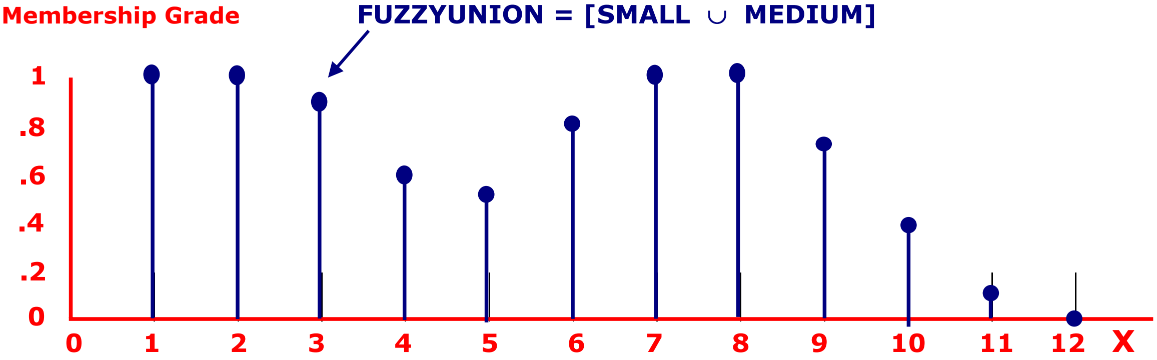


Fig Graphic Interpretation of Fuzzy Union
FuzzyPlot [UNION]

The notion of the union is closely related to that of the connective "or".

Let **A** is a class of "Young" men, **B** is a class of "Bald" men.

If "David is Young" or "David is Bald," then David is associated with the union of **A** and **B.** Implies David is a member of **A ∪ B**.

- **Intersection**

Let **A** and **B** be fuzzy sets defined in the space **X**. Intersection is defined as the greatest fuzzy set that include both **A** and **B**. Intersection of **A** and **B** is denoted by **A ∩ B.** The following relation must be satisfied for the intersection operation :

for all x in the set X, $(A \cap B)(x) = \text{Min}\ (A(x), B(x))$.

Fuzzy Intersection : $(A \cap B)(x) = \min\ [A(x), B(x)]$ **for all** $x \in X$

Example 1 : Intersection of Fuzzy **A** and **B**

$A(x) = 0.6$ **and** $B(x) = 0.4$ $\therefore$ $(A \cap B)(x) = \min\ [0.6, 0.4] = 0.4$

Example 2 : Union of **SMALL** and **MEDIUM**

The fuzzy set A SMALL

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

The fuzzy set B MEDIUM

**MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8},
{7, 1}, {8, 1}, {9, 0.7 }, {10, 0.4 }, {11, 0.1}, {12, 0}}**

The fuzzy operation : Intersection

FUZZYINTERSECTION = min [SMALL ∩ MEDIUM]

**SetSmallINTERSECTIONMedium = FuzzySet [{{1,0},{2,0}, {3,0}, {4,0.2},
{5,0.4}, {6,0.3}, {7,0.2}, {8, 0.1}, {9, 0},
{10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

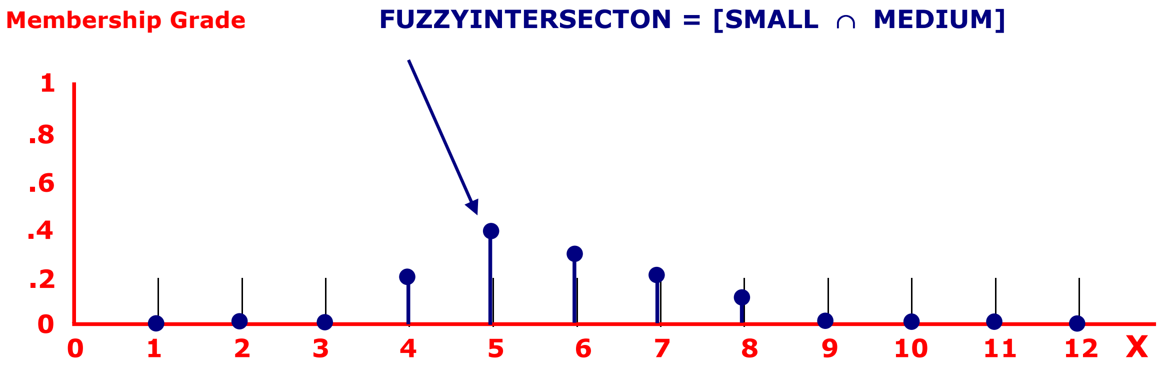


**Fig Graphic Interpretation of Fuzzy Union
FuzzyPlot [INTERSECTION]**

- **Difference**

Let **A** and **B** be fuzzy sets defined in the space **X**.

The difference of **A** and **B** is denoted by **A ∩ B'.**

Fuzzy Difference : **(A - B)(x) = min [A(x), 1- B(x)] for all x ∈ X**

Example : Difference of MEDIUM and SMALL

The fuzzy set A MEDIUM

**MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8},
{7, 1}, {8, 1}, {9, 0.7 }, {10, 0.4 }, {11, 0.1}, {12, 0}}**

The fuzzy set B SMALL

**MEDIUM = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3},
{7, 0.2}, {8, 0.1}, {9, 0.7 }, {10, 0.4 }, {11, 0}, {12, 0}}**

Fuzzy Complement : Bc(x) = 1 – B(x)

The fuzzy set Bc NOTSMALL

**NOTSMALL = FuzzySet {{1, 0 }, {2, 0 }, {3, 0.1}, {4, 0.4}, {5, 0.6}, {6, 0.7},
{7, 0.8}, {8, 0.9}, {9, 1 }, {10, 1 }, {11, 1}, {12, 1}}**

The fuzzy operation : Difference by the definition of **Difference**

FUZZYDIFFERENCE = [MEDIUM ∩ SMALL']

**SetMediumDIFFERECESmall = FuzzySet [{{1,0},{2,0}, {3,0}, {4,0.2},
{5,0.5}, {6,0.7}, {7,0.8}, {8, 0.9}, {9, 0.7},
{10, 0.4}, {11, 0.1}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

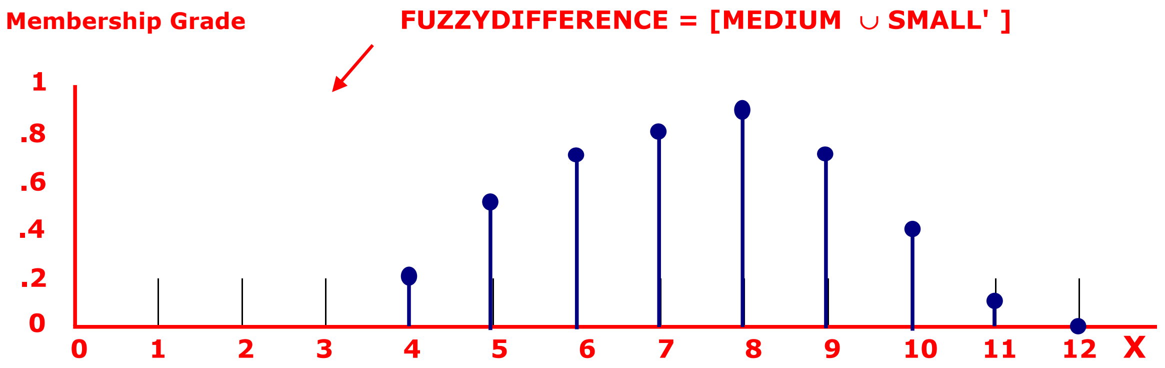


**Fig Graphic Interpretation of Fuzzy Union
FuzzyPlot [UNION]**

Fuzzy Properties

Properties related to Union, Intersection, Differences are illustrated below.

- **Properties Related to Union**

  The properties related to union are :

  Identity, Idempotence, Commutativity and Associativity.

  - **Identity:**

    $A \cup \Phi = A$

    input = Equality [SMALL ∪ EMPTY , SMALL]

    output = True

    $A \cup X = X$

    input = Equality [SMALL ∪ UnivrsalSpace , UnivrsalSpace]

    output = True

  - **Idempotence :**

    $A \cup A = A$

    input = Equality [SMALL ∪ SMALL , SMALL]

    output = True

  - **Commutativity :**

    $A \cup B = B \cup A$

    input = Equality [SMALL ∪ MEDIUM, MEDIUM ∪ SMALL]

    output = True

- **Associativity:**

A ∪ (B∪ C) = (A∪ B) ∪ C

input = Equality [Small ∪ (Medium ∪ Big) , (Small ∪ Medium**)** ∪ Big]

output = True

Fuzzy Set Small , Medium , Big

**Small = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3},
{7, 0.2}, {8, 0.1}, {9, 0.7 }, {10, 0.4 }, {11, 0}, {12, 0}}**

**Medium = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8},
{7, 1}, {8, 1}, {9, 0 }, {10, 0 }, {11, 0.1}, {12, 0}}**

**Big = FuzzySet [{{1,0}, {2,0}, {3,0}, {4,0}, {5,0}, {6,0.1},
{7,0.2}, {8,0.4}, {9,0.6}, {10,0.8}, {11,1}, {12,1}}]**

Calculate Fuzzy relations :

(1) Medium ∪ Big = **FuzzySet [{1,0},{2,0}, {3,0}, {4,0.2}, {5,0.5},
{6,0.8},{7,1}, {8, 1}, {9, 0.6}, {10, 0.8}, {11, 1}, {12, 1}]**

(2) Small ∪ Medium = **FuzzySet [{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.5},
{6,0.8}, {7,1}, {8, 1}, {9, 0.7}, {10, 0.4}, {11, 0.1}, {12, 0}]**

(3) Small ∪ (Medium ∪ Big) = **FuzzySet [{1,1},{2,1}, {3,0.9}, {4,0.6},
{5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.7}, {10, 0.8}, {11, 1}, {12, 1}]**

**(4) (Small ∪ Medium) ∪ Big] = FuzzySet [{1,1},{2,1}, {3,0.9}, {4,0.6},
{5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.7},{10, 0.8}, {11, 1},{12, 1}]**

Fuzzy set (3) and (4) proves Associativity relation

- **Properties Related to Intersection**

  Absorption, Identity, Idempotence, Commutativity, Associativity.

  - **Absorption by Empty Set :**

    $\mathbf{A \cap \Phi = \Phi}$

    input = Equality [Small ∩ Empty , Empty]

    output = True

  - **Identity :**

    $\mathbf{A \cap X = A}$

    input = Equality [Small ∩ UnivrsalSpace , Small]

    output = True

  - **Idempotence :**

    $\mathbf{A \cap A = A}$

    input = Equality [Small ∩ Small , Small]

    output = True

  - **Commutativity :**

    $\mathbf{A \cap B = B \cap A}$

    input = Equality [Small ∩ Big , Big ∩ Small]

    output = True

  - **Associativity :**

    $\mathbf{A \cap (B \cap C) = (A \cap B) \cap C}$

    input = Equality [Small ∩ (Medium ∩ Big), (Small ∩ Medium) ∩ Big]

    output = True

- **Additional Properties**

  Related to Intersection and Union

  - **Distributivity:**

    $\mathbf{A \cap (B \cup C) = (A \cap B) \cup (A \cap C)}$

    input = Equality [Small ∩ (Medium ∪ Big) ,
    (Small ∩ Medium) ∪ (Small ∩ Big)]

    output = True

  - **Distributivity:**

    $\mathbf{A \cup (B \cap C) = (A \cup B) \cap (A \cup C)}$

    input = Equality [Small ∪ (Medium ∩ Big) ,
    (Small ∪ Medium) ∩ (Small ∪ Big)]

    output = True

- **Law of excluded middle :**

  $A \cup A' = X$

  input = Equality [Small ∪ NotSmall , UnivrsalSpace]

  output = True

- **Law of contradiction**

  $A \cap A' = \Phi$

  input = Equality [Small ∩ NotSmall , EmptySpace]

  output = True

● Cartesian Product Of Two Fuzzy Sets

Cartesian Product of two Crisp Sets

Let **A** and **B** be two crisp sets in the universe of discourse **X** and **Y**..

The Cartesian product of **A** and **B** is denoted by **A x B**

Defined as $A \times B = \{ (a , b) \mid a \in A , b \in B \}$

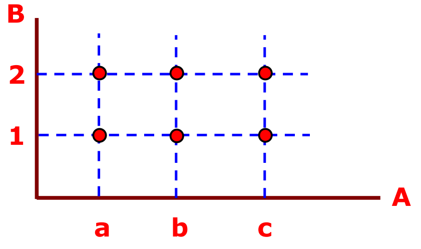
Note : Generally $A \times B \neq B \times A$

Example :

Let $A = \{a, b, c\}$ and $B = \{1, 2\}$

then $A \times B = \{ (a , 1) , (a , 2) , (b , 1) , (b , 2) , (c , 1) , (c , 2) \}$

Graphic representation of A x B



Cartesian product of two Fuzzy Sets

Let **A** and **B** be two fuzzy sets in the universe of discourse **X** and **Y**.

The Cartesian product of **A** and **B** is denoted by **A x B**

Defined by their membership function $\mu_A(x)$ and $\mu_B(y)$ as

$$\mu_{A \times B}(x , y) = \min [\mu_A(x) , \mu_B(y)] = \mu_A(x) \wedge \mu_B(y)$$

or $$\mu_{A \times B}(x , y) = \mu_A(x) \, \mu_B(y)$$

for all $x \in X$ and $y \in Y$

Thus the Cartesian product **A x B** is a fuzzy set of ordered pair (x , y) for all $x \in X$ and $y \in Y$, with grade membership of (x , y) in $X \times Y$ given by the above equations .

In a sense Cartesian product of two Fuzzy sets is a Fuzzy Relation.

Fuzzy Relations

Fuzzy Relations describe the degree of association of the elements;
Example : **"x is approximately equal to y".**

- Fuzzy relations offer the capability to capture the uncertainty and vagueness in relations between sets and elements of a set.
- Fuzzy Relations make the description of a concept possible.
- Fuzzy Relations were introduced to supersede classical crisp relations; It describes the total presence or absence of association of elements.

In this section, first the fuzzy relation is defined and then expressing fuzzy relations in terms of matrices and graphical visualizations. Later the properties of fuzzy relations and operations that can be performed with fuzzy relations are illustrated.

3.1 Definition of Fuzzy Relation

Fuzzy relation is a generalization of the definition of fuzzy set from 2-D space to 3-D space.

- **Fuzzy relation definition**

Consider a Cartesian product

$$\mathbf{A \times B = \{ (x , y) \mid x \in A, y \in B \}}$$

where **A** and **B** are subsets of universal sets $\mathbf{U_1}$ and $\mathbf{U_2}$.

Fuzzy relation on **A x B** is denoted by **R** or **R(x , y)** is defined as the set

$$\mathbf{R = \{ ((x , y) , \mu_R (x , y)) \mid (x , y) \in A \times B , \mu_R (x , y) \in [0,1] \}}$$

where μ_R **(x , y)** is a function in two variables called membership function.

- It gives the degree of membership of the ordered pair **(x , y)** in **R** associating with each pair **(x , y)** in **A x B** a real number in the interval **[0 , 1]**.
- The degree of membership indicates the degree to which **x** is in relation to **y**.

- **Example of Fuzzy Relation**

$$R = \{ ((x_1 , y_1) , 0)) , \quad ((x_1 , y_2) , 0.1)) , \quad ((x_1 , y_3) , 0.2)) ,$$
$$((x_2 , y_1) , 0.7)) , ((x_2 , y_2) , 0.2)) , \quad ((x_2 , y_3) , 0.3)) ,$$
$$((x_3 , y_1) , 1)) , \quad ((x_3 , y_2) , 0.6)) , \quad ((x_3 , y_3) , 0.2)) ,$$

The relation can be written in matrix form as

$$R \triangleq \begin{array}{c|ccc} {}_{x}\ {}^{y} & y_1 & Y_2 & Y_3 \\ \hline x_1 & 0 & 0.1 & 0.2 \\ X_2 & 0.7 & 0.2 & 0.3 \\ X_3 & 1 & 0.6 & 0.2 \end{array}$$

where symbol $\triangleq$ means ' is defined as' and

the values in the matrix are the values of membership function:

$$\mu_R (x_1 , y_1) = 0 \qquad \mu_R (x_1 , y_2) = 0.1 \qquad \mu_R (x_1 , y_3) = 0.2$$
$$\mu_R (x_2 , y_1) = 0.7 \qquad \mu_R (x_2, y_2) = 0.2 \qquad \mu_R (x_2 , y_3) = 0.3$$
$$\mu_R (x_3 , y_1) = 1 \qquad \mu_R (x_3 , y_2) = 0.6 \qquad \mu_R (x_3 , y_3) = 0.2$$

Assuming $x_1 = 1$, $x_2 = 2$, $x_3 = 3$ and $y_1 = 1$, $y_2 = 2$, $y_3 = 3$, the relation can be graphically represented by points in 3-D space (X, Y, μ) as :

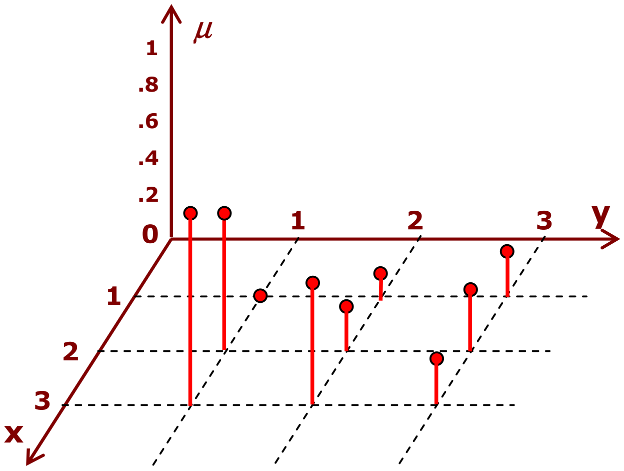


Fig Fuzzy Relation R describing x greater than y

Note : Since the values of the membership function **0.7, 1, 0.6** are in the direction of **x** below the major diagonal **(0, 0.2, 0.2)** in the matrix are grater than those **0.1, 0.2, 0.3** in the direction of **y**, we therefore say that the relation **R** describes **x** is grater than **y**.

Forming Fuzzy Relations

Assume that **V** and **W** are two collections of objects.

A fuzzy relation is characterized in the same way as it is in a fuzzy set.

- The first item is a list containing element and membership grade pairs,

  $\{\{v_1, w_1\}, R_{11}\}, \{\{v_1, w_2\}, R_{12}\}, \dots, \{\{v_n, w_m\}, R_{nm}\}\}.$

  where $\{v_1, w_1\}, \{v_1, w_2\}, \dots, \{v_n, w_m\}$ are the elements of the relation are defined as ordered pairs, and $\{R_{11}, R_{12}, \dots, R_{nm}\}$ are the membership grades of the elements of the relation that range from 0 to 1, inclusive.

- The second item is the universal space; for relations, the universal space consists of a pair of ordered pairs,

  $\{\{V_{min}, V_{max}, C_1\}, \{W_{min}, W_{max}, C_2\}\}.$

  where the first pair defines the universal space for the first set and the second pair defines the universal space for the second set.

Example showing how fuzzy relations are represented

Let V = {1, 2, 3} and W = {1, 2, 3, 4}.

A fuzzy relation **R** is, a function defined in the space **V x W**, which takes values from the interval [0, 1] , expressed as $R : V \times W \rightarrow [0, 1]$

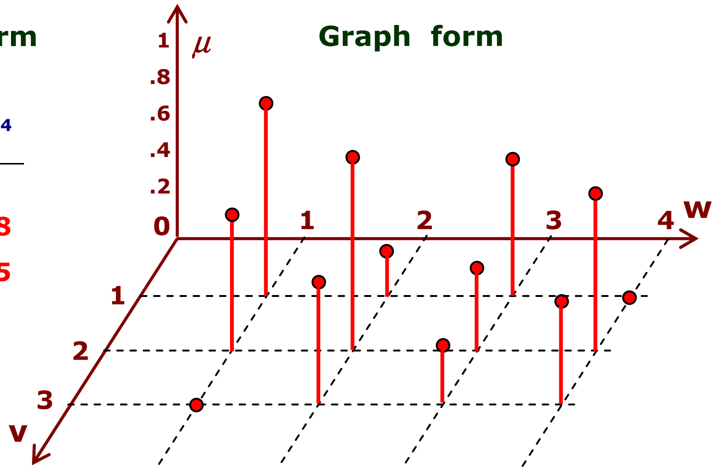
```
R = FuzzyRelation [{{{1, 1}, 1},   {{1, 2}, 0.2}, {{1, 3}, 0.7}, {{1, 4},  0},
                    {{2, 1}, 0.7}, {{2, 2}, 1},   {{2, 3}, 0.4}, {{2, 4},  0.8},
                    {{3, 1}, 0},   {{3, 2}, 0.6}, {{3, 3}, 0.3}, {{3, 4},  0.5},
                    UniversalSpace → {{1, 3, 1}, {1, 4, 1}}]
```

This relation can be represented in the following two forms shown below

Membership matrix form

$$R \triangleq \begin{array}{c|cccc} {}_{v}{}^{w} & w_1 & w_2 & w_3 & w_4 \\ \hline v_1 & 1 & 0.2 & 0.7 & 0 \\ v_2 & 0.7 & 1 & 0.4 & 0.8 \\ v_3 & 0 & 0.6 & 0.3 & 0.5 \end{array}$$

Graph form



Vertical lines represent membership grades

Elements of fuzzy relation are ordered pairs $\{v_i, w_j\}$, where v_i is first and w_j is second element. The membership grades of the elements are represented by the heights of the vertical lines.

Projections of Fuzzy Relations

Definition : A fuzzy relation on **A x B** is denoted by **R** or **R(x , y)** is defined as the set

$$R = \{ ((x , y) , \mu_R (x , y)) \mid (x , y) \in A \times B , \mu_R (x , y) \in [0,1] \}$$

where $\mu_R (x , y)$ is a function in two variables called membership function. The first, the second and the total projections of fuzzy relations are stated below.

- **First Projection of R :** defined as

$$R^{(1)} = \{(x) , \mu_R{}^{(1)} (x , y))\}$$

$$= \{(x) , \max_Y \mu_R (x , y)) \mid (x , y) \in A \times B \}$$

- **Second Projection of R :** defined as

$$R^{(2)} = \{(y) , \mu_R{}^{(2)} (x , y))\}$$

$$= \{(y) , \max_X \mu_R (x , y)) \mid (x , y) \in A \times B \}$$

- **Total Projection of R :** defined as

$$R^{(T)} = \max_X \max_Y \{\mu_R (x , y) \mid (x , y) \in A \times B \}$$

Note : In all these three expression

$\max_Y$ means **max** with respect to **y** while **x** is considered fixed

$\max_X$ means **max** with respect to **x** while **y** is considered fixed

The Total Projection is also known as Global projection

- **Example : Fuzzy Projections**

The Fuzzy Relation **R** together with First, Second and Total Projection of **R** are shown below.

$R \triangleq$

y / x	y_1	y_2	y_3	y_4	Y_5	$R^{(1)}$		
x_1	0.1	0.3	1	0.5	0.3	1		
x_2	0.2	0.5	0.7	0.9	0.6	0.9		
x_3	0.3	0.6	1	0.8	0.2	1		
$R^{(2)}$	0.3	0.6	1	0.9	0.6	1	=	$R^{(T)}$

Note :

For $R^{(1)}$ select $\max_{Y}$ means **max** with respect to **y** while **x** is considered fixed

For $R^{(2)}$ select $\max_{x}$ means **max** with respect to **x** while **y** is considered fixed

For $R^{(T)}$ select **max** with respect to $R^{(1)}$ and $R^{(2)}$

The Fuzzy plot of these projections are shown below.

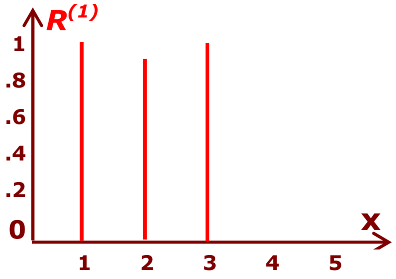


Fig Fuzzy plot of 1st projection $R^{(1)}$

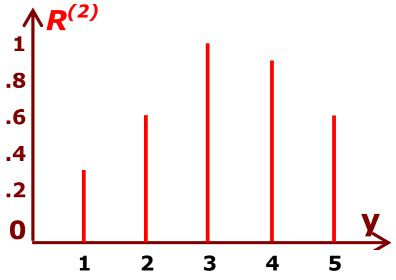


Fig Fuzzy plot of 2nd projection $R^{(2)}$

Max-Min and Min-Max Composition

The operation composition combines the fuzzy relations in different variables, say **(x , y)** and **(y , z)** ; $x \in A$, $y \in B$, $z \in C$.

Consider the relations :

$$R_1(x , y) = \{ ((x , y) , \mu_{R1} (x , y)) \mid (x , y) \in A \times B \}$$

$$R_2(y , z) = \{ ((y , y) , \mu_{R1} (y , z)) \mid (y , z) \in B \times C \}$$

The domain of R_1 is **A x B** and the domain of R_2 is **B x C**

- **Max-Min Composition**

Definition : The Max-Min composition denoted by $R_1 \circ R_2$ with membership function $\mu_{R1 \circ R2}$ defined as

$$R_1 \circ R_2 = \{ ((x , z) , \max_{Y} (\min (\mu_{R1} (x , y) , \mu_{R2} (y , z))))\} ,$$

$$(x , z) \in A \times C , \ y \in B$$

Thus $R_1 \circ R_2$ is relation in the domain **A x C**

An example of the composition is shown in the next slide.

- **Example : Max-Min Composition**

Consider the relations **$R_1(x , y)$** and **$R_2(y , z)$** as given below.

$R_1 \triangleq$

y x	y_1	y_2	y_3
x_1	0.1	0.3	0
x_2	0.8	1	0.3

$R_2 \triangleq$

z y	z_1	z_2	z_3
y_1	0.8	0.2	0
y_2	0.2	1	0.6
y_3	0.5	0	0.4

Note : Number of columns in the first table and second table are equal.

Compute max-min composition denoted by **$R_1 \circ R_2$** :

Step -1 Compute **min** operation (definition in previous slide).

Consider row **x_1** and column **z_1** , means the pair **(x_1 , z_1)** for all **y_j ,** **$j = 1, 2, 3$,** and perform **min** operation

$$\min (\mu_{R1} (x_1 , y_1) , \mu_{R2} (y_1 , z_1)) = \min (0.1, 0.8) = 0.1,$$

$$\min (\mu_{R1} (x_1 , y_2) , \mu_{R2} (y_2 , z_1)) = \min (0.3, 0.2) = 0.2,$$

$$\min (\mu_{R1} (x_1 , y_3) , \mu_{R2} (y_3 , z_1)) = \min (0, 0.5) = 0,$$

Step -2 Compute **max** operation (definition in previous slide).

For **$x = x_1$** , **$z = z_1$** , **$y = y_j$, $j = 1, 2, 3$,**

Calculate the grade membership of the pair **(x_1 , z_1)** as

$$\{ (x_1 , z_1) , \max ((\min (0.1, 0.8), \min (0.3, 0.2), \min (0, 0.5))$$

i.e. $\{ (x_1 , z_1) , \max(0.1, 0.2, 0) \}$

i.e. $\{ (x_1 , z_1) , 0.2 \}$

Hence the grade membership of the pair **(x_1 , z_1)** is **0.2** .

Similarly, find all the grade membership of the pairs

(x_1 , z_2) , (x_1 , z_3) , (x_2 , z_1) , (x_2 , z_2) , (x_2 , z_3)

The final result is

$R_1 \circ R_2 =$

z x	z_1	z_2	z_3
x_1	0.1	0.3	0
x_2	0.8	1	0.3

Note : If tables **R_1** and **R_2** are considered as matrices, the operation composition resembles the operation multiplication in matrix calculus linking row by columns. After each cell is occupied max-min value (the product is replaced by min, the sum is replaced by max).

- **Example : Min-Max Composition**

The min-max composition is similar to max-min composition with the difference that the roll of max and min are interchanged.

Definition : The max-min composition denoted by $R_1 \square R_2$ with membership function $\mu_{R1 \square R2}$ is defined by

$$R_1 \square R_2 = \{ ((x , z) , \min_{y} (\max (\mu_{R1} (x , y) , \mu_{R2} (y , z))))\} ,$$
$$(x , z) \in A \times C , \; y \in B$$

Thus $R_1 \square R_2$ is relation in the domain $A \times C$

Consider the relations $R_1(x , y)$ and $R_2(y , z)$ as given by the same relation of previous example of max-min composition, that is

$$R_1 \triangleq \begin{array}{c|ccc} {}^{x}\,{}_{}\, y & y_1 & y_2 & y_3 \\ \hline x_1 & 0.1 & 0.3 & 0 \\ x_2 & 0.8 & 1 & 0.3 \end{array} \qquad R_2 \triangleq \begin{array}{c|ccc} {}^{y}\, z & z_1 & z_2 & z_3 \\ \hline y_1 & 0.8 & 0.2 & 0 \\ y_2 & 0.2 & 1 & 0.6 \\ y_3 & 0.5 & 0 & 0.4 \end{array}$$

After computation in similar way as done in the case of max-min composition, the final result is

$$R_1 \square R_2 = \begin{array}{c|ccc} {}^{x}\, z & z_1 & z_2 & z_3 \\ \hline x_1 & 0.3 & 0 & 0.1 \\ x_2 & 0.5 & 0.4 & 0.4 \end{array}$$

- **Relation between Max-Min and Min-Max Compositions**

The Max-Min and Min-Max Compositions are related by the formula

$$\overline{R_1} \circ \overline{R_2} = \overline{R_1 \square R_2}$$

Fuzzy Systems

What are Fuzzy Systems ?

- Fuzzy Systems include Fuzzy Logic and Fuzzy Set Theory.
- Knowledge exists in two distinct forms :
  - the Objective knowledge that exists in mathematical form is used in engineering problems; and
  - the Subjective knowledge that exists in linguistic form, usually impossible to quantify.

  Fuzzy Logic can coordinate these two forms of knowledge in a logical way.
- Fuzzy Systems can handle simultaneously the numerical data and linguistic knowledge.
- Fuzzy Systems provide opportunities for modeling of conditions which are inherently imprecisely defined.
- Many real world problems have been modeled, simulated, and replicated with the help of fuzzy systems.
- The applications of Fuzzy Systems are many like : Information retrieval systems, Navigation system, and Robot vision.
- Expert Systems design have become easy because their domains are inherently fuzzy and can now be handled better;

  examples : Decision-support systems, Financial planners, Diagnostic system, and Meteorological system.

Introduction

Any system that uses Fuzzy mathematics may be viewed as Fuzzy system.

The Fuzzy Set Theory - membership function, operations, properties and the relations have been described in previous lectures. These are the prerequisites for understanding Fuzzy Systems. The applications of Fuzzy set theory is Fuzzy logic which is covered in this section.

Here the emphasis is on the design of fuzzy system and fuzzy controller in a closed–loop. The specific topics of interest are :

- Fuzzification of input information,
- Fuzzy Inferencing using Fuzzy sets ,
- De-Fuzzification of results from the Reasoning process, and
- Fuzzy controller in a closed–loop.

Fuzzy Inferencing, is the core constituent of a fuzzy system. A block schematic of Fuzzy System is shown in the next slide. Fuzzy Inferencing combines the facts obtained from the Fuzzification with the fuzzy rule base and conducts the Fuzzy Reasoning Process.

- **Fuzzy System**

A block schematic of Fuzzy System is shown below.

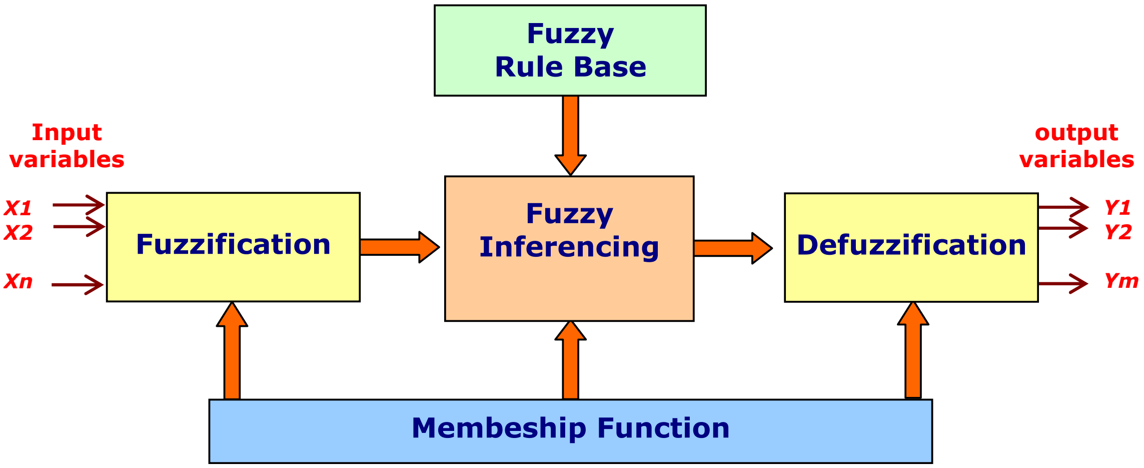


Fig. Elements of Fuzzy System

Fuzzy System elements

- **Input Vector** : $\mathbf{X = [x_1 , x_{2,} . . . x_n]^T}$ are crisp values**,** which are transformed into fuzzy sets in the fuzzification block.
- **Output Vector** : $\mathbf{Y = [y_1 , y_{2,} . . . y_m]^T}$ comes out from the defuzzification block, which transforms an output fuzzy set back to a crisp value**.**
- **Fuzzification** : a process of transforming crisp values into grades of membership for linguistic terms, "far", "near", "small" of fuzzy sets.
- **Fuzzy Rule base** : a collection of propositions containing linguistic variables; the rules are expressed in the form:

  If (x is A) AND (y is B) . . . . . . THEN (z is C)

  where **x, y** and **z** represent variables (e.g. distance, size) and **A, B** and **Z** are linguistic variables (e.g. `far', `near', `small').
- **Membership function** : provides a measure of the degree of similarity of elements in the universe of discourse **U** to fuzzy set.
- **Fuzzy Inferencing** : combines the facts obtained from the Fuzzification with the rule base and conducts the Fuzzy reasoning process.
- **Defuzzyfication:** Translate results back to the real world values.

Fuzzy Logic

A simple form of logic, called a two-valued logic is the study of "truth tables" and logic circuits. Here the possible values are true as **1**, and false as **0**.

This simple two-valued logic is generalized and called **fuzzy logic** which treats "truth" as a continuous quantity ranging from **0** to **1**.

Definition : Fuzzy logic (FL) is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical two-valued logic.

- FL is the application of Fuzzy set theory.
- FL allows set membership values to range (inclusively) between **0** and **1**.
- FL is capable of handling inherently imprecise concepts.
- FL allows in linguistic form, the set membership values to imprecise concepts like "**slightly**", "**quite**" and "**very**".

Classical Logic

Logic is used to represent simple facts. Logic defines the ways of putting symbols together to form **sentences** that represent facts. Sentences are either true or false but not both are called **propositions**.

Examples :

Sentence	Truth value	Is it a Proposition ?
"Grass is green"	"true"	Yes
"2 + 5 = 5"	"false"	Yes
"Close the door"	-	No
"Is it hot out side ?"	-	No
"x > 2"	-	No (since x is not defined)
"x = x"	-	No
		(don't know what is "x" and "=" mean; "3 = 3" or say "air is equal to air" or "Water is equal to water" has no meaning)

- **Propositional Logic (PL)**

A proposition is a statement - which in English is a declarative sentence and Logic defines the ways of putting symbols together to form sentences that represent facts. Every proposition is either true or false. Propositional logic is also called boolean algebra.

Examples: (a) The sky is blue., (b) Snow is cold. , (c) 12 * 12=144

Propositional logic : It is fundamental to all logic.

‡ Propositions are "Sentences"; either true or false but not both.

‡ A sentence is smallest unit in propositional logic

‡ If proposition is true, then truth value is "true"; else "false"

‡ **Example ;** **Sentence** **"Grass is green";**
Truth value **" true";**
Proposition **"yes"**

Statement, Variables and Symbols

Statement : A simple statement is one that does not contain any other statement as a part. A compound statement is one that has two or more simple statements as parts called components.

Operator or connective : Joins simple statements into compounds, and joins compounds into larger compounds.

Symbols for connectives

assertion	**P**					**"p is true"**
nagation	**¬p**	**~**	**!**		**NOT**	**"p is false"**
conjunction	**p ∧ q**	**·**	**&&**	**&**	**AND**	**"both p and q are true"**
disjunction	**P v q**	**||**	**|**		**OR**	**"either p is true, or q is true, or both "**
implication	**p → q**	⊃	⇒		**if . . then**	**"if p is true, then q is true" " p implies q "**
equivalence	↔	≡	⇔		**if and only if**	**"p and q are either both true or both false"**

Truth Value

The truth value of a statement is its truth or falsity ,

p is either true or false,

~p is either true or false,

p v q is either true or false, and so on.

"**T**" or "**1**" means "**true**". and

"**F**" or "**0**" means "**false**"

Truth table is a convenient way of showing relationship between several propositions. The truth table for negation, conjunction, disjunction, implication and equivalence are shown below.

p	**q**	**¬p**	**¬q**	**p ∧ q**	**p v q**	**p→q**	**p ↔ q**	**q→p**
T	**T**	**F**	**F**	**T**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**T**	**F**	**F**	**T**
F	**T**	**T**	**F**	**F**	**T**	**T**	**F**	**F**
F	**F**	**T**	**T**	**F**	**F**	**T**	**T**	**T**

- **Tautology**

A Tautology is proposition formed by combining other propositions **(p, q, r, . . .)** which is true regardless of truth or falsehood of **p, q, r, ...**

The important tautologies are :

$$(p \rightarrow q) \leftrightarrow \neg\ [p \wedge (\neg q)] \quad \text{and} \quad (p \rightarrow q) \leftrightarrow (\neg p) \vee q$$

A proof of these tautologies, using the truth tables are given below.

Tautologies $(p \rightarrow q) \leftrightarrow \neg\ [p \wedge (\neg q)]$ and $(p \rightarrow q) \leftrightarrow (\neg p) \vee q$

Table 1: Proof of Tautologies

p	q	p→q	¬q	p ∧ (¬q)	¬ [p ∧ (¬q)]	¬p	(¬p) ∨ q
T	T	T	F	F	T	F	T
T	F	F	T	T	F	F	F
F	T	T	F	F	T	T	T
F	F	T	T	F	T	T	T

Note :

1. The entries of two columns $p \rightarrow q$ and $\neg\ [p \wedge (\neg q)]$ are identical, proves the tautology. Similarly, the entries of two columns $p \rightarrow q$ and $(\neg p) \vee q$ are identical, proves the other tautology.

2. The importance of these tautologies is that they express the membership function for $p \rightarrow q$ in terms of membership functions of either propositions **p** and $\neg q$ or $\neg p$ and **q**.

- **Equivalences**

Between Logic , Set theory and Boolean algebra.

Some mathematical equivalence between Logic and Set theory and the correspondence between Logic and Boolean algebra **(0, 1)** are given below.

Logic	Boolean Algebra (0, 1)	Set theory
T	1	
F	0	
∧	x	∩ , ∩
∨	+	∪ , U
¬	′ ie complement	(¯)
↔	=	
p, q, r	a, b, c	

■ Membership Functions obtain from facts

Consider the facts (the two tautologies)

$$(p \rightarrow q) \leftrightarrow \neg [p \wedge (\neg q)] \quad \text{and} \quad (p \rightarrow q) \leftrightarrow (\neg p) \vee q$$

Using these facts and the equivalence between logic and set theory, we can obtain membership functions for $\mu_{p\rightarrow q}(x , y)$.

From 1st fact : $\mu_{p\rightarrow q}(x , y) = 1 - \mu_{p \cap \bar{q}}(x , y)$

$= 1 - \min [\mu_p(x) , 1 - \mu_q(y)]$ **Eq (1)**

From 2nd fact : $\mu_{p\rightarrow q}(x , y) = 1 - \mu_{\bar{p} \cup q}(x , y)$

$= \max [1 - \mu_p(x) , \mu_q(y)]$ **Eq (2)**

Boolean truth table below shows the validation membership functions

Table-2 : Validation of Eq (1) and Eq (2)

$\mu_p(x)$	$\mu_q(y)$	$1 - \mu_p(x)$	$1 - \mu_q(y)$	$\max [1 - \mu_p(x) , \mu_q(y)]$	$1 - \min [\mu_p(x) , 1 - \mu_q(y)]$
1	1	0	0	1	1
1	0	0	1	0	0
0	1	1	0	1	1
0	0	1	1	1	1

Note :

1. Entries in last two columns of this table-2 agrees with the entries in table-1 for p→q , the proof of tautologies, read **T** as **1** and **F** as **0**.
2. The implication membership functions of Eq.1 and Eq.2 are not the only ones that give agreement with p→q. The others are :

$\mu_{p\rightarrow q}(x , y) = 1 - \mu_p(x) (1 - \mu_q(y))$ **Eq (3)**

$\mu_{p\rightarrow q}(x , y) = \min [1, 1 - \mu_p(x) + \mu_q(y)]$ **Eq (4)**

▪ Modus Ponens and Modus Tollens

In traditional propositional logic there are two important inference rules, Modus Ponens and Modus Tollens**.**

Modus Ponens

Premise 1 : **" x is A "**

Premise 2 : **" if x is A then y is B "** ; Consequence : **" y is B "**

Modus Ponens is associated with the implication **" A implies B " [A→B]**

In terms of propositions **p** and **q**, the Modus Ponens is expressed as

$$(p \wedge (p \rightarrow q)) \rightarrow q$$

Modus Tollens

Premise 1 : **" y is not B "**

Premise 2 : **" if x is A then y is B "** ; Consequence : **" x is not A "**

In terms of propositions **p** and **q**, the Modus Tollens is expressed as

$$(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$$

Fuzzy Logic

Like the extension of crisp set theory to fuzzy set theory, the extension of crisp logic is made by replacing the bivalent membership functions of the crisp logic with the fuzzy membership functions.

In crisp logic, the truth value acquired by the proposition are 2-valued, namely true as **1** and false as **0**.

In fuzzy logic, the truth values are multi-valued, as absolute true, partially true, absolute false etc represented numerically as real value between **0** to **1**.

Note : The fuzzy variables in fuzzy sets, fuzzy propositions, fuzzy relations etc are represented usually using symbol **~** as $\tilde{P}$ but for the purpose of easy to write it is always represented as **P** .

- **Recaps**

01 **Membership function** $\mu_A(x)$ describes the membership of the elements **x** of the base set **X** in the fuzzy set **A** .

02 **Fuzzy Intersection operator** ∩ (AND connective) applied to two fuzzy sets **A** and **B** with the membership functions $\mu_A(x)$ and $\mu_B(x)$ <u>based on min/max operations</u> is $\mu_{A \cap B} = \min [\mu_A(x) , \mu_B(x)] , \quad x \in X$ **(Eq. 01)**

03 **Fuzzy Intersection operator** ∩ (AND connective) applied to two fuzzy sets **A** and **B** with the membership functions $\mu_A(x)$ and $\mu_B(x)$ <u>based on algebraic product</u> is $\mu_{A \cap B} = \mu_A(x) \, \mu_B(x) , \quad x \in X$ **(Eq. 02)**

04 **Fuzzy Union operator** **U** (OR connective) applied to two fuzzy sets **A** and **B** with the membership functions $\mu_A(x)$ and $\mu_B(x)$ based on <u>min/max operations</u> is $\mu_{A \cup B} = \max [\mu_A(x) , \mu_B(x)] , \quad x \in X$ **(Eq. 03)**

05 **Fuzzy Union operator** **U** (OR connective) applied to two fuzzy sets **A** and **B** with the membership functions $\mu_A(x)$ and $\mu_B(x)$ based on <u>algebraic sum</u> is $\mu_{A \cup B} = \mu_A(x) + \mu_B(x) - \mu_A(x) \, \mu_B(x) , \quad x \in X$ **(Eq. 04)**

06 **Fuzzy Compliment operator** **(¯)** (NOT operation) applied to fuzzy set **A** with the membership function $\mu_A(x)$ is $\mu_{\bar{A}} = 1 - \mu_A(x) , \quad x \in X$ **(Eq. 05)**

07 **Fuzzy relations** combining two fuzzy sets by connective "min operation" is an operation by cartesian product $R : X \times Y \rightarrow [0 , 1]$.

$\mu_{R(x,y)} = \min[\mu_A(x), \mu_B(y)]$ **(Eq. 06)** or

$\mu_{R(x,y)} = \mu_A(x) \, \mu_B(y)$ **(Eq. 07)**

Example : Relation **R** between fruit colour **x** and maturity grade **y** characterized by base set linguistic colorset **X = {green, yellow, red}** maturity grade as **Y = {verdant, half-mature, mature}**

$R \triangleq$

Y x	V	h-m	m
G	1	0.5	0.0
Y	0.3	1	0.4
R	0	0.2	1

08 **Max-Min Composition -** combines the fuzzy relations variables, say (x , y) and (y , z) ; $x \in A , \quad y \in B , \quad z \in C$.

consider the relations :

$$R_1(x , y) = \{ ((x , y) , \mu_{R1}(x , y)) \mid (x , y) \in A \times B \}$$

$$R_2(y , z) = \{ ((y , y) , \mu_{R1}(y , z)) \mid (y , z) \in B \times C \}$$

The domain of R_1 is $A \times B$ and the domain of R_2 is $B \times C$

max-min composition denoted by $R_1 \circ R_2$ with membership function $\mu_{R1 \circ R2}$

$$R_1 \circ R_2 = \{ ((x , z) , \max_{y} (\min (\mu_{R1}(x , y) , \mu_{R2}(y , z))))\} , \quad (x , z) \in A \times C , \; y \in B \qquad \textbf{(Eq. 08)}$$

Thus $R_1 \circ R_2$ is relation in the domain $A \times C$

- **Fuzzy Propositional**

  A fuzzy proposition is a statement **P** which acquires a fuzzy truth value **T(P)** .

  Example :

  P : Ram is honest

  T(P) = 0.8 , means **P** is partially true.

  T(P) = 1 , means **P** is absolutely true.

- **Fuzzy Connectives**

  The fuzzy logic is similar to crisp logic supported by connectives.

  Table below illustrates the definitions of fuzzy connectives.

Table : Fuzzy Connectves

Connective	Symbols	Usage	Definition
Nagation	$\neg$	$\neg$ P	1 – T(P)
Disjuction	$\vee$	P $\vee$ Q	Max[T(P) , T(Q)]
Conjuction	$\wedge$	P $\wedge$ Q	min[T(P) , T(Q)]
Implication	$\Rightarrow$	P $\Rightarrow$ Q	$\neg$P $\vee$ Q = max (1-T(P), T(Q)]

Here **P , Q** are fuzzy proposition and **T(P) , T(Q)** are their truth values.

- the **P** and **Q** are related by the $\Rightarrow$ operator are known as antecedents and consequent respectively.
- as crisp logic, here in fuzzy logic also the operator $\Rightarrow$ represents **IF-THEN** statement like,

  IF x is A THEN y is B, is equivalent to

  R = (A x B) U ($\neg$ A x Y)

  the membership function of **R** is given by

  μ_R **(x , y) = max [min (μ_A (x) , μ_B (y)) , 1 – μ_A (x)]**
- For the compound implication statement like

  IF x is A THEN y is B, ELSE y is C is equivalent to

  R = (A x B) U ($\neg$ A x C)

  the membership function of **R** is given by

  μ_R **(x , y) = max [min (μ_A (x) , μ_B (y)) , min (1 – μ_A (x), μ_C (y))]**

Example 1 : (Ref : Previous slide)

P : Mary is efficient , $T(P) = 0.8$,

Q : Ram is efficient , $T(Q) = 0.65$,

$\neg$ P : Mary is efficient , $T(\neg P) = 1 - T(P) = 1- 0.8 = 0.2$

$P \wedge Q$: Mary is efficient and so is Ram, i.e.

$T(P \wedge Q) = \min (T(P), T(Q)) = \min (0.8, 0.65)) = 0.65$

$P \vee Q$: Either Mary or Ram is efficient i.e.

$T(P \vee Q) = \max (T(P), T(Q)) = \max (0.8, 0.65)) = 0.8$

$P \Rightarrow Q$: If Mary is efficient then so is Ram, i.e.

$T(P \Rightarrow Q) = \max (1- T(P), T(Q)) = \max (0.2, 0.65)) = 0.65$

Example 2 : (Ref : Previous slide on fuzzy connective)

Let **X = {a, b, c, d}** ,

A = {(a, 0) (b, 0.8) (c, 0.6) (d, 1)}

B = {(1, 0.2) (2, 1) (3, 0.8) (4, 0)}

C = {(1, 0) (2, 0.4) (3, 1) (4, 0.8)}

Y = { 1, 2, 3, 4} the universe of discourse could be viewed as

{ (1, 1) (2, 1) (3, 1) (4, 1) }

i.e., a fuzzy set all of whose elements x have $\mu(x) = 1$

Determine the implication relations

(i) If x is A THEN y is B

(ii) If x is A THEN y is B Else y is C

Solution

To determine implication relations (i) compute :

The operator ⇒ represents **IF-THEN** statement like,

IF x is A THEN y is B, is equivalent to $R = (A \times B) \cup (\neg A \times Y)$ and

the membership function **R** is given by

$\mu_R (x , y) = \max [\min (\mu_A (x) , \mu_B (y)) , 1 - \mu_A (x)]$

Fuzzy Intersection A x B is defined as :

for all x in the set X,

(A ∩ B)(x) = min [A(x), B(x)],

***A x B* =**

B A	1	2	3	4
a	0	0	0	0
b	0.2	0.8	0.8	0
c	0.2	0.6	0.6	0
d	0.2	1	0.8	0

Fuzzy Intersection ¬*A* x Y is defined as :

for all x in the set X

(¬A ∩ Y)(x) = min [A(x), Y(x)],

¬*A x Y* =

y A	1	2	3	4
a	1	1	1	1
b	0.2	0.2	0.2	0.2
c	0.4	0.4	0.4	0.4
d	0	0	0	0

Fuzzy Union is defined as **(A ∪ B)(x) = max [A(x), B(x)] for all x ∈ X**

Therefore **R = (A x B) U (¬ A x Y)** gives

***R* =**

y x	1	2	3	4
a	1	1	1	1
b	0.2	0.8	0.8	0
c	0.4	0.6	0.6	0.4
d	0.2	1	0.8	0

This represents **If x is A THEN y is B** ie **T(A ⇒ B) = max (1- T(A), T(B))**

To determine implication relations (ii) compute : (Ref : Previous slide)

Given **X = {a, b, c, d}** ,

A = {(a, 0) (b, 0.8) (c, 0.6) (d, 1)}

B = {(1, 0.2) (2, 1) (3, 0.8) (4, 0)}

C = {(1, 0) (2, 0.4) (3, 1) (4, 0.8)}

Here, the operator ⇒ represents **IF-THEN-ELSE** statement like,

IF x is A THEN y is B Else y is C, is equivalent to

R = (A x B) U (¬ A x C) and

the membership function of **R** is given by

$$\mu_R (x , y) = \max [\min (\mu_A (x) , \mu_B (y)) , \min(1 - \mu_A (x), \mu_C (y)]$$

Fuzzy Intersection A x B is defined as :
for all x in the set X,
(A ∩ B)(x) = min [A(x), B(x)],

A x B =

A B	1	2	3	4
a	0	0	0	0
b	0.2	0.8	0.8	0
c	0.2	0.6	0.6	0
d	0.2	1	0.8	0

Fuzzy Intersection ¬A x Y is defined as :
for all x in the set X
(¬A ∩ C)(x) = min [A(x), C(x)],

***¬A x C* =**

A y	1	2	3	4
a	0	0.4	1	0.8
b	0.2	0.2	0.2	0.2
c	0.4	0.4	0.4	0.4
d	0	0	0	0

Fuzzy Union is defined as **(A ∪ B)(x) = max [A(x), B(x)] for all x ∈ X**

Therefore **R = (A x B) U (¬ A x C) gives**

***R* =**

x y	1	2	3	4
a	1	1	1	1
b	0.2	0.8	0.8	0
c	0.4	0.6	0.6	0.4
d	0.2	1	0.8	0

This represents **If x is A THEN y is B Else y is C**

- **Fuzzy Quantifiers**

In crisp logic, the predicates are quantified by quantifiers.

Similarly, in fuzzy logic the propositions are quantified by quantifiers.

There are two classes of fuzzy quantifiers :

- Absolute quantifiers and
- Relative quantifiers

Examples :

Absolute quantifiers	**Relative quantifiers**
round about 250	almost
much greater than 6	about
some where around 20	most

Fuzzification

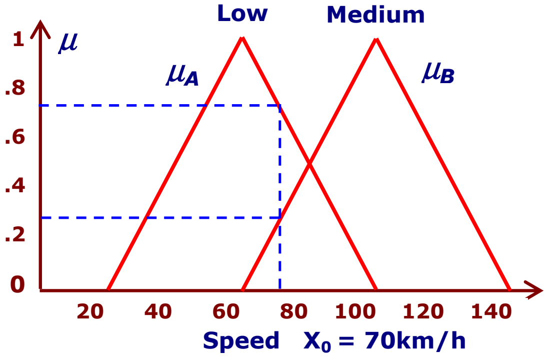
The fuzzification is a process of transforming crisp values into grades of membership for linguistic terms of fuzzy sets.

The purpose is to allow a fuzzy condition in a rule to be interpreted.

- **Fuzzification of the car speed**

Example 1 : Speed X_0 = 70km/h

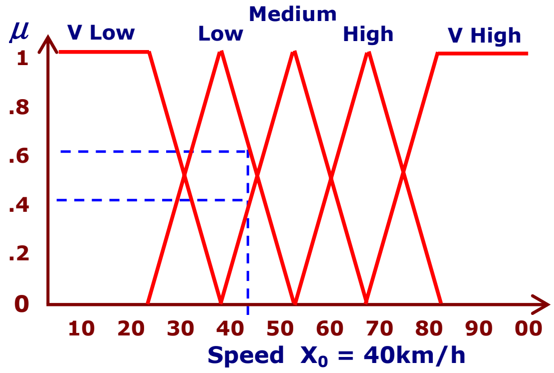
Fig below shows the fuzzification of the car speed to characterize a low and a medium speed fuzzy set.



Given car speed value **X_0=70km/h** : grade **$\mu_A(x_0)$ = 0.75** belongs to fuzzy low, and grade **$\mu_B(x_0)$ = 0.25** belongs to fuzzy medium

Characterizing two grades, low and medium speed fuzzy set

Example 2 : Speed X_0 = 40km/h



Given car speed value **X_0=40km/h** : grade **$\mu_A(x_0)$ = 0.6** belongs to fuzzy low, and grade **$\mu_B(x_0)$ = 0.4** belongs to fuzzy medium.

Characterizing five grades, Very low, low, medium, high and very high speed fuzzy set

Fuzzy Inference

Fuzzy Inferencing is the core element of a fuzzy system.

Fuzzy Inferencing combines - the facts obtained from the fuzzification with the rule base, and then conducts the fuzzy reasoning process.

Fuzzy Inference is also known as **approximate reasoning**.

Fuzzy Inference is computational procedures used for evaluating linguistic descriptions. Two important inferring procedures are

- Generalized Modus Ponens (GMP)
- Generalized Modus Tollens (GMT)

- **Generalized Modus Ponens (GMP)**

This is formally stated as

$$\begin{array}{l} \textbf{If } x \textbf{ is } A \textbf{ THEN } y \textbf{ is } B \\ \qquad \underline{x \textbf{ is } \neg A} \\ \qquad y \textbf{ is } \neg B \end{array}$$

where **A , B , ¬A , ¬B** are fuzzy terms.

Note : Every fuzzy linguistic statements above the line is analytically known and what is below the line is analytically unknown.

To compute the membership function **¬B** , the max-min composition of fuzzy set **¬A** with **R(x , y)** which is the known implication relation (**IF-THEN**) is used. i.e. $\neg B = \neg A \circ R(x, y)$

In terms of membership function

$\mu_{\neg B}(y) = \max(\min(\mu_{\neg A}(x), \mu_R(x, y)))$ where

$\mu_{\neg A}(x)$ is the membership function of **¬A** ,

$\mu_R(x, y)$ is the membership function of the implication relation and

$\mu_{\neg B}(y)$ is the membership function of **¬B**

- **Generalized Modus Tollens (GMT)**

This is formally stated as

If x is A THEN y is B

$$\frac{\textbf{y is } \neg\textbf{B}}{\textbf{x is } \neg\textbf{A}}$$

where **A , B , ¬A** , **¬B** are fuzzy terms.

Note : Every fuzzy linguistic statements above the line is analytically known and what is below the line is analytically unknown.

To compute the membership function **¬A** , the max-min composition of fuzzy set **¬B** with **R(x , y)** which is the known implication relation (**IF-THEN**) is used. i.e. $\neg A = \neg B \circ R(x, y)$

In terms of membership function

$\mu_{\neg A}(y) = \max(\min(\mu_{\neg B}(x), \mu_R(x, y)))$ where

$\mu_{\neg B}(x)$ is the membership function of **¬B** ,

$\mu_R(x, y)$ is the membership function of the implication relation and

$\mu_{\neg A}(y)$ is the membership function of **¬A**

Example :

Apply the fuzzy Modus Ponens rules to deduce Rotation is quite slow?

Given **:**

(i) If the temperature is high then then the rotation is slow.

(ii) The temperature is very high.

Let **H (High) , VH (Very High) , S (Slow) and QS (Quite Slow)** indicate the associated fuzzy sets.

Let the set for temperatures be **X = {30, 40, 50, 60, 70, 80, 90, 100}** , and

Let the set of rotations per minute be **Y = {10, 20, 30, 40, 50, 60}** and

H = {(70, 1) (80, 1) (90, 0.3)}

VH = {(90, 0.9) (100, 1)}

QS = {10, 1) (20, 08) }

S = {(30, 0.8) (40, 1) (50, 0.6)

To derive **R(x, y)** representing the implication relation (i) above, compute

$R(x, y) = \max(H \times S, \neg H \times Y)$

$$H \text{ x } S = \begin{array}{ccccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ 30 & 0 & 0 & 0 & 0 & 0 & 0 \\ 40 & 0 & 0 & 0 & 0 & 0 & 0 \\ 50 & 0 & 0 & 0 & 0 & 0 & 0 \\ 60 & 0 & 0 & 0 & 0 & 0 & 0 \\ 70 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 80 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 90 & 0 & 0 & 0.3 & 0.3 & 0.3 & 0 \\ 100 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$H \text{ x } Y = \begin{array}{ccccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ \mathit{30} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathit{40} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathit{50} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathit{60} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathit{70} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathit{80} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathit{90} & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ \mathit{100} & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$R(x,Y) = \begin{array}{ccccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ 30 & 1 & 1 & 1 & 1 & 1 & 1 \\ 40 & 1 & 1 & 1 & 1 & 1 & 1 \\ 50 & 1 & 1 & 1 & 1 & 1 & 1 \\ 60 & 1 & 1 & 1 & 1 & 1 & 1 \\ 70 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 80 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 90 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 100 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

To deduce Rotation is quite slow, we make use of the composition rule

QS = VH o R (x, y)

$$= [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0.9 \;\; 1] \quad \text{x} \quad \begin{array}{ccccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ 30 & 1 & 1 & 1 & 1 & 1 & 1 \\ 40 & 1 & 1 & 1 & 1 & 1 & 1 \\ 50 & 1 & 1 & 1 & 1 & 1 & 1 \\ 60 & 1 & 1 & 1 & 1 & 1 & 1 \\ 70 & 0 & 0 & 0 & 0 & 0 & 0 \\ 80 & 0 & 0 & 0 & 0 & 0 & 0 \\ 90 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 100 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$= [1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\;]$$

Fuzzy Rule Based System

The fuzzy linguistic descriptions are formal representation of systems made through fuzzy IF-THEN rule. They encode knowledge about a system in statements of the form :

IF (a set of conditions) are satisfied THEN (a set of consequents) can be inferred.

IF (x_1 is A_1, x_2 is A_2, xn is An) THEN (y_1 is B_1, y_2 is B_2, yn is Bn)

where linguistic variables **x_i, y_j** take the values of fuzzy sets **A_i** and **B_j** respectively.

Example :

IF there is "*heavy*" rain and "*strong*" winds
THEN there must "*severe*" flood warnings.

Here, **heavy** , **strong** , and **severe** are fuzzy sets qualifying the variables *rain, wind,* and *flood* warnings respectively.

A collection of rules referring to a particular system is known as a fuzzy rule base. If the conclusion **C** to be drawn from a rule base **R** is the conjunction of all the individual consequents **C i** of each rule , then

$\mathbf{C = C_1 \cap C_2 \cap . . . \cap C_n}$ where

$\mu_c (y) = \min (\mu_{c1}(y), \mu_{c2}(y) , \mu_{cn}(y)) , \quad \forall y \in Y$

where **Y** is universe of discourse.

On the other hand, if the conclusion **C** to be drawn from a rule base **R** is the disjunction of the individual consequents of each rule, then

$\mathbf{C = C_1 \cup C_2 \cup . . . \cup C_n}$ where

$\mu_c (y) = \max (\mu_{c1} (y), \mu_{c2}(y) , \mu_{cn} (y)) , \quad \forall y \in Y$ where

Y is universe of discourse.

Defuzzification

In many situations, for a system whose output is fuzzy, it is easier to take a crisp decision if the output is represented as a single quantity. This conversion of a single crisp value is called Defuzzification.

Defuzzification is the reverse process of fuzzification.

The typical Defuzzification methods are

– Centroid method,

– Center of sums,

– Mean of maxima.

Centroid method

It is also known as the "center of gravity" of area method.

It obtains the centre of area **(x*)** occupied by the fuzzy set .

For discrete membership function, it is given by

$$x^* = \frac{\sum_{i=1}^{n} xi\ \mu\,(xi)}{\sum_{i=1}^{n} \mu\,(xi)} \quad \text{where}$$

n represents the number elements in the sample, and

x_i are the elements, and

$\mu\,(x_i)$ is the membership function.

Module - II

Probabilistic Reasoning

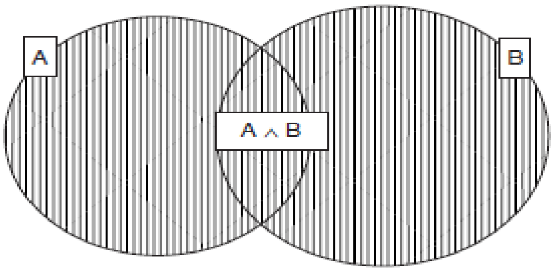
- Probability theory is used to discuss events, categories, and hypotheses about which there is not 100% certainty.
- We might write A→B, which means that if A is true, then B is true. If we are unsure whether A is true, then we cannot make use of this expression.
- In many real-world situations, it is very useful to be able to talk about things that lack certainty. For example, what will the weather be like tomorrow? We might formulate a very simple hypothesis based on general observation, such as "it is sunny only 10% of the time, and rainy 70% of the time". We can use a notation similar to that used for predicate calculus to express such statements:

  $P(S) = 0.1$

  $P(R) = 0.7$

  The first of these statements says that the probability of S ("it is sunny") is 0.1. The second says that the probability of R is 0.7. Probabilities are always expressed as real numbers between 0 and 1. A probability of 0 means "definitely not" and a probability of 1 means "definitely so." Hence, P(S) = 1 means that it is always sunny.

- Many of the operators and notations that are used in prepositional logic can also be used in probabilistic notation. For example, $P(\neg S)$ means "the probability that it is not sunny"; $P(S \wedge R)$ means "the probability that it is both sunny and rainy." $P(A \vee B)$, which means "the probability that either A is true or B is true," is defined by the following rule: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



- The notation P(B|A) can be read as "the probability of B, given A." This is known as conditional probability—it is conditional on A. In other words, it states the probability that B is true, given that we already know that A is true. P(B|A) is defined by the following rule: Of course, this rule cannot be used in cases where P(A) = 0.

- For example, let us suppose that the likelihood that it is both sunny and rainy at the same time is 0.01. Then we can calculate the probability that it is rainy, given that it is sunny as follows:

$$
\begin{aligned}
P(R|S) &= \frac{P(R \wedge S)}{P(S)} \\
&= \frac{0.01}{0.1} \\
&= 0.1
\end{aligned}
$$

- The basic approach statistical methods adopt to deal with uncertainty is via the axioms of probability:
  - ➢ Probabilities are (real) numbers in the range 0 to 1.
  - ➢ A probability of *P*(*A*) = 0 indicates total uncertainty in *A*, *P*(*A*) = 1 total certainty and values in between some degree of (un)certainty.
  - ➢ Probabilities can be calculated in a number of ways.
- Probability = (number of desired outcomes) / (total number of outcomes)

  So given a pack of playing cards the probability of being dealt an ace from a full normal deck is 4 (the number of aces) / 52 (number of cards in deck) which is 1/13. Similarly the probability of being dealt a spade suit is 13 / 52 = 1/4.

  If you have a choice of number of items *k* from a set of items *n* then the $C^n_k = \frac{n!}{k!(n-k)!}$ formula is applied to find the number of ways of making this choice. (! = factorial).

  So the chance of winning the national lottery (choosing 6 from 49) is $\frac{49!}{6!43!} = 13,983,816$ to 1.
- Conditional probability, *P*(*A*|*B*), indicates the probability of of event *A* given that we know event *B* has occurred.
- *A* Bayesian Network *is a* directed acyclic graph*:*
  - ➢ A graph where the directions are links which indicate dependencies that exist between nodes.
  - ➢ Nodes represent propositions about events or events themselves.
  - ➢ Conditional probabilities quantify the strength of dependencies.

- Consider the following example:
  - The probability, $P(s_1)$ that my car won't start.
  - If my car won't start then it is likely that
    - The battery is flat or
    - The staring motor is broken.
- In order to decide whether to fix the car myself or send it to the garage I make the following decision:
  - If the headlights do not work then the battery is likely to be flat so i fix it myself.
  - If the starting motor is defective then send car to garage.
  - If battery and starting motor both gone send car to garage.
- The network to represent this is as follows:

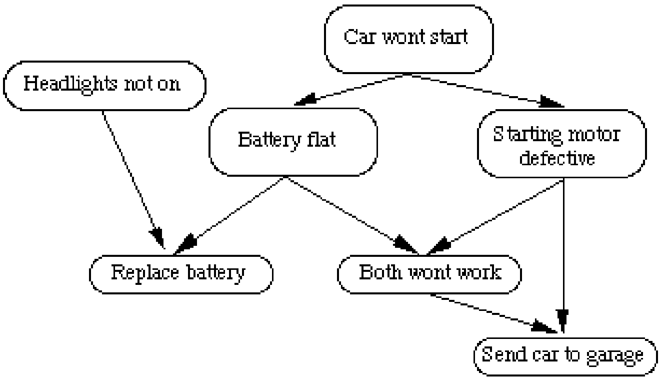


Fig. A simple Bayesian network

Bayesian probabilistic inference

- Bayes' theorem can be used to calculate the probability that a certain event will occur or that a certain proposition is true
- The theorem is stated as follows:

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

.

  - P(B) is called the prior probability of B. P(B|A), as well as being called the conditional probability, is also known as the posterior probability of B.
  - $P(A \wedge B) = P(A|B)P(B)$

  - Note that due to the commutativity of ∧, we can also write
  - P(A ∧B) = P(B|A)P(A)
  - Hence, we can deduce: P(B|A)P(A) = P(A|B)P(B)
  - This can then be rearranged to give Bayes' theorem:

$$P(B|A) = \frac{P(A|B)\cdot P(B)}{P(A)}$$

- Bayes Theorem states:

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

  - This reads that given some evidence *E* then probability that hypothesis H_i is true is equal to the ratio of the probability that *E* will be true given H_i times the *a priori* evidence on the probability of H_i and the sum of the probability of *E* over the set of all hypotheses times the probability of these hypotheses.
  - The set of all hypotheses must be mutually exclusive and exhaustive.
  - Thus to find if we examine medical evidence to diagnose an illness. We must know all the prior probabilities of find symptom and also the probability of having an illness based on certain symptoms being observed.
- Bayesian statistics lie at the heart of most statistical reasoning systems. How is Bayes theorem exploited?
  - The key is to formulate problem correctly:
    P(*A*|*B*) states the probability of *A* given only *B*'s evidence. *If* there is other relevant evidence then it must also be considered.
- All events must be *mutually exclusive*. However in real world problems events are not generally unrelated. For example in diagnosing measles, the symptoms of spots and a fever are related. This means that computing the conditional probabilities gets complex.

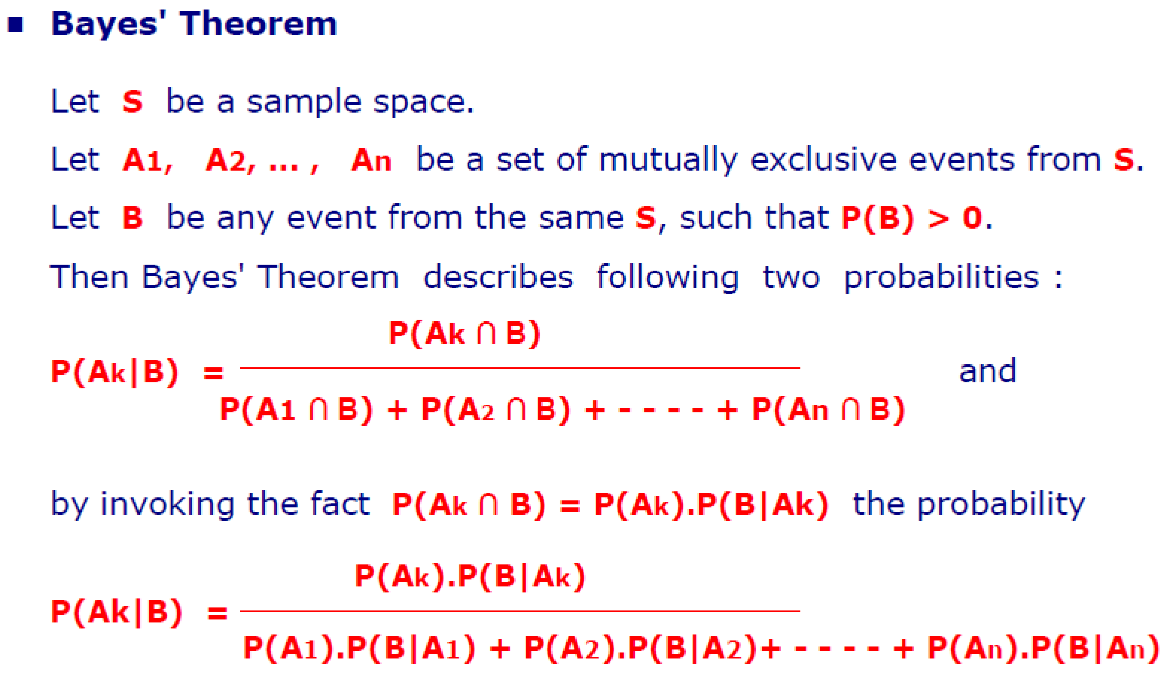
  In general if a prior evidence, *p* and some new observation, *N* then computing

$$P(H|N,p) = P(H|N)\frac{P(p|N,H)}{P(p|N)}$$

  grows exponentially for large sets of *p*

- All events must be *exhaustive*. This means that in order to compute all probabilities the set of possible events must be closed. Thus if new information arises the set must be created afresh and *all* probabilities recalculated.
- Thus Simple Bayes rule-based systems are not suitable for uncertain reasoning.
  - ➢ Knowledge acquisition is very hard.
  - ➢ Too many probabilities needed -- too large a storage space.
  - ➢ Computation time is too large.
  - ➢ Updating new information is difficult and time consuming.
  - ➢ Exceptions like ``none of the above'' cannot be represented.
  - ➢ Humans are not very good probability estimators.
- However, Bayesian statistics still provide the core to reasoning in many uncertain reasoning systems with suitable enhancement to overcome the above problems. We will look at three broad categories:
  - ➢ Certainty factors
  - ➢ Dempster-Shafer models
  - ➢ Bayesian networks.

Bayesian networks are also called Belief Networks or Probabilistic Inference Networks.



■ **Bayes' Theorem**

Let **S** be a sample space.

Let **A1, A2, ... , An** be a set of mutually exclusive events from **S**.

Let **B** be any event from the same **S**, such that **P(B) > 0**.

Then Bayes' Theorem describes following two probabilities :

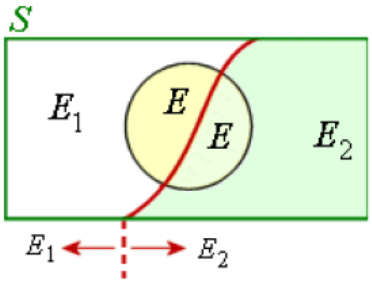
$$P(A_k|B) = \frac{P(A_k \cap B)}{P(A_1 \cap B) + P(A_2 \cap B) + - - - - + P(A_n \cap B)} \quad \text{and}$$

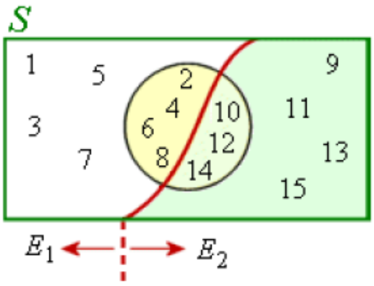
by invoking the fact $P(A_k \cap B) = P(A_k).P(B|A_k)$ the probability

$$P(A_k|B) = \frac{P(A_k).P(B|A_k)}{P(A_1).P(B|A_1) + P(A_2).P(B|A_2) + - - - - + P(A_n).P(B|A_n)}$$

Application Of Bayes Therom:

- Let **S** be a sample space.
- Let **E_1** and **E_2** be two mutually exclusive events forming a partition of the sample space **S**
- Let **E** be any event of the sample space such that **P(E) ≠ 0**.





Recall from Conditional Probability

The notation **$P(E_1 \mid E)$** means "the probability of the event **E_1** given that **E** has already occurred"**.**

Recall from Conditional Probability

The notation **$P(E_1 \mid E)$** means "the probability of the event **E_1** given that **E** has already occurred"**.**

- The sample space **S** is described as "the integers 1 to 15" and is partitioned into :

  E_1 = "the integers 1 to 8" and

  E_2 = "the integers 9 to 15".

- If **E** is the event "**even number**" then the probabilities for the situation described by Baye's Theorem can be calculated in two ways, both giving same results.

$$P(E_1|E) = \frac{P(E_1 \cap E)}{P(E_1 \cap E) + P(E_2 \cap E)} = \frac{4 / 15}{(4 / 15) + (3 / 15)} = 4 / 7$$

$$P(E1|E) = \frac{P(E_1).P(E|E_1)}{P(E_1).P(E|E_1) + P(E_2).P(E|E_2)} = \frac{8/15 \times 4/8}{(8/15 \times 4/8) + (7/15 \times 3/15)} = 4/7$$

Thus Bayes' Theorem can be extended for Mutually Exclusive Events as :

$$P(Ei \mid E) = \frac{P(Ei \cap E)}{P(E1 \cap E) + P(E2 \cap E) + \ldots\ldots + P(Ek \cap E)}$$

Clinical Example:

In a clinic, the probability of the patients having HIV virus is **0.15**.

A blood test done on patients :

If patient has virus, then the test is **+ve** with probability **0.95**.

If the patient does not have the virus, then the test is **+ve** with probability **0.02.**

Assign labels to events : **H** = patient has virus; **P** = test +ve

Given : **P(H) = 0.15 ; P(P|H) = 0.95 ; P(P|¬H) = 0.02**

Find :

If the test is **+ve** what are the probabilities that the patient

i) has the virus ie **P(H|P)** ; ii) does not have virus ie **P(¬H|P)** ;

If the test is **-ve** what are the probabilities that the patient

iii) has the virus ie **P(H|¬P)** ; iv) does not have virus ie **P(¬H|¬P)** ;

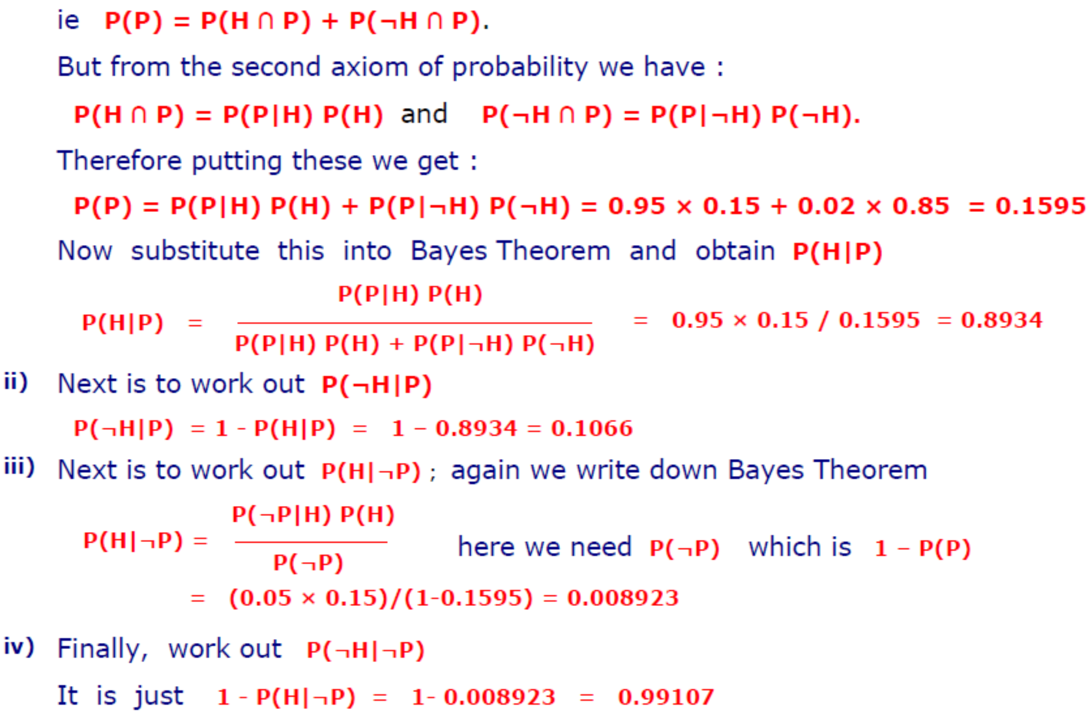
Calculations :

i) For **P(H|P)** we can write down Bayes Theorem as

P(H|P) = [P(P|H) P(H)] / P(P)

We know **P(P|H)** and **P(H)** but not **P(P)** which is probability of a **+ve** result.

There are two cases, that a patient could have a **+ve** result, stated below :



ie $P(P) = P(H \cap P) + P(\neg H \cap P)$.

But from the second axiom of probability we have :

$P(H \cap P) = P(P|H)\ P(H)$ and $P(\neg H \cap P) = P(P|\neg H)\ P(\neg H)$.

Therefore putting these we get :

$P(P) = P(P|H)\ P(H) + P(P|\neg H)\ P(\neg H) = 0.95 \times 0.15 + 0.02 \times 0.85 = 0.1595$

Now substitute this into Bayes Theorem and obtain $P(H|P)$

$$P(H|P) = \frac{P(P|H)\ P(H)}{P(P|H)\ P(H) + P(P|\neg H)\ P(\neg H)} = 0.95 \times 0.15 / 0.1595 = 0.8934$$

ii) Next is to work out $P(\neg H|P)$

$P(\neg H|P) = 1 - P(H|P) = 1 - 0.8934 = 0.1066$

iii) Next is to work out $P(H|\neg P)$; again we write down Bayes Theorem

$$P(H|\neg P) = \frac{P(\neg P|H)\ P(H)}{P(\neg P)}$$ here we need $P(\neg P)$ which is $1 - P(P)$

$= (0.05 \times 0.15)/(1-0.1595) = 0.008923$

iv) Finally, work out $P(\neg H|\neg P)$

It is just $1 - P(H|\neg P) = 1 - 0.008923 = 0.99107$

Definition and importance of knowledge

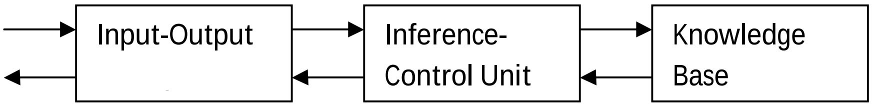
- Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact, or state of knowing
- Knowledge is having familiarity with language, concepts, procedures, rules, ideas, abstractions, places, customs, facts, and associations, coupled with an ability to use theses notions effectively in modeling different aspects of the world
- The meaning of knowledge is closely related to the meaning of intelligence
- Intelligent requires the possession of and access to knowledge
- A common way to represent knowledge external to a computer or a human is in the form of written language
- Example:
  - ✓ Ramu is tall – This expresses a simple fact, an attribute possessed by a person
  - ✓ Ramu loves his mother – This expresses a complex binary relation between two persons
- Knowledge may be declarative or procedural
  - ✓ Procedural knowledge is compiled knowledge related to the performance of some task. For example, the steps used to solve an algebraic equation

  - ✓ Declarative knowledge is passive knowledge expressed as statements of facts about the world. For example, personnel data in a database, such data are explicit pieces of independent knowledge

- Knowledge includes and requires the use of data and information
- Knowledge combines relationships, correlations, dependencies, and the notion of gestalt with data and information
- Belief is a meaningful and coherent expression. Thus belief may be true or false
- Hypothesis is defined as a belief which is backed up with some supporting evidence, but it may still be false
- Knowledge is true justified belief
- Epistemology is the study of the nature of knowledge
- Metaknowledge is knowledge about knowledge, that is, knowledge about what we know

Knowledge Based Systems

- Systems that depend on a rich base of knowledge to perform difficult tasks
- It includes work in vision, learning, general problem solving and natural language understanding
- The systems get their power from the expert knowledge that has been coded into facts, rules, heuristics and procedures.
- In Fig 2.1, the knowledge is stored in a knowledge base separate from the control and inferencing components
- It is possible to add new knowledge or refine existing knowledge without recompiling the control and inferencing programs
- Components of a knowledge based system



Representation of knowledge

- The object of a knowledge representation is to express knowledge in a computer tractable form, so that it can be used to enable our AI agents to perform well.
- A knowledge representation language is defined by two aspects:
  - ✓ Syntax The syntax of a language defines which configurations of the components of the language constitute valid sentences.
  - ✓ Semantics The semantics defines which facts in the world the sentences refer to, and hence the statement about the world that each sentence makes.
- Suppose the language is arithmetic, then ‘x’, ‘=’ and ‘y’ are components (or symbols or words) of the language the syntax says that ‘x = y’ is a valid sentence in the language, but ‘= = x y’ is not the semantics say that ‘x = y’ is false if y is bigger than x, and true otherwise
- The requirements of a knowledge representation are:
  - ✓ Representational Adequacy – the ability to represent all the different kinds of knowledge that might be needed in that domain.
  - ✓ Inferential Adequacy – the ability to manipulate the representational structures to derive new structures (corresponding to new knowledge) from existing structures.
  - ✓ Inferential Efficiency – the ability to incorporate additional information into the knowledge structure which can be used to focus the attention of the inference mechanisms in the most promising directions.
  - ✓ Acquisitional Efficiency – the ability to acquire new information easily. Ideally the agent should be able to control its own knowledge acquisition, but direct insertion of information by a ‘knowledge engineer’ would be acceptable. Finding a system that optimizes these for all possible domains is not going to be feasible.
- In practice, the theoretical requirements for good knowledge representations can usually be achieved by dealing appropriately with a number of practical requirements:
  - ✓ The representations need to be complete – so that everything that could possibly need to be represented can easily be represented.

  - ✓ They must be computable – implementable with standard computing procedures.
  - ✓ They should make the important objects and relations explicit and accessible – so that it is easy to see what is going on, and how the various components interact.
  - ✓ They should suppress irrelevant detail – so that rarely used details don’t introduce unnecessary complications, but are still available when needed.
  - ✓ They should expose any natural constraints – so that it is easy to express how one object or relation influences another.
  - ✓ They should be transparent – so you can easily understand what is being said.
  - ✓ The implementation needs to be concise and fast – so that information can be stored, retrieved and manipulated rapidly.
- The four fundamental components of a good representation
  - ✓ The lexical part – that determines which symbols or words are used in the representation’s vocabulary.
  - ✓ The structural or syntactic part – that describes the constraints on how the symbols can be arranged, i.e. a grammar.
  - ✓ The semantic part – that establishes a way of associating real world meanings with the representations.
  - ✓ The procedural part – that specifies the access procedures that enables ways of creating and modifying representations and answering questions using them, i.e. how we generate and compute things with the representation.
- Knowledge Representation in Natural Language
  - ✓ Advantages of natural language
    - o It is extremely expressive – we can express virtually everything in natural language (real world situations, pictures, symbols, ideas, emotions, reasoning).
    - o Most humans use it most of the time as their knowledge representation of choice
  - ✓ Disadvantages of natural language
    - o Both the syntax and semantics are very complex and not fully understood.
    - o There is little uniformity in the structure of sentences.

- It is often ambiguous – in fact, it is usually ambiguous.

Knowledge Organization

- The organization of knowledge in memory is key to efficient processing
- Knowledge based systems performs their intended tasks
- The facts and rules are easy to locate and retrieve. Otherwise much time is wasted in searching and testing large numbers of items in memory
- Knowledge can be organized in memory for easy access by a method known as indexing
- As a result, the search for some specific chunk of knowledge is limited to the group only

Knowledge Manipulation

- Decisions and actions in knowledge based systems come from manipulation of the knowledge
- The known facts in the knowledge base be located, compared, and altered in some way
- This process may set up other subgoals and require further inputs, and so on until a final solution is found
- The manipulations are the computational equivalent of reasoning. This requires a form of inference or deduction, using the knowledge and inferring rules.
- All forms of reasoning requires a certain amount of searching and matching.
- The searching and matching operations consume greatest amount of computation time in AI systems
- It is important to have techniques that limit the amount of search and matching required to complete any given task

Module 3

Matching techniques:

Matching is the process of comparing two or more structures to discover their likenesses or differences. The structures may represent a wide range of objects including physical entities, words or phrases in some language, complete classes of things, general concepts, relations between complex entities, and the like. The representations will be given in one or more of the formalisms like FOPL, networks, or some other scheme, and matching will involve comparing the component parts of such structures.

Matching is used in a variety of programs for different reasons. It may serve to control the sequence of operations, to identify or classify objects, to determine the best of a number of different alternatives, or to retrieve items from a database. It is an essential operation such diverse programs as speech recognition, natural language understanding, vision, learning, automated reasoning, planning, automatic programming, and expert systems, as well as many others.

In its simplest form, matching is just the process of comparing two structures or patterns for equality. The match fails if the patterns differ in any aspect. For example, a match between the two character strings acdebfba and acdebeba fails on an exact match since the strings differ in the sixth character positions.

In more complex cases the matching process may permit transformations in the patterns in order to achieve an equality match. The transformation may be a simple change of some variables to constants, or ti may amount to ignoring some components during the match operation. For example, a pattern matching variable such as ?x may be used to permit successful matching between the two patterns (a b (c d) e) and (a b ?x e) by binding ?x to (c, d). Such matching are usually restricted in some way, however, as is the case with the unification of two classes where only consistent bindings are permitted. Thus, two patterns such as

(a b (c d) e f) and (a b ?x e ?x)

would not match since ?x could not be bound to two different constants.

In some extreme cases, a complete change of representational form may be required in either one or both structures before a match can be attempted. This will be the case, for example, when one visual object is represented as a vector of pixel gray levels and objects to be matched are represented as descriptions in predicate logic or some other high level statements. A direct comparison is impossible unless one form has been transformed into the other.

In subsequent chapters we will see examples of many problems where exact matches are inappropriate, and some form of partial matching is more meaningful. Typically in such cases, one is interested in finding a best match between pairs of structures. This will be the case in object classification problems, for example, when object descriptions are subject to corruption by noise or distortion. In such cases, a measure of the degree of match may also be required.

Other types of partial matching may require finding a match between certain key elements while ignoring all other elements in the pattern. For example, a human language input unit should be flexible enough to recognize any of the following three statements as expressing a choice of preference for the low-calorie food item

I prefer the low-calorie choice.

I want the low-calorie item

The low-calorie one please.

Recognition of the intended request can be achieved by matching against key words in a template containing “low-calorie” and ignoring other words except, perhaps, negative modifiers.

Finally, some problems may obviate the need for a form of fuzzy matching where an entity’s degree of membership in one or more classes is appropriate. Some classification problems will apply here if the boundaries between the classes are not distinct, and an object may belong to more than one class.

Fig 8.1 illustrates the general match process where an input description is being compared with other descriptions. As stressed earlier, their term object is used here in a general sense. It does not necessarily imply physical objects. All objects will be represented in some formalism such as a vector of attribute values, prepositional logic or FOPL statements, rules, frame-like structures, or other scheme. Transformations, if required, may involve simple instantiations or unifications among clauses or more complex operations such as transforming a two-dimensional scene to a description in some formal language. Once the descriptions have been transformed into the same schema, the matching process is performed element-by-element using a relational or other test (like equality or ranking). The test results may then be combined in some way to provide an overall measure of similarity. The choice of measure will depend on the match criteria and representation scheme employed.

The output of the matcher is a description of the match. It may be a simple yes or no response or a list of variable bindings, or as complicated as a detailed annotation of the similarities and differences between the matched objects.

`To summarize then, matching may be exact, used with or without pattern variables, partial, or fuzzy, and any matching algorithm will be based on such factors as

Choice of representation scheme for the objects being matched,

Criteria for matching (exact, partial, fuzzy, and so on),

Choice of measure required to perform the match in accordance with the chosen criteria, and

Type of match description required for output.

In the remainder of this chapter we examine various types of matching problems and their related algorithms. We bin with a description of representation structures and measures commonly found in matching problems. We next look at various matching techniques based on exact, partial, and fuzzy approaches. We conclude the chapter with an example of an efficient match algorithm used in some rule-based expert systems.

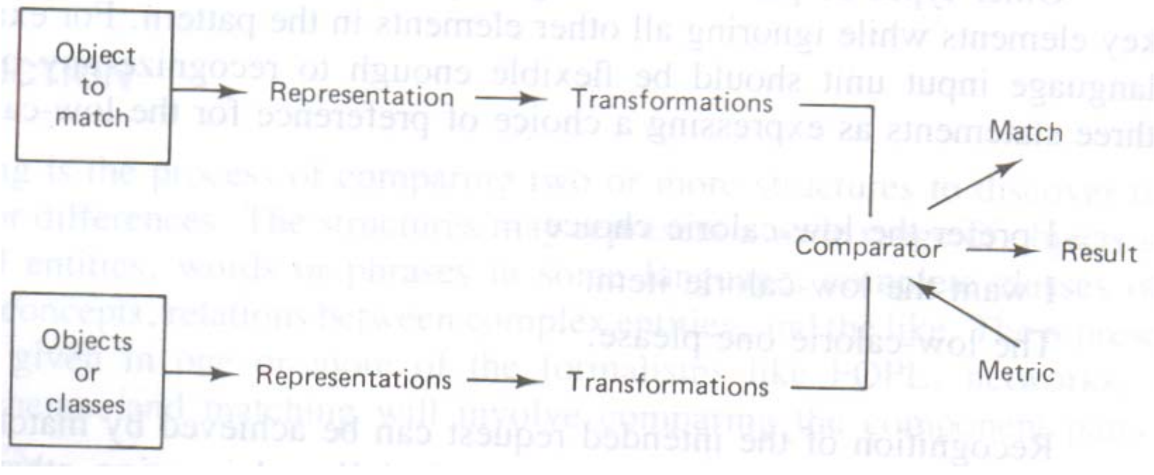
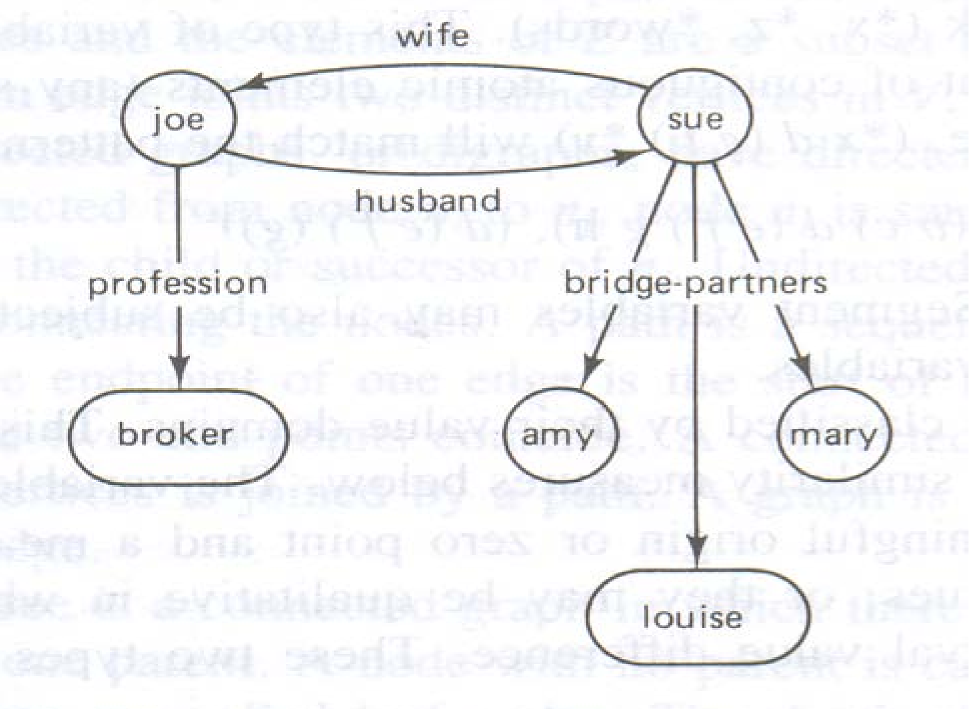


Fig Typical Matching Process

Structures used in Matching

- The types of list structures represent clauses in propositional or predicate logic such as (or ~(MARRIED ?x ?y) ~(DAUGHTER ?z ?y) (MOTHER ?y ?z)) or rules such as (and ((cloudy-sky) (low-bar-pressure) (high-humidity)) (conclude (rain likely)) or fragments of associative networks in below Fig
- The other common structures include strings of characters a_1 a_2 . . . a_k , where the a_i belong to given alphabet A, vector X = (x_1 x_2 . . . x_n), where the x_i represents attribute values, matrices M (rows of vectors), general graphs, trees and sets



```
(putprop 'joe '(sue) 'wife)
(putprop 'sue '(joe) 'husband)
(putprop 'joe '(broker) 'profession)
(putprop 'sue '(amy louise mary) 'bridge-partners)
```

Fig Fragment of associative network and corresponding LISP Code

- Variables
  - The structures are constructed from basic atomic elements, numbers and characters

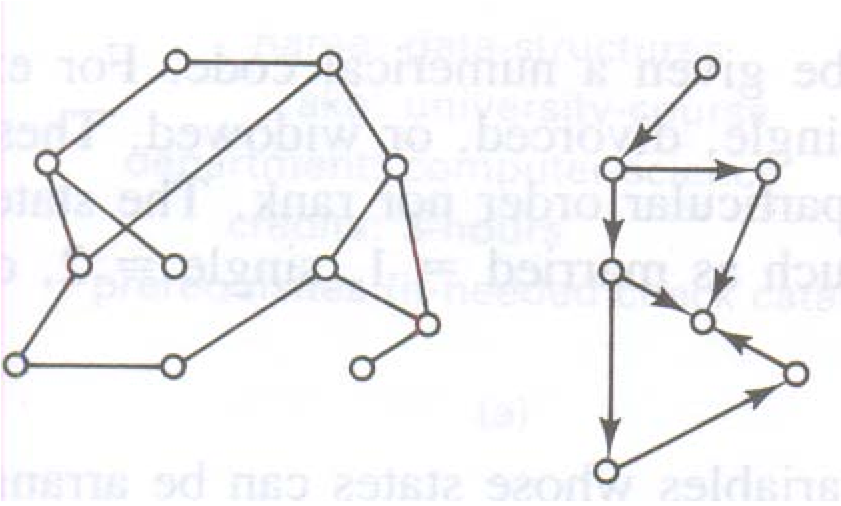
- Character string elements may represent either constants or variables
- If variables, they may be classified by either the type of match permitted or their value domains
- An open variable can be replaced by a single item
- Segment variable can be replaced by zero or more items
- Open variable are replaced with a preceding question mark (?x. ?y, ?class)
- They may match or assume the value of any single string element or word, but they are subject to consistency constraints
- For example, to be consistent, the variable ?x can be bound only to the same top level element in any single structure
- Thus (a ?x d ?x e) may match (a b d b e), but not (a b d a e)
- Segment variable types will be preceded with an asterisk (*x, *z, *words)
- This type of variable can match an arbitrary number or segment of contiguous atomic elements
- For example, (* x d (e g) *y) will match the patterns

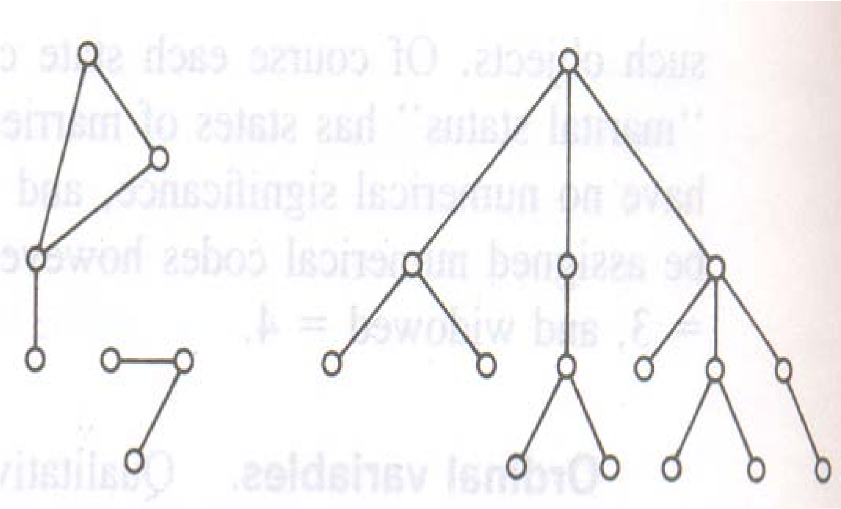
  (a (b c) d (e f) g h), (d (e f) (g))

- Subject variable may also be subject to consistency constraints similar to open variables

- Nominal variables
  - Qualitative variables whose values or states have no order nor rank
  - It is possible to distinguish between equality or inequality between two objects
  - Each state can be given a numerical code
  - For example, “marital status” has states of married, single, divorced or widowed. These states could be assigned numerical codes, such as married = 1, single = 2, divorced = 3 and widowed = 4
- Ordinal variables
  - Qualitative variables whose states can be arranged in a rank order
  - It may be assigned numerical values
  - Foe example, the states very tall, tall, medium, short and very short can be arranged in order from tallest to shortest and can be assigned an arbitrary scale of 5 to 1
- Binary variable

  - Qualitative discrete variables which may assume only one of two values, such as 0 or 1, good or bad, yes or no, high or low
- Interval variables or Metric variables
  - Qualitative variables which take on numeric values and for which equal differences between values have the same significance
  - For example, real numbers corresponding to temperature or integers corresponding to an amount of money are considered as interval variables
- Graphs and Trees
  - A graph is a collection of points called vertices, some of which are connected by line segments called edges
  - Graphs are used to model a wide variety of real-life applications, including transportation and communication networks, project scheduling, and games
  - A graph G = (V, E) is an ordered pair of sets V and E. the elements V are nodes or vertices and the elements of E are a subset of V X V called edges
  - An edge joints two distinct vertices in V
  - Directed graphs or digraphs, have directed edges or arcs with arrows
  - If an arc is directed from node n_i to n_j , node n_i is said to be a parent or successor of n_j and n_j is the child or successor of n_i
  - Undirected graphs have simple edges without arrows connecting the nodes
  - A path is a sequence of edges connecting two modes where the endpoint of one edge is the start of its successor
  - A cycle is a path in which the two end points coincide
  - A Connected graph is a graph for which every pair of vertices is joined by a path
  - A graph is complete if every element of V X V is an edge
  - A tree is a connected graph in which there are no cycles and each node has at most one parent below
  - A node with no parent is called root node
  - A node with no children is called leaf node
  - The depth of the root node is defined as zero
  - The depth of any other node is defined to be the depth of its parent plus 1





- Sets and Bags
  - A set is represented as an unordered list of unique elements such as the set (a d f c) or (red blue green yellow)
  - A bag is a set which may contain more than one copy of the same member a, b, d and e
  - Sets and bags are structures used in matching operations

Measure for Matching

- The problem of comparing structures without the use of pattern matching variables. This requires consideration of measures used to determine the likeness or similarity between two or more structures

- The similarity between two structures is a measure of the degree of association or likeness between the object's attributes and other characteristics parts.
- If the describing variables are quantitative, a distance metric is used to measure the proximity
- Distance Metrics
  - For all elements x, y, z of the set E, the function d is metric if and only if

    $$d(x, x) = 0$$

    $$d(x,y) \geq 0$$

    $$d(x,y) = d(y,x)$$

    $$d(x,y) \leq d(x,z) + d(z,y)$$

  - The Minkowski metric is a general distance measure satisfying the above assumptions
  - It is given by

    $$d_p = [\sum_{i=1}^{n} |x_i - y_i|^p]^{1/p}$$

  - For the case p = 2, this metric is the familiar Euclidian distance. Where p = 1, d_p is the so-called absolute or city block distance
- Probabilistic measures
  - The representation variables should be treated as random variables
  - Then one requires a measure of the distance between the variates, their distributions, or between a variable and distribution
  - One such measure is the Mahalanobis distance which gives a measure of the separation between two distributions
  - Given the random vectors X and Y let C be their covariance matrix
  - Then the Mahalanobis distance is given by

    $$D = X' C^{-1} Y$$

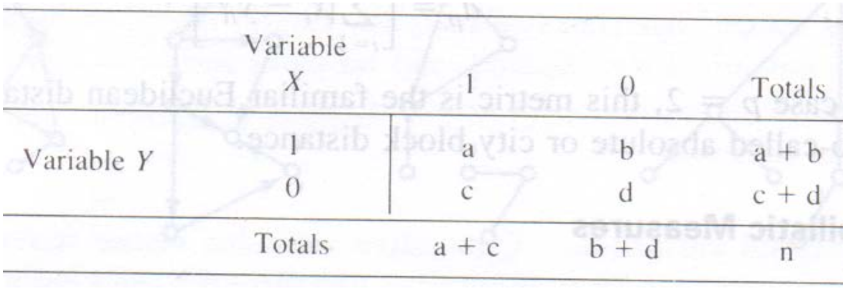
  - Where the prime (') denotes transpose (row vector) and C^{-1} is the inverse of C
  - The X and Y vectors may be adjusted for zero means bt first substracting the vector means u_x and u_y

  - Another popular probability measure is the product moment correlation r, given by

$$r = \frac{\mathrm{Cov}(X, Y)}{[\ \mathrm{Var}(X)^{*}\ \mathrm{Var}(Y)]^{1/2}}$$

  - Where Cov and Var denote covariance and variance respectively
  - The correlation r, which ranges between -1 and +1, is a measure of similarity frequently used in vision applications
  - Other probabilistic measures used in AI applications are based on the scatter of attribute values
  - These measures are related to the degree of clustering among the objects
  - Conditional probabilities are sometimes used
  - For example, they may be used to measure the likelihood that a given X is a member of class C_i , $P(C_i| X)$, the conditional probability of C_i given an observed X
  - These measures can establish the proximity of two or more objects

- Qualitative measures
  - Measures between binary variables are best described using contingency tables in the below
  - Table



	Variable *X*	1	0	Totals
Variable *Y*	1	a	b	a + b
	0	c	d	c + d
	Totals	a + c	b + d	n

  - The table entries there give the number of objects having attribute X or Y with corresponding value of 1 or 0
  - For example, if the objects are animals might be horned and Y might be long tailed. In this case, the entry a is the number of animals having both horns and long tails

  - Note that n = a + b + c + d, the total number of objects
  - Various measures of association for such binary variables have been defined
  - For example

$$\frac{a}{a + b + c + d} = \frac{a}{n}, \qquad \frac{a + d}{n}$$

$$\frac{a}{a + b + c}, \qquad \frac{a}{b + c}$$

  - Contingency tables are useful for describing other qualitative variables, both ordinal and nominal. Since the methods are similar to those for binary variables
  - Whatever the variable types used in a measure, they should all be properly scaled to prevent variables having large values from negating the effects of smaller valued variables
  - This could happen when one variable is scaled in millimeters and another variable in meters

- Similarity measures
  - Measures of dissimilarity like distance, should decrease as objects become more alike
  - The similarities are not in general symmetric
  - Any similarity measure between a subject description A and its referent B, denoted by s(A,B), is not necessarily equal
  - In general, s(A,B) ≠ s(B,A) or “A is like B” may not be the same as “B is like A”
  - Tests on subjects have shown that in similarity comparisons, the focus of attention is on the subject and, therefore, subject features are given higher weights than the referent

- For example, in tests comparing countries, statements like “North Korea is similar to Red China” and “Red China is similar to North Korea” were not rated as symmetrical or equal
- Similarities may depend strongly on the context in which the comparisons are made
- An interesting family of similarity measures which takes into account such factors as asymmetry and has some intuitive appeal has recently been proposed
- Let O ={o_1, o_2, . . . } be the universe of objects of interest
- Let A_i be the set of attributes used to represent o_i
- A similarity measure s which is a function of three disjoint sets of attributes common to any two objects A_i and A_j is given as

$$s(A_{i,} A_j) = F(A_i \& A_j, A_i - A_j, A_j - A_i)$$

- Where A_i & A_j is the set of features common to both o_i and o_j
- Where A_i - A_j is the set of features belonging to o_i and not o_j
- Where A_j - A_i is the set of features belonging to o_j and not o_i
- The function F is a real valued nonnegative function

$$s(A_{i,} A_j) = af(A_i \& A_j) – bf(A_i - A_j) – cf(A_j - A_i) \text{ for some } a,b,c \geq 0$$

- Where f is an additive interval metric function
- The function f(A) may be chosen as any nonnegative function of the set A, like the number of attributes in A or the average distance between points in A

$$S(A_{i,} A2_j) = \frac{f(A_i \& A_j)}{f(A_i \& A_j) + af(A_i - A_j) + bf(A_j - A_i)}$$

for some $a,b \geq 0$

- When the representations are graph structures, a similarity measure based on the cost of transforming one graph into the other may be used
- For example, a procedure to find a measure of similarity between two labeled graphs decomposes the graphs into basic subgraphs and computes the minimum cost to transform either graph into the other one, subpart-by-subpart

Matching like Patterns

- We consider procedures which amount to performing a complete match between two structures
- The match will be accomplished by comparing the two structures and testing for equality among the corresponding parts
- Pattern variables will be used for instantiations of some parts subject to restrictions
- Matching Substrings
  - ➢ A basic function required in many match algorithms is to determine if a substring S_2 consisting of m characters occurs somewhere in a string S_1 of n characters, $m \leq n$
  - ➢ A direct approach to this problem is to compare the two strings character-by-character, starting with the first characters of both S_1 and S_2
  - ➢ If any two characters disagree, the process is repeated, starting with the second character of S_1 and matching again against S_2 character-by-character until a match is found or disagreements occurs again
  - ➢ This process continues until a match occurs or S_1 has no more characters
  - ➢ Let i and j be position indices for string S1 and a k position index for S_2
  - ➢ We can perform the substring match with the following algorithm

```
i := 0
While i ≤ (n – m + 1) do
begin
    i := i + 1;
    j := i;
    k :=1;
while S1(j) = S2(k) do
begin
    if k = m
        writeln("success")
    else do
```

```
                begin
                        j := j +1;
                        k := k + 1;
                end
        end
end
writeln(“fail”)
end
```

- This algorithm requires m(n - m) comparisons in the worst case
- A more efficient algorithm will not repeat the same comparisons over and over again
- One such algorithm uses two indices I and j, where I indexes the character positions in S_1 and j is set to a “match state” value ranging from 0 to m
- The state 0 corresponds to no matched characters between the strings, while state 1 corresponds to the first letter in S_2 matching character i in S_2
- State 2 corresponds to the first two consecutive letters in S_2 matching letters i and i+1 in S_1 respectively, and so on, with state m corresponding to a successful match
- Whenever consecutive letters fail to match, the state index is reduced accordingly

- Matching Graphs
  - Two graphs G_1 and G_2 match if they have the same labeled nodes and same labeled arcs and all node-to-node arcs are the same
  - If G_2 with m nodes is a subgraph of G_1 with n nodes, where $n \geq m$
  - In a worst cas match, this will require n!/(n - m)! node comparison and $O(m^2)$ arc comparisons
  - Finding subgraph isomorphisms is also an important matching problem
  - An isomorphism between the graphs G_1 and G_2 with vertices V_1, V_2 and edges E1, E2, that is, (V1, E1) and (V2, E2) respectively, is a one-to-one mapping to f between V1 and V2, such that for all v1 € V1, f(v1) = v2, and for each arc e1 € E1 connecting v1 and $v1'$, there is a corresponding arc e2 € E2 connecting f(v1) and $f(v1')$

- Matching Sets and Bags
  - An exact match of two sets having the same number of elements requires that their intersection also have the number of elements
  - Partial matches of two sets can also be determined by taking their intersections
  - If the two sets have the same number of elements and all elements are of equal importance, the degree of match can be the proportion of the total members which match
  - If the number of elements differ between the sets, the proportion of matched elements to the minimum of the total number of members can be used as a measure of likeness
  - When the elements are not of equal importance, weighting factors can be used to score the matched elements
  - For example, a measure such as

    $$s(S1,S2) \quad = \quad (\sum w_i N(a_i))/m$$

    could be used, where w_i =1 and $N(a_i)$ = 1 if a_i is in the intersection; otherwise it is 0
  - An efficient way to find the intersection of two sets of symbolic elements in LISP is to work through one set marking each element on the elements property list and then saving all elements from the other list that have been marked
  - The resultant list of saved elements is the required intersection
  - Matching two bags is similar to matching two sets except that counts of the number of occurrences of each element must also be made
  - For this, a count of the number of occurrences can be used as the property mark for elements found in one set. This count can then be used to compare against a count of elements found in the second set
- Matching to Unify Literals
  - One of the best examples of nontrivial pattern matching is in the unification of two FOPL literals
  - For example, to unify P(f(a,x), y ,y) and P(x, b, z) we first rename variables so that the two predicates have no variables in common
  - This can be done by replacing the x in the second predicate with u to give P(u, b, z)

- Compare the two symbol-by-symbol from left to right until a disagreement is found
- Disagreements can be between two different variables, a nonvariable term and a variable, or two nonvariable terms. If no disagreements is found, the two are identical and we have succeeded
- If disagreements is found and both are nonvariable terms, unification is impossible; so we have failed
- If both are variables, one is replaced throughout by the other. Finally, if disagreement is a variable and a nonvariable term, the variable is replaced by the entire term
- In this last step, replacement is possible only if the term does not contain the variable that is being replaced. This matching process is repeated until the two are unified or until a failure occurs
- For the two predicates P, above, a disagreement is first found between the term f(a,x) and variable u. Since f(a,x) does not contain the variable u, we replace u with f(a,x) everywhere it occurs in the literal
- This gives a substitution set of {f(a,x)/u} and the partially matched predicates P(f(a,x),y ,y) and P(f(a,x), b, z)
- Proceeding with the match, we find the next disagreements pair y and b, a variable and term, respectively
- Again we replace the variable y with the terms b and update the substitution list to get {f(a,x)/u, b/y}
- The final disagreements pair is two variables. Replacing the variable in the second literal with the first we get the substitution set {f(a,x)/u, b/y, y/z}or {f(a,x), b/y, b/z}
- For example, a LISP program which uses both the open and the segment pattern matching variables to find a match between a pattern and a clause

```
(defun match (pattern clause)

  (cond ((equal pattern clause) t )              ; return t if

      ((or (null pattern) (null clause)) nil)     ; equal, nil

                                                  ; if not

      ((or (equal (car pattern) (car clause))     ;not, ?x
                                                        ; binds
                (equal (car pattern) ‘?x))              ; to single
              (match (cdr pattern) (cdr clause)))      ; term, *y
                                                        ; binds
        ((equal (car pattern) ‘*y)                      ; to several
        (or (match (cdr pattern) (cdr clause))          ;contiguous
        (match pattern (cdr clause))))))                ; terms
```

- ➢ When a segment variable is encountered (the *y), match is recursively executed on the cdrs of both pattern and clause or on the cdr of clause and pattern as *y matches one or more than one item respectively

Partial Matching

- For many AI applications complete matching between two or more structures is inappropriate
- For example, input representations of speech waveforms or visual scenes may have been corrupted by noise or other unwanted distortions.
- In such cases, we do not want to reject the input out of hand. Our systems should be more tolerant of such problems
- We want our system to be able to find an acceptable or best match between the input and some reference description
- Compensating for Distortions
    - ➢ Finding an object in a photograph given only a general description of the object is a common problems in vision applications
    - ➢ For example, the task may be to locate a human face or human body in photographs without the necessity of storing hundreds of specific face templates
    - ➢ A better approach in this case would be to store a single reference description of the object

- Matching between photographs regions and corresponding descriptions then could be approached using either a measure of correlation, by altering the image to obtain a closer fit
- If nothing is known about the noise and distortion characteristics, correlation methods can be ineffective or even misleading. In such cases, methods based on mechanical distortion may be appropriate
- For example, our reference image is on a transparent rubber sheet. This sheet is moved over the input image and at each location is stretched to get the best match alignment between the two images
- The match between the two can then be evaluated by how well they correspond and how much push-and-pull distortion is needed to obtain the best correspondence
- Use the number of rigid pieces connected with springs. This pieces can correspond to low level areas such as pixels or even larger area segments

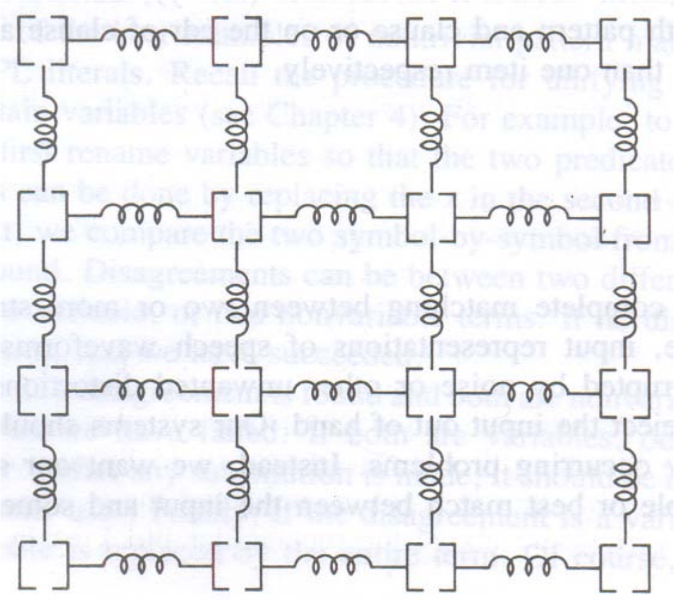


Fig Discrete version of stretchable overlay image

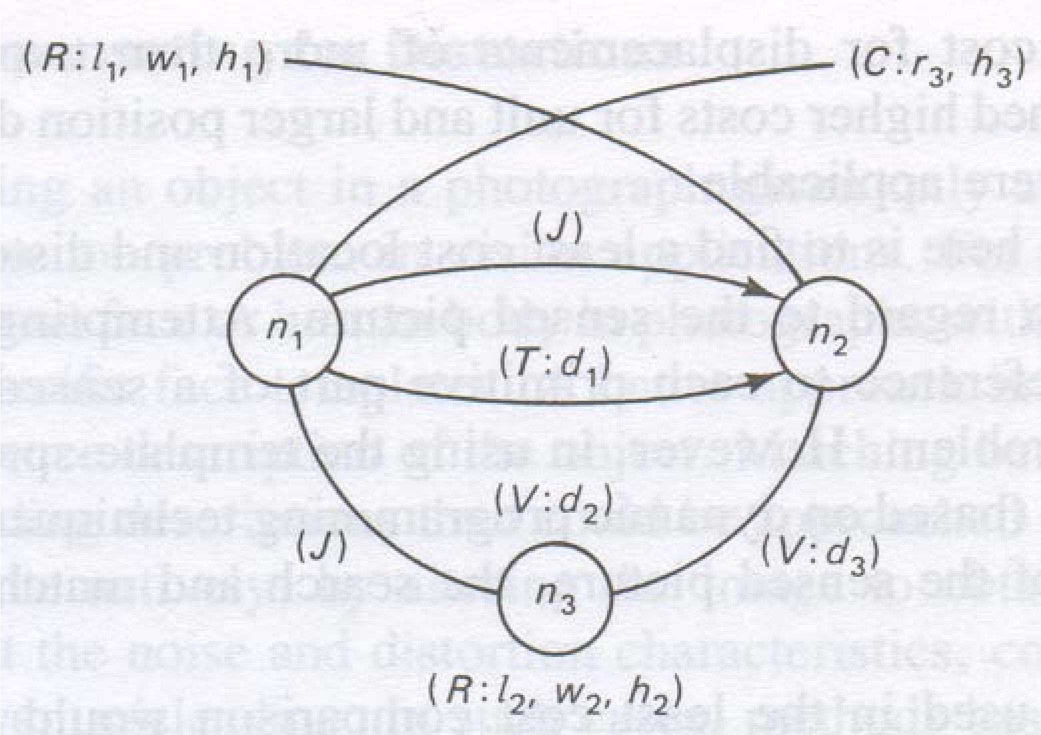
- To model any restrictions such as the relative positions of body parts, nonlinear cost functions of piece displacements can be used
- The costs can correspond to different spring tensions which reflect the constraints
- For example, the cost of displacing some pieces might be zero for no displacement, one unit for single increment displacements in any one of the

permissible directions, two units for two position displacements and infinite cost for displacements of more than two increments. Other pieces would be assigned higher costs for unit and larger position displacements when stronger constraints were applicable

  - The matching problem is to find a least cost location and distortion pattern for the reference sheet with regard to the sensed picture
  - Attempting to compare each component of some reference to each primitive part of a sensed picture is a combinatorially explosive problem
  - In using the template-spring reference image and heuristic methods to compare against different segments of the sensed picture, the search and match process can be made tractable
  - Any matching metric used in the least cost comparison would need to take into account the sum of the distortion costs C_d , the sum of the costs for reference and sensed component dissimilarities Cc , and the sum of penalty costs for missing components C_m . Thus, the total cost is given by

$$C_t = C_d + C_c + C_m$$

- Finding Match Differences
  - Distortions occurring in representations are not the only reason for partial matches
  - For example, in problem solving or analogical inference, differences are expected. In such cases the two structures are matched to isolate the differences in order that they may be reduced or transformed. Once again, partial matching techniques are appropriate
  - In visual application, an industrial part may be described using a graph structure where the set of nodes correspond to rectangular or cylindrical block subparts
  - The arc in the graph correspond to positional relations between the subparts
  - Labels for rectangular block nodes contain length, width and height, while labels for cylindrical block nodes, where location can be above, to the right of, behind, inside and so on
  - In Fig 8.5 illustrates a segment of such a graph



- In the fig the following abbreviations are used
- Interpreting the graph, we see it is a unit consisting of subparts, mode up of rectangular and cylindrical blocks with dimensions specified by attribute values
- The cylindrical block n_1 is to the right of n_2 by d_1 units and the two are connected by a joint
- The blocks n_1 and n_2 are above the rectangular block n_3 by d_2 and d_3 units respectively, and so on
- Graphs such as this are called attributed relational graphs (ATRs). Such a graph G is defined as a sextuple G = (N, B, A, G_n, G_b)
- Where N = { n_1, n_2, . . ., n_k}is a set of nodes, A = {a n_1, an_2, . . . , an_k} is an alphabet of node attributes, B = { b_1, b_2, . . . , b_m}is a set of directed branches (b = (n_i, n_j)), and G_n and G_b are functions for generating node and branch attributes respectively
- When the representations are graph structures like ARGs, a similarity measure may be computed as the total cost of transforming one graph into the other
- For example, the similarity of two ARGs may be determined with the following steps:
    - Decompose the ARGs into basic subgraphs, each having a depth of one

  - Compute the minimum cost to transform either basic ARG into the other one subgraph-by-subgraph
  - Compute the total transformation cost from the sum of the subgraph costs
- An ARG may be transformed by the three basic operations of node or branch deletions, insertions, or substitutions, where each operation is given a cost based on computation time or other factors

The RETE matching algorithm

- One potential problem with expert systems is the number of comparisons that need to be made between rules and facts in the database.
- In some cases, where there are hundreds or even thousands of rules, running comparisons against each rule can be impractical.
- The Rete Algorithm is an efficient method for solving this problem and is used by a number of expert system tools, including OPS5 and Eclipse.
- The Rete is a directed, acyclic, rooted graph.
- Each path from the root node to a leaf in the tree represents the left-hand side of a rule.
- Each node stores details of which facts have been matched by the rules at that point in the path. As facts are changed, the new facts are propagated through the Rete from the root node to the leaves, changing the information stored at nodes appropriately.
- This could mean adding a new fact, or changing information about an old fact, or deleting an old fact. In this way, the system only needs to test each new fact against the rules, and only against those rules to which the new fact is relevant, instead of checking each fact against each rule.
- The Rete algorithm depends on the principle that in general, when using forward chaining in expert systems, the values of objects change relatively infrequently, meaning that relatively few changes need to be made to the Rete.
- In such cases, the Rete algorithm can provide a significant improvement in performance over other methods, although it is less efficient in cases where objects are continually changing.
- The basic inference cycle of a production system is match, select and execute as indicated in Fig 8.6. These operations are performed as follows

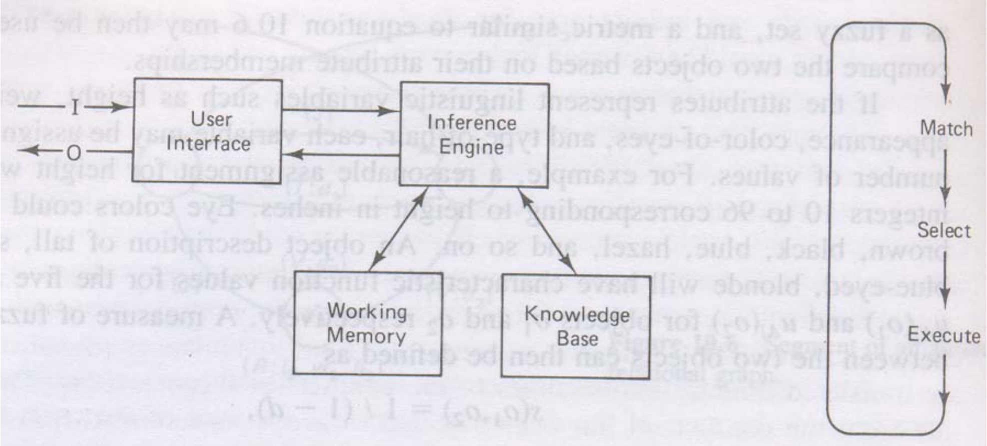


Fig Production system components and basic cycle

- Match
  - During the match portion of the cycle, the conditions in the LHS of the rules in the knowledge base are matched against the contents of working memory to determine which rules have their LHS conditions satisfied with consistent bindings to working memory terms.
  - Rules which are found to be applicable are put in a conflict set
- Select
  - From the conflict set, one of the rules is selected to execute. The selection strategy may depend on recency of useage, specificity of the rule or other criteria
- Execute
  - The rule selected from the conflict set is executed by carrying the action or conclusion part of the rule, the RHS of the rule. This may involve an I/O operation, adding, removing or changing clauses in working memory or simply causing a halt
  - The above cycle is repeated until no rules are put in the conflict set or until a stopping condition is reached
  - The main time saving features of RETE are as follows
    1. in most expert systems, the contents of working memory change very little from cycle to cycle. There is persistence in the data known as temporal redundancy. This makes exhaustive matching on every cycle unnecessary. Instead, by saving match information, it is only necessary to compare working memory changes on each cycle. In RETE,

addition to, removal from, and changes to working memory are translated directly into changes to the conflict set in Fig . Then when a rule from the conflict set has been selected to fire, it is removed from the set and the remaining entries are saved for the next cycle. Consequently, repetitive matching of all rules against working memory is avoided. Furthermore, by indexing rules with the condition terms appearing in their LHS, only those rules which could match. Working memory changes need to be examined. This greatly reduces the number of comparisons required on each cycle

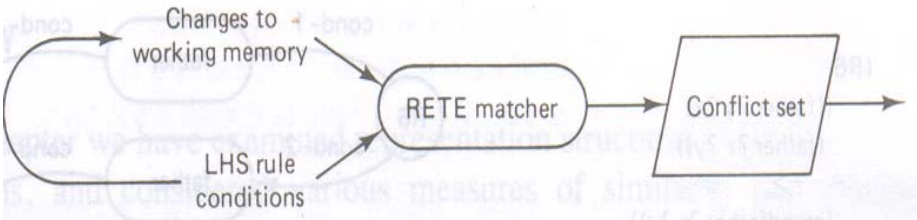


Fig Changes to working memory are mapped to the conflict set

2. Many rules in a knowledge base will have the same conditions occurring in their LHS. This is just another way in which unnecessary matching can arise. Repeating testing of the same conditions in those rules could be avoided by grouping rules which share the same conditions and linking them to their common terms. It would then be possible to perform a single set of tests for all the applicable rules shown in Fig below

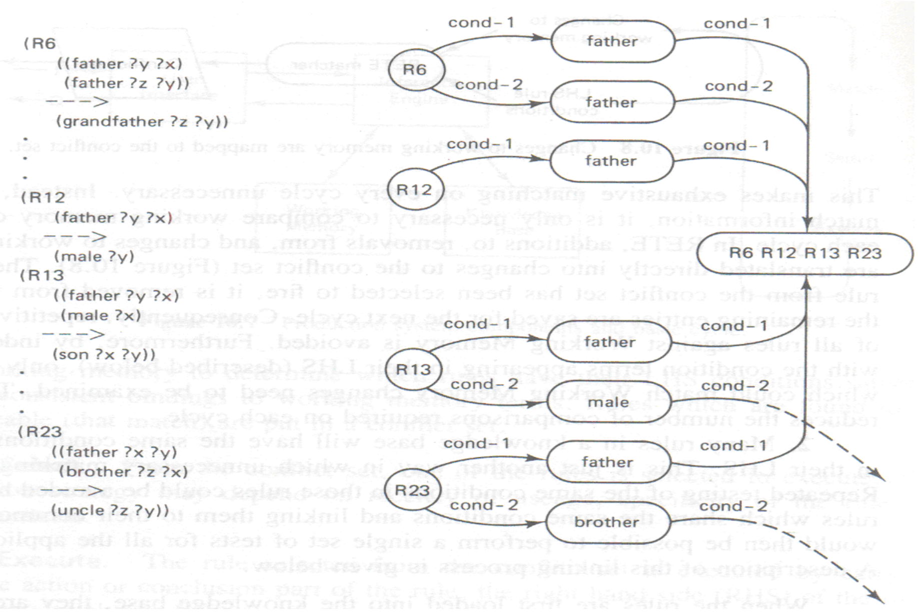


Fig Typical rules and a portion of a compiled network

Knowledge Organization and Management

The advantage of using structured knowledge representation schemes (frames, associative networks, or object-oriented structures) over unstructured ones (rules or FOPL clauses) should be understood and appreciated at this point. Structured schemes group or link small related chunks of knowledge together as a unit. This simplifies the processing operations, since knowledge required for a given task is usually contained within a limited semantic region, which can be accessed as a unit or traced through a few linkages.

But, as suggested earlier, representation is not the only factor, which affects efficient manipulation. A program must first locate and retrieve the appropriate knowledge in an efficient manner whenever it is needed. One of the most direct methods for finding the appropriate knowledge is exhaustive search or the enumerations of all items in memory. This is also one of the least efficient access methods. More efficient retrieval is accomplished through some form of indexing or grouping. We consider some of these processes in the next section where we review traditional access and retrieval methods used in memory organizations. This is followed by a description of less commonly used forms of indexing.

A “smart” expert system can be expected to have thousands or even tens of thousands of rules (or their equivalent) in its KB. A good example is XCON (or RI), an expert system which was developed for the Digital Equipment Corporation to configure their customer’s computer systems. XCON has a rapidly growing KB, which, at the present time, consists of more than 12,000 production rules. Large numbers of rules are needed in systems like this, which deal with complex reasoning tasks. System configuration becomes very complex when the number of components and corresponding parameters is large (several hundred). If each rule contained above four or five conditions in its antecedent or If part and an exhaustive search was used, as many as 40,000-50,000 tests could be required on each recognition cycle. Clearly, the time required to perform this number of tests is intolerable. Instead, some form of memory management is needed. We saw one way this problem was solved using a form of indexing with the RETE algorithm described in the preceding chapter, More direct memory organization approaches to this problem are considered in this chapter.

We humans live in a dynamic, continually changing environment. To cope with this change, our memories exhibit some rather remarkable properties. We are able to adapt to varied changes in the environment and still improve our performance. This is because our memory system is continuously adapting through a reorganization process. New knowledge is continually being added to our memories, existing knowledge is continually being revised, and less important knowledge is gradually being forgotten. Our memories are continually being reorganized to expand our recall and reasoning abilities. This process leads to improved memory performance throughout most of our lives.

When developing computer memories for intelligent systems, we may gain some useful insight by learning what we can from human memory systems. We would expect computer memory systems to possess some of the same features. For example, human memories tend to be limitless in capacity, and they provide a uniform grade of recall service, independent of the amount of information store. For later use, we have summarized these and other desirable characteristics that we feel an effective computer memory organization system should possess.

1. It should be possible to add and integrate new knowledge in memory as needed without concern for limitations in size.
2. Any organizational scheme chosen should facilitate the remembering process. Thus, it should be possible to locate any stored item of knowledge efficiently from its content alone.

3. The addition of more knowledge to memory should have no adverse effects on the accessibility of items already stored there. Thus, the search time should not increase appreciably with the amount of information stored.
4. The organization scheme should facilitate the recognition of similar items of knowledge. This is essential for reasoning and learning functions. It suggests that existing knowledge be used to determine the location and manner in which new knowledge is integrated into memory.
5. The organization should facilitate the process of consolidating recurrent incidents or episodes and "forgetting" knowledge when it is no longer valid or no longer needed.

These characteristics suggest that memory be organized around conceptual clusters of knowledge. Related clusters should be grouped and stored in close proximity to each other and be linked to similar concepts through associative relations. Access to any given cluster should be possible through either direct or indirect links such as concept pointers indexed by meaning. Index keys with synonomous meanings should provide links to the same knowledge clusters. These notions are illustrated graphically in Fig 9.1 where the clusters represent arbitrary groups closely related knowledge such as objects and their properties or basic conceptual categories. The links connecting the clusters are two-way pointers which provide relational associations between the clusters they connect.

Indexing and retrieval techniques

- **The Frame Problem**

  One tricky aspect of systems that must function in dynamic environments is due to the so called frame problem. This is the problem of knowing what changes have and have not taken place following some action. Some changes will be the direct result of the action. Other changes will be the result of secondary or side effects rather than the result of the action. Foe example, if a robot is cleaning the floors in a house, the location of the floor sweeper changes with the robot even though this is not explicitly stated. Other objects not attached to the robot remain in their original places. The actual changes must somehow be reflected in memory, a fear that requires some ability to infer. Effective memory organization and management methods must take into account effects caused by the frame problem

The three basic problems related to knowledge organization:

1. classifying and computing indices for input information presented to system
2. access and retrieval of knowledge from mrmory through the use of the computed indices
3. the reorganization of memory structures when necessary to accommodate additions, revisions and forgetting. These functions are depicted in Fig 9.1

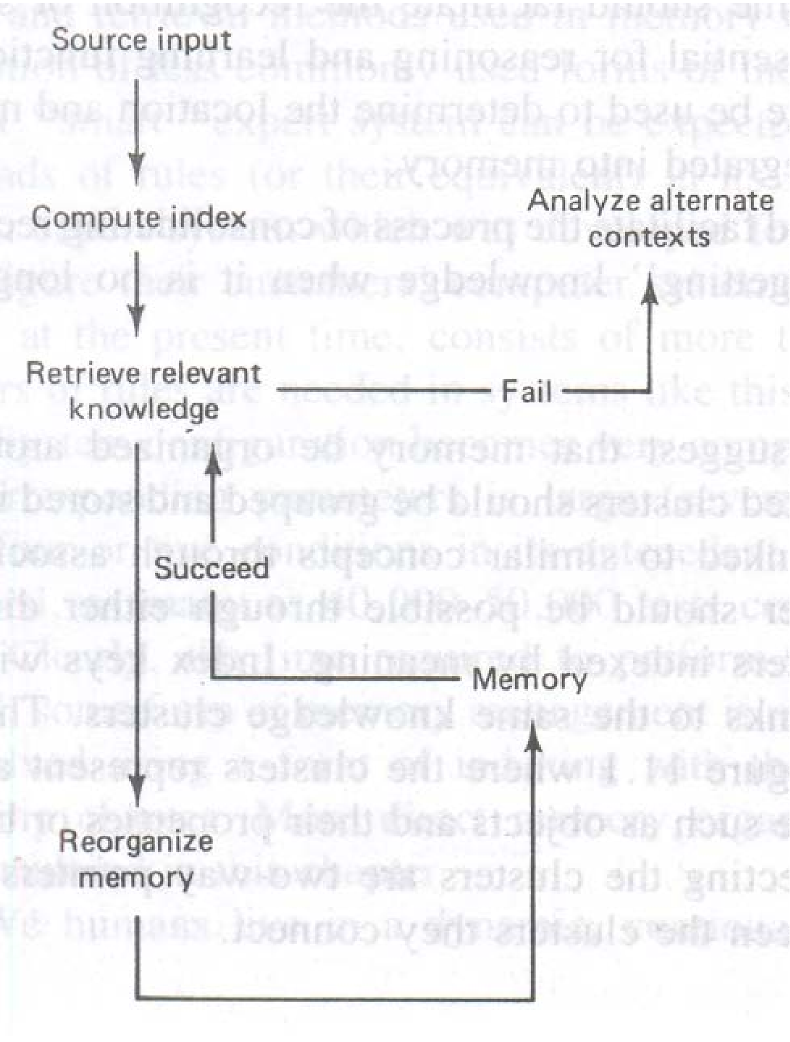


Fig Memory Organization Function

- When a knowledge base is too large to be held in main memory, it must be stored as a file in secondary storage (disk, drum or tape).
- Storage and retrieval of information in secondary memory is then performed through the transfer of equal-size physical blocks consisting of between 256 and 4096 bytes.
- When an item of information is retrieved or stored, at least one complete block must be transferred between main and secondary memory.
- The time required to transfer a block typically ranges between 10ms and 100ms, about the same amount of time required to sequentially searching the whole block for an item.
- Grouping related knowledge together as a unit can help to reduce the number of block transfers, hence the total access time
- An example of effective grouping can be found in some expert system KB organizations
- Grouping together rules which share some of the same conditions and conclusions can reduce block transfer times since such rules are likely to be needed during the same problem solving session
- Collecting rules together by similar conditions or content can help to reduce the number of block transfers required
- **Indexed Organization**
    - While organization by content can help to reduce block transfers, an indexed organization scheme can greatly reduce the time to determine the storage location of an item
    - Indexing is accomplished by organizing the information in some way for easy access
    - One way to index is by segregating knowledge into two or more groups and storing the locations of the knowledge for each group in a smaller index file
    - To build an indexed file, knowledge stored as units is first arranged sequentially by some key value
    - The key can be any chosen fields that uniquely identify the record
    - A second file containing indices for the record locations is created while the sequential knowledge file is being loaded
    - Each physical block in this main file results in one entry in the index file
    - The index file entries are pairs of record key values and block addresses

- The key value is the key of the first record stored in the corresponding block
- To retrieve an item of knowledge from the main file, the index file is searched to find the desired record key and obtain the corresponding block address
- The block is then accessed using this address. Items within the block are then searched sequentially for the desired record
- An indexed file contains a list of the entry pairs (k,b) where the values k are the keys of the first record in each block whose starting address is b
- Fig 9.2 illustrates the process used to locate a record using the key value of 378

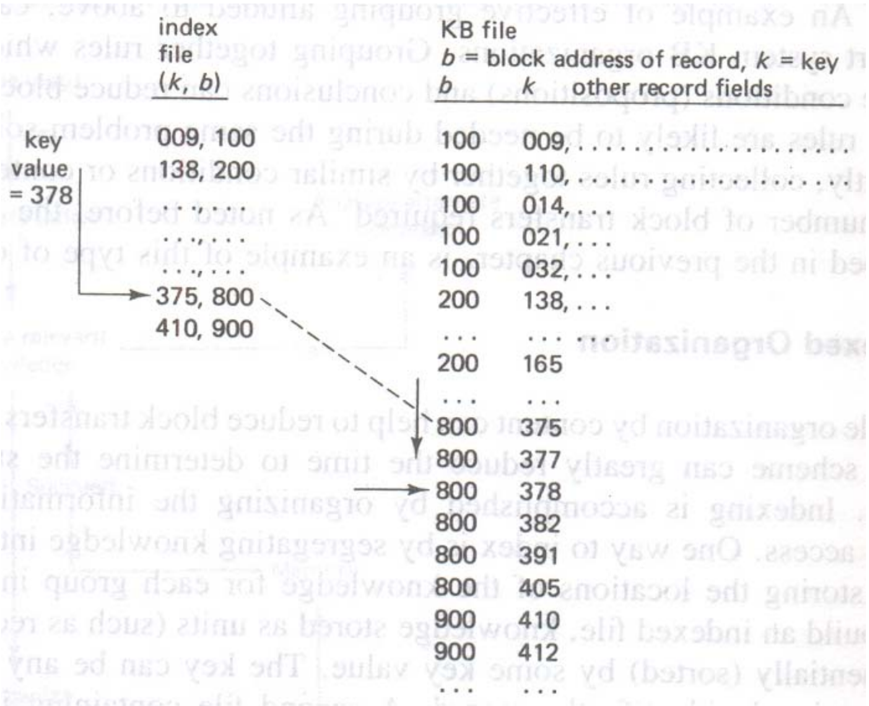


Fig Indexed File Organization

- The largest key value less than 378 (375) gives the block address (800) where the item will be found
- Once the 800 block has been retrieved, it can be searched linearly to locate the record with key value 378. this key could be any alphanumeric string that uniquely identifies a block, since such strings usually have a collation order defined by their code set
- If the index file is large, a binary search can be used to speed up the index file search
- A binary search will significantly reduce the search time over linear search when the number of items is not too small

- When a file contains n records, the average time for a linear search is proportional to n/2 compared to a binary search time on the order of $\ln_2(n)$
- Further reductions in search time can be realized using secondary or higher order arranged index files
- In this case the secondary index file would contain key and block address pairs for the primary index file
- Similar indexing would apply for higher order hierarchies where a separate file is used for each level
- Both binary search and hierarchical index file organization may be needed when the KB is a very large file
- Indexing in LISP can be implemented with property lists, A-lists, and/or hash tables. For example, a KB can be partitioned into segments by storing each segment as a list under the property value for that segment
- Each list indexed in this way can be found with the get property function and then searched sequentially or sorted and searched with binary search methods
- A hash-table is a special data structure in LISP which provides a means of rapid access through key hashing

- **Hashed Files**
  - Indexed organizations that permit efficient access are based on the use of a hash function
  - A hash function, h, transforms key values k into integer storage location indices through a simple computation
  - When a maximum number of items C are to stored, the hashed values h(k) will range from 0 to C – 1. Therefore, given any key value k, h(k) should map into one of 0…C – 1
  - An effective hash function can be computed by choosing the largest prime number p less than or equal to C, converting the key value k into an integer k' if necessary, and then using the value k' mod p as the index value h
  - For example, if C is 1000, the largest prime less than C is p = 997. thus, if the record key value is 123456789, the hashed value is h = (k mod 997) = 273
  - When using hashed access, the value of C should be chosen large enough to accommodate the maximum number of categories needed

- The use of the prime number p in the algorithm helps to insure that the resultant indices are somewhat uniformly distributed or hashed throughout the range 0 . . . C – 1
- This type of organization is well suited for groups of items corresponding to C different categories
- When two or more items belong to the same category, they will have the same hashed values. These values are called synonyms
- One way to accommodate collisions is with data structures known as buckets
- A bucket is a linked list of one or more items, where each item is record, block, list or other data strucyure
- The first item in each bucket has an address corresponding to the hashed address
- Fig 9.3 illustrates a form of hashed memory organization which uses buckets to hold all items with the same hashed key value

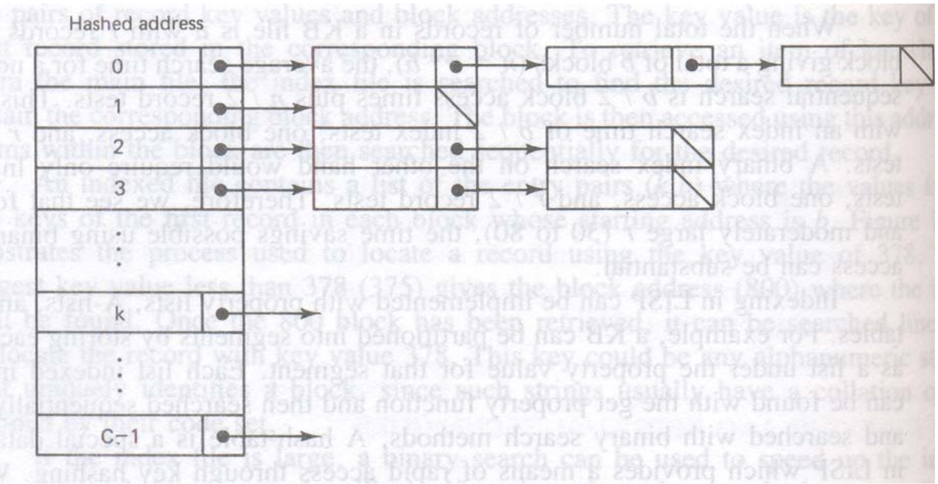


Fig Hashed Memory File organization

- The address of each bucket in this case is the indexed location in an array

- **Conceptual Indexing**
  - A better approach to indexed retrieval is one which makes use of the content or meaning associated with the stored entities rather than some nonmeaningful key value
  - This suggests the use of indices which name and define the entity being retrieved. Thus, if the entity is an object, its name and characteristic attributes would make meaningful indices

- If the entity is an abstract object such as a concept, the name and other defining traits would ne meaningful as indices
- Nodes within the network correspond to different knowledge entities, whereas the links are indices or pointers to the entities
- Links connecting two entities name the association or relationship between them
- The relationship between entities may be defined as a hierarchical one or just through associative links
- As an example of an indexed network, the concept of computer science CS should be accessible directly through the CS name or indirectly through associative links like a university major, a career field, or a type of classroom course
- These notions are illustrated in Fig 9.4

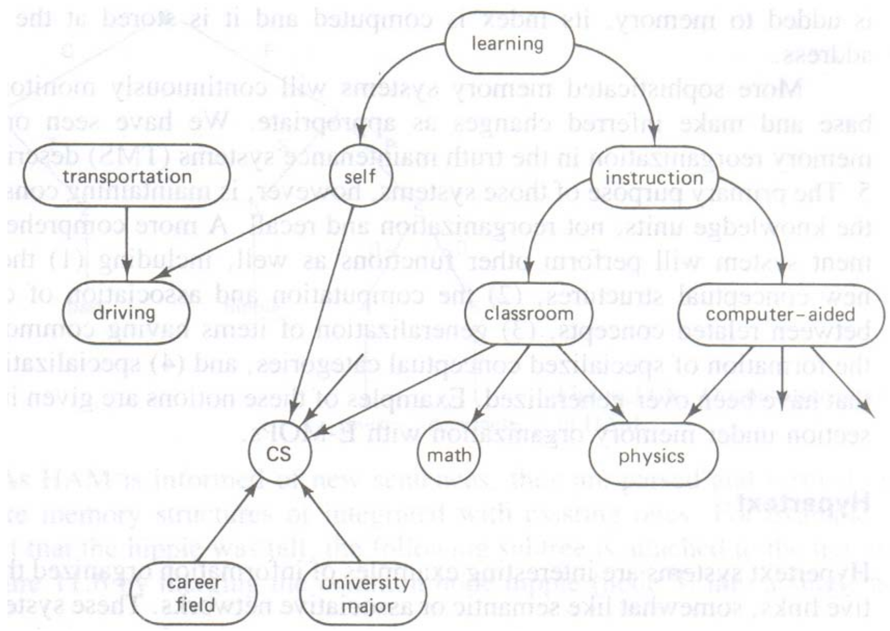


Fig Associative Network Indexing and Organization

- Object attributes can also serve as indices to locate items based on the attribute values
- In this case, the best attribute keys are those which provide the greatest discrimination among objects within the same category

  - For example, suppose we wish to organize knowledge by object types. In this case, the choice of attributes should depend on the use intended for the knowledge. Since objects may be classified with an unlimited number of attributes , those attributes which are most discriminable with respect to the concept meaning should be chosen

Integrating knowledge and memory

- Integrating new knowledge in traditional data bases is accomplished by simply adding an item to its key location, deleting an item from a key directed location, or modifying fields of an existing item with specific input information.
- When an item in inventory is replaced with a new one, its description is changed accordingly. When an item is added to memory, its index is computed and it is stored at the corresponding address
- More sophisticated memory systems will continuously monitor a knowledge base and make inferred changes as appropriate
- A more comprehensive management system will perform other functions as well, including the formation of new conceptual structures, the computation and association of casual linkages between related concepts, generalization of items having common features and the formation of specialized conceptual categories and specialization of concepts that have been over generalized
- **Hypertext**
  - Hypertext systems are examples of information organized through associative links, like associative networks
  - These systems are interactive window systems connected to a database through associative links
  - Unlike normal text which is read in linear fashion, hypertext can be browsed in a nonlinear way by moving through a network of information nodes which are linked bidirectionally through associative
  - Users of hypertext systems can wander through the database scanning text and graphics, creating new information nodes and linkages or modify existing ones
  - This approach to documentation use is said to more closely match the cognitive process

- It provides a new approach to information access and organization for authors, researchers and other users of large bodies of information

Memory organization system

- **HAM, a model of memory**
    - One of the earliest computer models of memory was the Human Associative memory (HAM) system developed by John Anderson and Gordon Bower
    - This memory is organized as a network of prepositional binary trees
    - An example of a simple tree which represents the statement “In a park s hippie touched a debutante” is illustrated in Fig 9.5
    - When an informant asserts this statement to HAM, the system parses the sentence and builds a binary tree representation
    - Node in the tree are assigned unique numbers, while links are labeled with the following functions:

        C: context for tree fact

        e: set membership

        F: a fact

        L: a location

        O: object

        P: predicate

        R: relation

        S: subject

        T: time

    - As HAM is informed of new sentences, they are parsed and formed into new tree-like memory structures or integrated with existing ones
    - For example, to add the fact that the hippie was tall, the following subtree is attached to the tree structure of Fig below by merging the common node hippie (node 3) into a single node

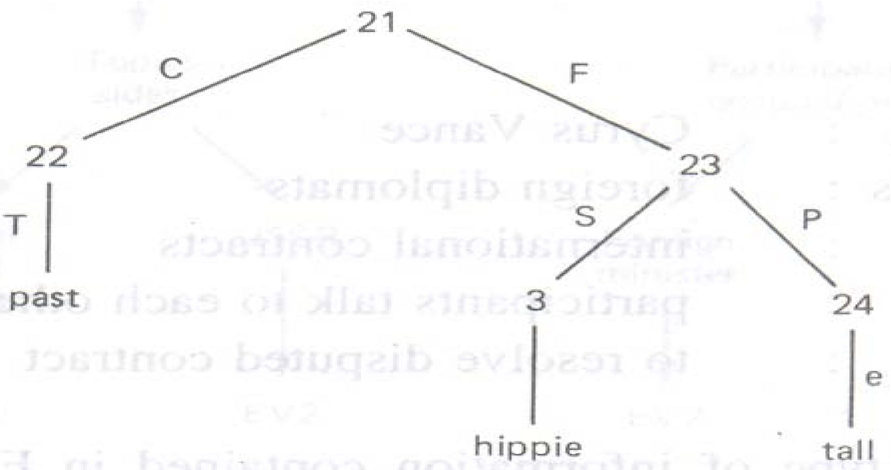


Fig Organization of knowledge in HAM

- When HAM is posed with a query, it is formed into a tree structure called a probe. This structure is then matched against existing memory structures for the best match
- The structure with the closest match is used to formulate an answer to the query
- Matching is accomplished by first locating the leaf nodes in memory that match leaf nodes in the probe
- The corresponding links are then checked to see if they have the same labels and in the same order
- The search process is constrained by searching only node groups that have the same relation links, based on recency of usage
- The search is not exhaustive and nodes accessed infrequently may be forgotten
- Access to nodes in HAM is accomplished through word indexing in LISP

- **Memory Organization with E-MOPs**
  - One system was developed by Janet Kolodner to study problems associated with the retrieval and organization of reconstructive memory, called CYRUS (Computerized Yale Retrieval and Updating System) stores episodes from the lives of former secretaries of state Cyrus Vance and Edmund Muskie
  - The episodes are indexed and stored in long term memory for subsequent use in answering queries posed in English

- The basic memory model in CYRUS is a network consisting of Episodic Memory Organization Packets (E-MOPs)
- Each such E-MOP is a frame-like node structure which contains conceptual information related to different categories of episodic events
- E-MOP are indexed in memory by one or more distinguishing features. For example, there are basic E-MOPs for diplomatic meetings with foreign dignitaries, specialized political conferences, traveling, state dinners as well as other basic events related to diplomatic state functions
- This diplomatic meeting E-MOP called $MEET, contains information which is common to all diplomatic meeting events
- The common information which characterizes such as E-MOP is called its content
- For example, $MEET might contain the following information:
- A second type of information contained in E-MOPs are the indices which index either individual episodes or other E-MOPs which have become specializations of their parent E-MOPs
- A typical $MEET E-MOP which has indices to two particular event meetings EV1 and EV2, is illustrated in Fig 9.6

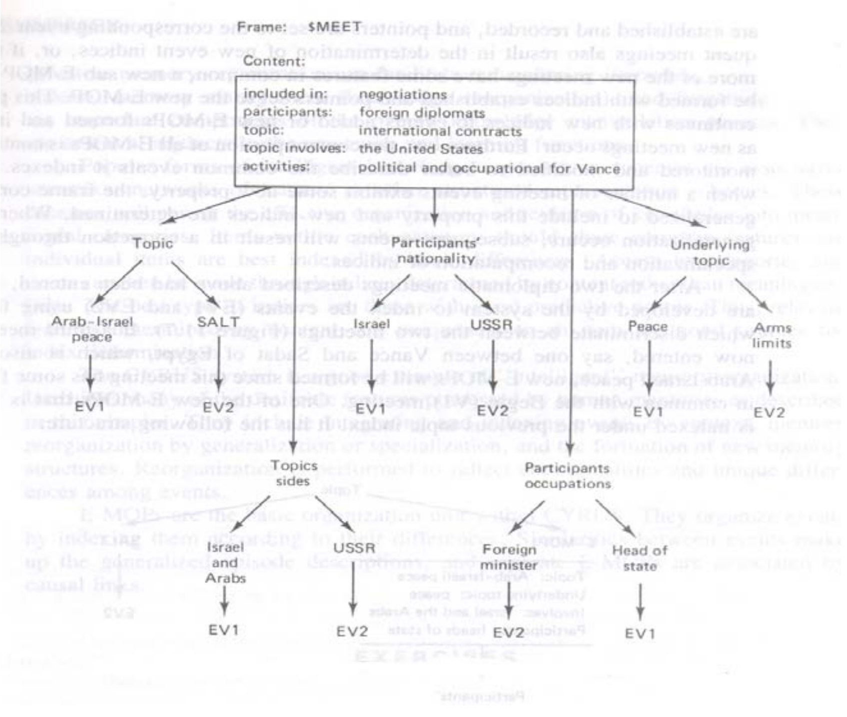


Fig An example of an EMOP with two indexed events EV1 and EV2

- For example, one of the meetings indexed was between Vance and Gromyko of the USSR in which they discussed SALT. This is labeled as event EV1 in the figure. The second meeting was between Vance and Begin of Israel in which they discussed Arab-Israeli peace. This is labeled as event EV2
- Note that each of these events can be accessed through more than one feature (index). For example, EV1 can be located from the $MEET event through a topic value of “Arab-Israel peace,” through a participants’ nationality value of “Israel,” through a participants’ occupation value of “head of state,” and so on
- As new diplomatic meetings are entered into the system, they are either integrated with the $MEET E-MOP as a separately indexed event or merged with another event to form a new specialized meeting E-MOP.
- When several events belonging to the same MOP category are entered, common event features are used to generalize the E-MOP. This information is collected in the frame contents. Specialization may also be required when over-generalization has occurred. Thus, memory is continually being reorganized as new facts are entered.
- This process prevents the addition of excessive memory entries and much redundancy which would result if every event entered resulted in the addition of a separate event
- Reorganization can also cause forgetting, since originally assigned indices may be changed when new structures are formed
- When this occurs, an item cannot be located, so the system attempts to derive new indices from the context and through other indices by reconstructing related events
- The key issues in this type of the organizations are:
    - ✓ The selection and computation of good indices for new events so that similar events can be located in memory for new event integration
    - ✓ Monitoring and reorganization of memory to accommodate new events as they occur
    - ✓ Access of the correct event information when provided clues for retrieval

Module -4

Natural Language Processing :

Developing programs to understand natural language is important in AI because a natural form of communication with systems is essential for user acceptance. One of the most critical tests for intelligent behavior is the ability to communicate effectively. This was the test proposed by Alan Turing. AI programs must be able to communicate with their human counterparts in a natural way, and natural language is one of the most important mediums for that purpose. A program understands a natural language if it behaves by taking a correct or acceptable action in response to the input. For example, we say a child demonstrates understanding if it responds with the correct answer to a question. The action taken need not be the external response. It may be the creation of some internal data structures. The structures created should be meaningful and correctly interact with the world model representation held by the program. In this chapter we explore many of the important issues related to natural language understanding and language generation.

This chapter explores several techniques that are used to enable humans to interact with computers via natural human languages. Natural languages are the languages used by humans for communication (among other functions). They are distinctly different from formal languages, such as C++, Java, and PROLOG. One of the main differences, which we will examine in some detail in this chapter, is that natural languages are ambiguous, meaning that a given sentence can have more than one possible meaning, and in some cases the correct meaning can be very hard to determine. Formal languages are almost always designed to ensure that ambiguity cannot occur. Hence, a given program written in C++ can have only one interpretation. This is clearly desirable because otherwise the computer would have to make an arbitrary decision as to which interpretation to work with. It is becoming increasingly important for computers to be able to understand natural languages. Telephone systems are now widespread that are able to understand a narrow range of commands and questions to assist callers to large call centers, without needing to use human resources. Additionally, the quantity of unstructured textual data that exists in the world (and in particular, on the Internet) has reached unmanageable proportions. For humans to search through these data using traditional techniques such as Boolean queries or the database query language SQL is impractical. The idea that people should be able to pose questions in their own language, or something similar to it, is an increasingly popular one. Of course, English is not the only natural language. A great deal of research in natural language processing and information retrieval is carried out in English, but many human languages differ enormously from English. Languages such as Chinese, Finnish, and Navajo have almost nothing in common with English (although of course Finnish uses the same alphabet). Hence, a system that can work with one human language cannot necessarily deal with any other human language. In this section we will explore two main topics. First, we will examine natural language processing, which is a collection of techniques used to enable computers to "understand" human language. In general, they are concerned with extracting grammatical information as well as meaning from human utterances but they are also concerned with understanding those utterances, and performing useful tasks as a result. Two of the earliest goals of natural language processing were automated translation (which is explored in this chapter) and database access. The idea here was that if a user wanted to find some information from a database, it would

make much more sense if he or she could query the database in her language, rather than needing to learn a new formal language such as SQL. Information retrieval is a collection of techniques used to try to match a query (or a command) to a set of documents from an existing corpus of documents. Systems such as the search engines that we use to find data on the Internet use information retrieval (albeit of a fairly simple nature).

Overview of linguistics

In dealing with natural language, a computer system needs to be able to process and manipulate language at a number of levels.

- Phonology. This is needed only if the computer is required to understand spoken language. Phonology is the study of the sounds that make up words and is used to identify words from sounds. We will explore this in a little more detail later, when we look at the ways in which computers can understand speech.
- Morphology. This is the first stage of analysis that is applied to words, once they have been identified from speech, or input into the system. Morphology looks at the ways in which words break down into components and how that affects their

grammatical status. For example, the letter “s” on the end of a word can often either indicate that it is a plural noun or a third-person present-tense verb.

- Syntax. This stage involves applying the rules of the grammar from the language being used. Syntax determines the role of each word in a sentence and, thus, enables a computer system to convert sentences into a structure that can be more easily manipulated.
- Semantics. This involves the examination of the meaning of words and sentences. As we will see, it is possible for a sentence to be syntactically correct but to be semantically meaningless. Conversely, it is desirable that a computer system be able to understand sentences with incorrect syntax but that still convey useful information semantically.
- Pragmatics. This is the application of human-like understanding to sentences and discourse to determine meanings that are not immediately clear from the semantics. For example, if someone says, “Can you tell me the time?”, most people know that “yes” is not a suitable answer. Pragmatics enables a computer system to give a sensible answer to questions like this.
- In addition to these levels of analysis, natural language processing systems must apply some kind of world knowledge. In most real-world systems, this world knowledge is limited to a specific domain (e.g., a system might have detailed knowledge about the Blocks World and be able to answer questions about this world). The ultimate goal of natural language processing would be to have a system with enough world knowledge to be able to engage a human in discussion on any subject. This goal is still a long way off.
- **Morphological Analysis**
    - In studying the English language, morphology is relatively simple. We have endings such as -ing, -s, and -ed, which are applied to verbs; endings such as -s and -es, which are applied to nouns; we also have the ending -ly, which usually indicates that a word is an adverb.
    - We also have prefixes such as anti-, non-, un-, and in-, which tend to indicate negation, or opposition.
    - We also have a number of other prefixes and suffixes that provide a variety of semantic and syntactic information.

- In practice, however, morphologic analysis for the English language is not terribly complex, particularly when compared with agglutinative languages such as German, which tend to combine words together into single words to indicate combinations of meaning.
- Morphologic analysis is mainly useful in natural language processing for identifying parts of speech (nouns, verbs, etc.) and for identifying which words belong together.
- In English, word order tends to provide more of this information than morphology, however. In languages such as Latin, word order was almost entirely superficial, and the morphology was extremely important. Languages such as French, Italian, and Spanish lie somewhere between these two extremes.
- As we will see in the following sections, being able to identify the part of speech for each word is essential to understanding a sentence. This can partly be achieved by simply looking up each word in a dictionary, which might contain for example the following entries:

  (swims, verb, present, singular, third person)

  (swimmer, noun, singular)

  (swim, verb, present, singular, first and second persons)

  (swim, verb, present plural, first, second, and third persons)

  (swimming, participle)

  (swimmingly, adverb)

  (swam, verb, past)

- Clearly, a complete dictionary of this kind would be unfeasibly large. A more practical approach is to include information about standard endings, such as:

  (-ly, adverb)

  (-ed, verb, past)

  (-s, noun, plural)

- This works fine for regular verbs, such as walk, but for all natural languages there are large numbers of irregular verbs, which do not follow these rules. Verbs such as to be and to do are particularly difficult in English as they do not seem to follow any morphologic rules.
- The most sensible approach to morphologic analysis is thus to include a set of rules that work for most regular words and then a list of irregular words.
- For a system that was designed to converse on any subject, this second list would be extremely long. Most natural language systems currently are designed to discuss fairly limited domains and so do not need to include over-large look-up tables.
- In most natural languages, as well as the problem posed by the fact that word order tends to have more importance than morphology, there is also the difficulty of ambiguity at a word level.
- This kind of ambiguity can be seen in particular in words such as trains, which could be a plural noun or a singular verb, and set, which can be a noun, verb, or adjective.

- **BNF**
    - Parsing involves mapping a linear piece of text onto a hierarchy that represents the way the various words interact with each other syntactically.
    - First, we will look at grammars, which are used to represent the rules that define how a specific language is built up.
    - Most natural languages are made up of a number of parts of speech, mainly the following:
        - Verb
        - Noun
        - Adjective
        - Adverb
        - Conjunction
        - Pronoun
        - Article
    - In fact it is useful when parsing to combine words together to form syntactic groups. Hence, the words, a dog, which consist of an article and a noun, can also be described as a noun phrase.

- A noun phrase is one or more words that combine together to represent an object or thing that can be described by a noun. Hence, the following are valid noun phrases: christmas, the dog, that packet of chips, the boy who had measles last year and nearly died, my favorite color
- A noun phrase is not a sentence—it is part of a sentence.
- A verb phrase is one or more words that represent an action. The following are valid verb phrases: swim, eat that packet of chips, walking
- A simple way to describe a sentence is to say that it consists of a noun phrase and a verb phrase. Hence, for example: That dog is eating my packet of chips.
- In this sentence, that dog is a noun phrase, and is eating my packet of chips is a verb phrase. Note that the verb phrase is in fact made up of a verb phrase, is eating, and a noun phrase, my packet of chips.
- A language is defined partly by its grammar. The rules of grammar for a language such as English can be written out in full, although it would be a complex process to do so.
- To allow a natural language processing system to parse sentences, it needs to have knowledge of the rules that describe how a valid sentence can be constructed.
- These rules are often written in what is known as Backus–Naur form (also known as Backus normal form—both names are abbreviated as BNF).
- BNF is widely used by computer scientists to define formal languages such as C++ and Java. We can also use it to define the grammar of a natural language.
- A grammar specified in BNF consists of the following components:
    - Terminal symbols. Each terminal symbol is a symbol or word that appears in the language itself. In English, for example, the terminal symbols are our dictionary words such as the, cat, dog, and so on. In formal languages, the terminal symbols include variable names such as x, y, and so on, but for our purposes we will consider the terminal symbols to be the words in the language.
    - Nonterminal symbols. These are the symbols such as noun, verb phrase, and conjunction that are used to define words and phrases of

the language. A nonterminal symbol is so-named because it is used to represent one or more terminal symbols.

  - The start symbol. The start symbol is used to represent a complete sentence in the language. In our case, the start symbol is simply sentence, but in first-order predicate logic, for example, the start symbol would be expression.
  - Rewrite rules. The rewrite rules define the structure of the grammar. Each rewrite rule details what symbols (terminal or nonterminal) can be used to make up each nonterminal symbol.

- Let us now look at rewrite rules in more detail. We saw above that a sentence could take the following form: noun phrase verb phrase
- We thus write the following rewrite rule: Sentence→NounPhrase VerbPhrase This does not mean that every sentence must be of this form, but simply that a string of symbols that takes on the form of the right-hand side can be rewritten in the form of the left-hand side. Hence, if we see the words

  The cat sat on the mat

- we might identify that the cat is a noun phrase and that sat on the mat is a verb phrase. We can thus conclude that this string forms a sentence.
- We can also use BNF to define a number of possible noun phrases.
- Note how we use the "|" symbol to separate the possible right-hand sides in BNF:

  NounPhrase→ Noun

  | Article Noun

  | Adjective Noun

  | Article Adjective Noun

- Similarly, we can define a verb phrase:

  VerbPhrase→ Verb

  | Verb NounPhrase

  | Adverb Verb NounPhrase

- The structure of human languages varies considerably. Hence, a set of rules like this will be valid for one language, but not necessarily for any other language.
- For example, in English it is usual to place the adjective before the noun (black cat, stale bread), whereas in French, it is often the case that the adjective comes after the noun (moulin rouge). Thus far, the rewrite rules we have written consist solely of nonterminal symbols.
- Rewrite rules are also used to describe the parts of speech of individual words (or terminal symbols):

Noun→ cat

| dog

| Mount Rushmore

| chickens

Verb→ swims

| eats

| climbs

Article→ the

| a

Adjective→ black

| brown

| green

| stale

Grammars and Languages

- The types of grammars that exist are Noam Chomsky invented a hierarchy of grammars.

- The hierarchy consists of four main types of grammars.
- The simplest grammars are used to define regular languages.
- A regular language is one that can be described or understood by a finite state automaton. Such languages are very simplistic and allow sentences such as “aaaaabbbbbb.” Recall that a finite state automaton consists of a finite number of states, and rules that define how the automaton can transition from one state to another.
- A finite state automaton could be designed that defined the language that consisted of a string of one or more occurrences of the letter a. Hence, the following strings would be valid strings in this language:

  aaa

  a

  aaaaaaaaaaaaaaaaa

- Regular languages are of interest to computer scientists, but are not of great interest to the field of natural language processing because they are not powerful enough to represent even simple formal languages, let alone the more complex natural languages.
- Sentences defined by a regular grammar are often known as regular expressions.
- The grammar that we defined above using rewrite rules is a context-free grammar.
- It is context free because it defines the grammar simply in terms of which word types can go together—it does not specify the way that words should agree with each.

  A stale dog climbs Mount Rushmore.

- It also, allows the following sentence, which is not grammatically correct:

  Chickens eats.

- A context-free grammar can have only at most one terminal symbol on the right-hand side of its rewrite rules.
- Rewrite rules for a context-sensitive grammar, in contrast, can have more than one terminal symbol on the right-hand side. This enables the grammar to specify number, case, tense, and gender agreement.
- Each context-sensitive rewrite rule must have at least as many symbols on the right-hand side as it does on the left-hand side.

- Rewrite rules for context-sensitive grammars have the following form:

  A X B→A Y B

  which means that in the context of A and B, X can be rewritten as Y.

- Each of A, B, X, and Y can be either a terminal or a nonterminal symbol.
- Context-sensitive grammars are most usually used for natural language processing because they are powerful enough to define the kinds of grammars that natural languages use. Unfortunately, they tend to involve a much larger number of rules and are a much less natural way to describe language, making them harder for human developers to design than context free grammars.
- The final class of grammars in Chomsky's hierarchy consists of recursively enumerable grammars (also known as unrestricted grammars).
- A recursively enumerable grammar can define any language and has no restrictions on the structure of its rewrite rules. Such grammars are of interest to computer scientists but are not of great use in the study of natural language processing.
- Parsing: Syntactic Analysis
  - ➢ As we have seen, morphologic analysis can be used to determine to which part of speech each word in a sentence belongs. We will now examine how this information is used to determine the syntactic structure of a sentence.
  - ➢ This process, in which we convert a sentence into a tree that represents the sentence's syntactic structure, is known as parsing.
  - ➢ Parsing a sentence tells us whether it is a valid sentence, as defined by our grammar
  - ➢ If a sentence is not a valid sentence, then it cannot be parsed. Parsing a sentence involves producing a tree, such as that shown in Fig 10.1, which shows the parse tree for the following sentence:

    The black cat crossed the road.

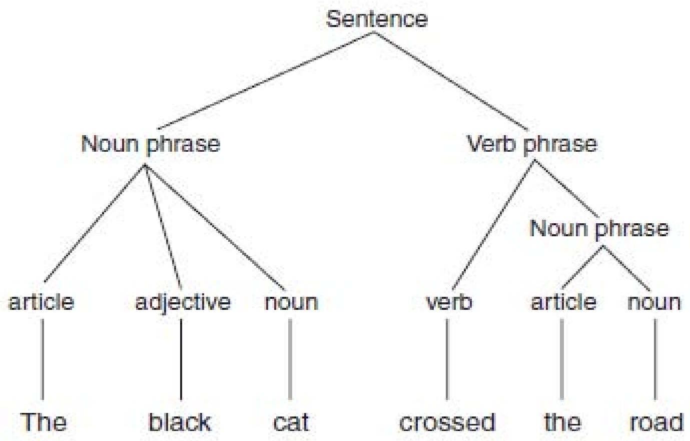


Fig 10.1

- This tree shows how the sentence is made up of a noun phrase and a verb phrase.
- The noun phrase consists of an article, an adjective, and a noun. The verb phrase consists of a verb and a further noun phrase, which in turn consists of an article and a noun.
- Parse trees can be built in a bottom-up fashion or in a top-down fashion.
- Building a parse tree from the top down involves starting from a sentence and determining which of the possible rewrites for Sentence can be applied to the sentence that is being parsed. Hence, in this case, Sentence would be rewritten using the following rule:

  Sentence→NounPhrase VerbPhrase

- Then the verb phrase and noun phrase would be broken down recursively in the same way, until only terminal symbols were left.
- When a parse tree is built from the top down, it is known as a derivation tree.
- To build a parse tree from the bottom up, the terminal symbols of the sentence are first replaced by their corresponding nonterminals (e.g., cat is replaced by noun), and then these nonterminals are combined to match the right-hand sides of rewrite rules.
- For example, the and road would be combined using the following rewrite rule: NounPhrase→Article Noun

Basic parsing techniques

- Transition Networks
  - A transition network is a finite state automaton that is used to represent a part of a grammar.
  - A transition network parser uses a number of these transition networks to represent its entire grammar.
  - Each network represents one nonterminal symbol in the grammar. Hence, in the grammar for the English language, we would have one transition network for Sentence, one for Noun Phrase, one for Verb Phrase, one for Verb, and so on.
  - Fig 10.2 shows the transition network equivalents for three production rules.
  - In each transition network, S1 is the start state, and the accepting state, or final state, is denoted by a heavy border. When a phrase is applied to a transition network, the first word is compared against one of the arcs leading from the first state.
  - If this word matches one of those arcs, the network moves into the state to which that arc points. Hence, the first network shown in Fig 10.2, when presented with a Noun Phrase, will move from state S1 to state S2.
  - If a phrase is presented to a transition network and no match is found from the current state, then that network cannot be used and another network must be tried. Hence, when starting with the phrase the cat sat on the mat, none of the networks shown in Fig 10.2 will be used because they all have only nonterminal symbols, whereas all the symbols in the cat sat on the mat are terminal.Hence, we need further networks, such as the ones shown in Figure 10.2, which deal with terminal symbols.

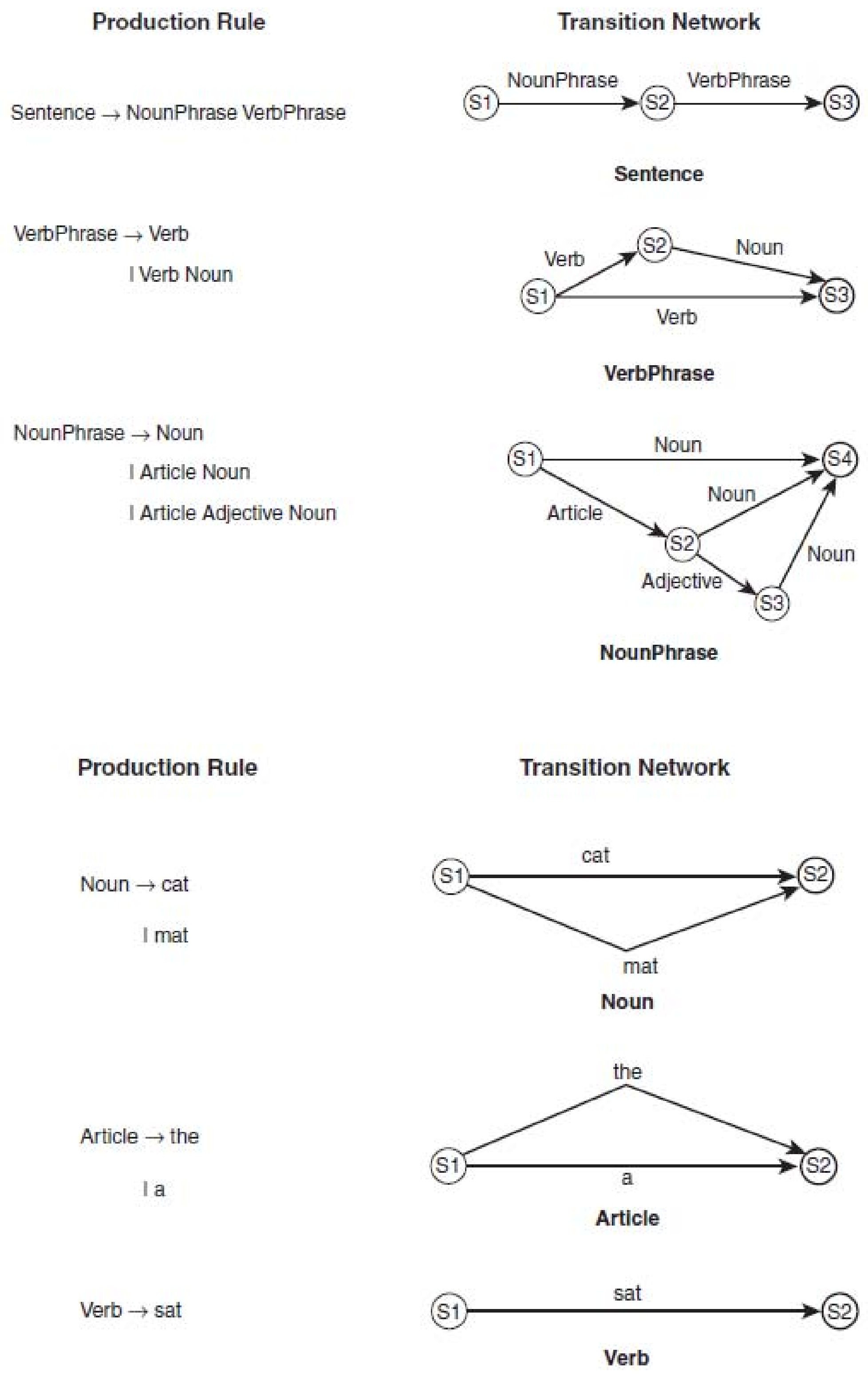

Production Rule
Transition Network
Sentence → NounPhrase VerbPhrase
S1
NounPhrase
S2
VerbPhrase
S3
Sentence
VerbPhrase → Verb
| Verb Noun
Verb
Noun
Verb
VerbPhrase
NounPhrase → Noun
| Article Noun
| Article Adjective Noun
Noun
Article
Noun
Adjective
Noun
S4
NounPhrase
Production Rule
Transition Network
Noun → cat
| mat
cat
mat
Noun
Article → the
| a
the
a
Article
Verb → sat
sat
Verb

Fig 10.2

- Transition networks can be used to determine whether a sentence is grammatically correct, at least according to the rules of the grammar the networks represent.
- Parsing using transition networks involves exploring a search space of possible parses in a depth-first fashion.
- Let us examine the parse of the following simple sentence:

  A cat sat.

- We begin in state S1 in the Sentence transition network. To proceed, we must follow the arc that is labeled NounPhrase. We thus move out of the Sentence network and into the NounPhrase network.
- The first arc of the NounPhrase network is labeled Noun. We thus move into the Noun network. We now follow each of the arcs in the Noun network and discover that our first word, A, does not match any of them. Hence, we backtrack to the next arc in the NounPhrase network.
- This arc is labeled Article, so we move on to the Article transition network. Here, on examining the second label, we find that the first word is matched by the terminal symbol on this arc.
- We therefore consume the word, A, and move on to state S2 in the Article network. Because this is a success node, we are able to return to the NounPhrase network and move on to state S2 in this network. We now have an arc labeled Noun.
- As before, we move into the Noun network and find that our next word, cat, matches. We thus move to state S4 in the NounPhrase network. This is a success node, and so we move back to the Sentence network and repeat the process for the VerbPhrase arc.
- It is possible for a system to use transition networks to generate a derivation tree for a sentence, so that as well as determining whether the sentence is grammatically valid, it parses it fully to obtain further information by semantic analysis from the sentence.
- This can be done by simply having the system build up the tree by noting which arcs it successfully followed. When, for example, it successfully follows the NounPhrase arc in the Sentence network, the system generates a root node labeled Sentence and an arc leading from that node to a new node

labeled NounPhrase.When the system follows the NounPhrase network and identifies an article and a noun, these are similarly added to the tree.

  - In this way, the full parse tree for the sentence can be generated using transition networks. Parsing using transition networks is simple to understand, but is not necessarily as efficient or as effective as we might hope for. In particular, it does not pay any attention to potential ambiguities or the need for words to agree with each other in case, gender, or number.

- **Augmented Transition Networks**
  - An augmented transition network, or ATN, is an extended version of a transition network.
  - ATNs have the ability to apply tests to arcs, for example, to ensure agreement with number. Thus, an ATN for Sentence would be as shown in Figure 10.2, but the arc from node S2 to S3 would be conditional on the number of the verb being the same as the number for the noun.
  - Hence, if the noun phrase were three dogs and the verb phrase were is blue, the ATN would not be able to follow the arc from node S2 to S3 because the number of the noun phrase (plural) does not match the number of the verb phrase (singular).
  - In languages such as French, checks for gender would also be necessary. The conditions on the arcs are calculated by procedures that are attached to the arcs. The procedure attached to an arc is called when the network reaches that arc. These procedures, as well as carrying out checks on agreement, are able to form a parse tree from the sentence that is being analyzed.

- **Chart Parsing**
  - Parsing using transition networks is effective, but not the most efficient way to parse natural language. One problem can be seen in examining the following two sentences: 1. Have all the fish been fed? , Have all the fish.
  - Clearly these are very different sentences—the first is a question, and the second is an instruction. In spite of this, the first threewords of each sentence are the same.

- When a parser is examining one of these sentences, it is quite likely to have to backtrack to the beginning if it makes the wrong choice in the first case for the structure of the sentence. In longer sentences, this can be a much greater problem, particularly as it involves examining the same words more than once, without using the fact that the words have already been analyzed.

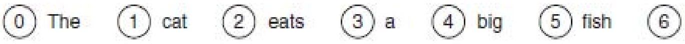


Fig 10.3

- Another method that is sometimes used for parsing natural language is chart parsing.
- In the worst case, chart parsing will parse a sentence of n words in O(n3) time. In many cases it will perform better than this and will parse most sentences in O(n2) or even O(n) time.
- In examining sentence 1 above, the chart parser would note that the words two children form a noun phrase. It would note this on its first pass through the sentence and would store this information in a chart, meaning it would not need to examine those words again on a subsequent pass, after backtracking.
- The initial chart for the sentence The cat eats a big fish is shown in Fig 10.3 It shows the chart that the chart parse algorithm would start with for parsing the sentence.
- The chart consists of seven vertices, which will become connected to each other by edges. The edges will show how the constituents of the sentence combine together.
- The chart parser starts by adding the following edge to the chart:

  [0, 0, Target→• Sentence]

- This notation means that the edge connects vertex 0 to itself (the first two numbers in the square brackets show which vertices the edge connects).
- Target is the target that we want to find, which is really just a placeholder to enable us to have an edge that requires us to find a whole sentence. The arrow indicates that in order to make what is on its left-hand side (Target) we need to find what is on its right-hand side (Sentence). The dot (•) shows

what has been found already, on its left-hand side, and what is yet to be found, on its right-hand side. This is perhaps best explained by examining an example.

- ➢ Consider the following edge, which is shown in the chart in Figure 10.4:

  [0, 2, Sentence→NounPhrase • VerbPhrase]

- ➢ This means that an edge exists connecting nodes 0 and 2. The dot shows us that we have already found a NounPhrase (the cat) and that we are looking for a VerbPhrase.

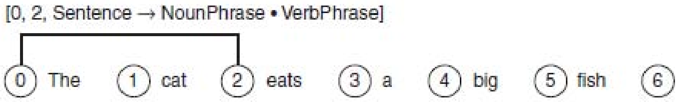


Fig 10.4

- ➢ Once we have found the VerbPhrase, we will have what is on the left-hand side of the arrow—that is, a Sentence.
- ➢ The chart parser can add edges to the chart using the following three rules:
  - o If we have an edge [x, y, A →□B • C], which needs to find a C, then an edge can be added that supplies that C (i.e., the edge [x, y, C→ • E]), where E is some sequence of terminals or nonterminals which can be replaced by a C).
  - o If we have two edges, [x, y, A →□B • C D] and [y, z, C →□E •}, then these two edges can be combined together to form a new edge: [x, z, A→B C • D].
  - o If we have an edge [x, y, A →□B • C], and the word at vertex y is of type C, then we have found a suitable word for this edge, and so we extend the edge along to the next vertex by adding the following edge: [y, y + 1, A→B C •].

- **Semantic Analysis**
  - ➢ Having determined the syntactic structure of a sentence, the next task of natural language processing is to determine the meaning of the sentence.
  - ➢ Semantics is the study of the meaning of words, and semantic analysis is the analysis we use to extract meaning from utterances.

- Semantic analysis involves building up a representation of the objects and actions that a sentence is describing, including details provided by adjectives, adverbs, and prepositions. Hence, after analyzing the sentence The black cat sat on the mat, the system would use a semantic net such as the one shown in Figure 10.5 to represent the objects and the relationships between them.

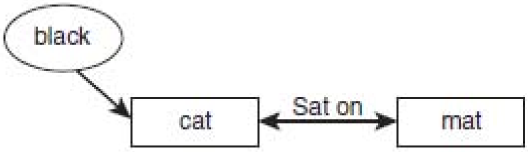


Fig 10.5

- A more sophisticated semantic network is likely to be formed, which includes information about the nature of a cat (a cat is an object, an animal, a quadruped, etc.) that can be used to deduce facts about the cat (e.g., that it likes to drink milk).

- Ambiguity and Pragmatic Analysis
  - One of the main differences between natural languages and formal languages like C++ is that a sentence in a natural language can have more than one meaning. This is ambiguity—the fact that a sentence can be interpreted in different ways depending on who is speaking, the context in which it is spoken, and a number of other factors.
  - The more common forms of ambiguity and look at ways in which a natural language processing system can make sensible decisions about how to disambiguate them.
  - Lexical ambiguity occurs when a word has more than one possible meaning. For example, a bat can be a flying mammal or a piece of sporting equipment. The word set is an interesting example of this because it can be used as a verb, a noun, an adjective, or an adverb. Determining which part of speech is intended can often be achieved by a parser in cases where only one analysis is possible, but in other cases semantic disambiguation is needed to determine which meaning is intended.
  - Syntactic ambiguity occurs when there is more than one possible parse of a sentence. The sentence Jane carried the girl with the spade could be

interpreted in two different ways, as is shown in the two parse trees in Fig 10.6. In the first of the two parse trees in Fig 10.6, the prepositional phrase with the spade is applied to the noun phrase the girl, indicating that it was the girl who had a spade that Jane carried. In the second sentence, the prepositional phrase has been attached to the verb phrase carried the girl, indicating that Jane somehow used the spade to carry the girl.

- Semantic ambiguity occurs when a sentence has more than one possible meaning—often as a result of a syntactic ambiguity. In the example shown in Fig 10.6 for example, the sentence Jane carried the girl with the spade, the sentence has two different parses, which correspond to two possible meanings for the sentence. The significance of this becomes clearer for practical systems if we imagine a robot that receives vocal instructions from a human.

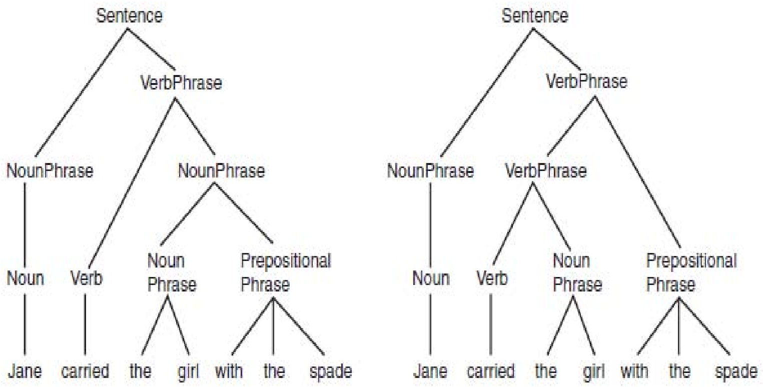


Fig 10.6

- Referential ambiguity occurs when we use anaphoric expressions, or pronouns to refer to objects that have already been discussed. An anaphora occurs when a word or phrase is used to refer to something without naming it. The problem of ambiguity occurs where it is not immediately clear which object is being referred to. For example, consider the following sentences:

  John gave Bob the sandwich. He smiled.

- It is not at all clear from this who smiled—it could have been John or Bob. In general, English speakers or writers avoid constructions such as this to avoid humans becoming confused by the ambiguity. In spite of this, ambiguity can also occur in a similar way where a human would not have a problem, such as

John gave the dog the sandwich. It wagged its tail.

- In this case, a human listener would know very well that it was the dog that wagged its tail, and not the sandwich. Without specific world knowledge, the natural language processing system might not find it so obvious.
- A local ambiguity occurs when a part of a sentence is ambiguous; however, when the whole sentence is examined, the ambiguity is resolved. For example, in the sentence There are longer rivers than the Thames, the phrase longer rivers is ambiguous until we read the rest of the sentence, than the Thames.
- Another cause of ambiguity in human language is vagueness. we examined fuzzy logic, words such as tall, high, and fast are vague and do not have precise numeric meanings.
- The process by which a natural language processing system determines which meaning is intended by an ambiguous utterance is known as disambiguation.
- Disambiguation can be done in a number of ways. One of the most effective ways to overcome many forms of ambiguity is to use probability.
- This can be done using prior probabilities or conditional probabilities. Prior probability might be used to tell the system that the word bat nearly always means a piece of sporting equipment.
- Conditional probability would tell it that when the word bat is used by a sports fan, this is likely to be the case, but that when it is spoken by a naturalist it is more likely to be a winged mammal.
- Context is also an extremely important tool in disambiguation. Consider the following sentences:

  I went into the cave. It was full of bats.

  I looked in the locker. It was full of bats.

- In each case, the second sentence is the same, but the context provided by the first sentence helps us to choose the correct meaning of the word “bat” in each case.
- Disambiguation thus requires a good world model, which contains knowledge about the world that can be used to determine the most likely

meaning of a given word or sentence. The world model would help the system to understand that the sentence Jane carried the girl with the spade is unlikely to mean that Jane used the spade to carry the girl because spades are usually used to carry smaller things than girls. The challenge, of course, is to encode this knowledge in a way that can be used effectively and efficiently by the system.

- The world model needs to be as broad as the sentences the system is likely to hear. For example, a natural language processing system devoted to answering sports questions might not need to know how to disambiguate the sporting bat from the winged mammal, but a system designed to answer any type of question would.

Expert System Architecture

An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal, articles and databases. This type of knowledge usually requires much training and experience in some specialized field such as medicine, geology, system configuration, or engineering design. Once a sufficient body of expert knowledge has been auquired, it must be encoded in some form, loaded into a knowledge base, then tested, and refined continually throughout the life of the system

Characteristics Features of Expert Systems

Expert systems differ from conventional computer system in several important ways

1. Expert systems use knowledge rather than data to control the solution process. Much of the knowledge used in heuristic in nature rather than algorithmic
2. The knowledge is encoded and maintained as an entity separate from the aontrol program. As such, it is not complicated together with the control program itself. This permits the incremental addition and modification of the knowledge base without recompilation of the control programs. Furthermore, it is possible in some cases to use different knowledge bases with the same control programs to produce different types

of expert systems. Such systems are known as expert system shells since they may be loaded with different knowledge bases

3. Expert systems are capable of explaining how a particular conclusion was reached, and why requested information is needed during a consultation. This is important as it gives the user a chance to assess and understand the systems reasoning ability, thereby improving the user's confidence in the system
4. Expert systems use symbolic representations for knowledge and perform their inference through symbolic computations that closely resemble manipulations of natural language
5. Expert systems often reason with metaknowledge, that is, they reason with knowledge about themselves, and their own knowledge limits and capabilities

Rules for Knowledge Representation

- One way to represent knowledge is by using rules that express what must happen or what does happen when certain conditions are met.
- Rules are usually expressed in the form of IF . . . THEN . . . statements, such as: IF A THEN B This can be considered to have a similar logical meaning as the following: $A \rightarrow B$
- A is called the antecedent and B is the consequent in this statement.
- In expressing rules, the consequent usually takes the form of an action or a conclusion.
- In other words, the purpose of a rule is usually to tell a system (such as an expert system) what to do in certain circumstances, or what conclusions to draw from a set of inputs about the current situation.
- In general, a rule can have more than one antecedent, usually combined either by AND or by OR (logically the same as the operators $\wedge$ and $\vee$).
- Similarly, a rule may have more than one consequent, which usually suggests that there are multiple actions to be taken.
- In general, the antecedent of a rule compares an object with a possible value, using an operator.
- For example, suitable antecedents in a rule might be

  IF $x > 3$

IF name is “Bob”

IF weather is cold

- Here, the objects being considered are x, name, and weather; the operators are “>” and “is”, and the values are 3, “Bob,” and cold.
- Note that an object is not necessarily an object in the real-world sense—the weather is not a real world object, but rather a state or condition of the world.
- An object in this sense is simply a variable that represents some physical object or state in the real world.
- An example of a rule might be

IF name is “Bob”

AND weather is cold

THEN tell Bob ‘Wear a coat’

- This is an example of a recommendation rule, which takes a set of inputsand gives advice as a result.
- The conclusion of the rule is actually an action, and the action takes the form of a recommendation to Bob that he should wear a coat.
- In some cases, the rules provide more definite actions such as “move left” or “close door,” in which case the rules are being used to represent directives.
- Rules can also be used to represent relations such as:

IF temperature is below 0

THEN weather is cold

Rule-Based Systems

- **Rule-based systems** or **production systems** are computer systems that use rules to provide recommendations or diagnoses, or to determine a course of action in a particular situation or to solve a particular problem.
- A rule-based system consists of a number of components:
  - a database of rules (also called a **knowledge base**)
  - a database of facts

    - an **interpreter**, or **inference engine**

- In a rule-based system, the knowledge base consists of a set of rules that represent the knowledge that the system has.
- The database of facts represents inputs to the system that are used to derive conclusions, or to cause actions.
- The interpreter, or inference engine, is the part of the system that controls the process of deriving conclusions. It uses the rules and facts, and combines them together to draw conclusions.
- Using deduction to reach a conclusion from a set of antecedents is called **forward chaining**.
- An alternative method, **backward chaining**, starts from a conclusion and tries to show it by following a logical path backward from the conclusion to a set of antecedents that are in the database of facts.
- **Forward Chaining**
    - Forward chaining employs the system starts from a set of facts, and a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.
    - This is known as **data-driven reasoning** because the reasoning starts from a set of data and ends up at the goal, which is the conclusion.
    - When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database.
    - When all the antecedents of a rule are matched by facts in the database, then this rule is **triggered**.
    - Usually, when a rule is triggered, it is then **fired**, which means its conclusion is added to the facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.
    - For example, consider the following set of rules that is used to control an elevator in a three-story building:

Rule 1

IF on first floor and button is pressed on first floor

THEN open door

Rule 2

IF on first floor

AND button is pressed on second floor

THEN go to second floor

Rule 3

IF on first floor

AND button is pressed on third floor

THEN go to third floor

Rule 4

IF on second floor

AND button is pressed on first floor

AND already going to third floor

THEN remember to go to first floor later

- This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.
- Let us imagine that we start with the following facts in our database:

Fact 1

At first floor

Fact 2

Button pressed on third floor

Fact 3

Today is Tuesday

- Now the system examines the rules and finds that Facts 1 and 2 match the antecedents of Rule 3. Hence, Rule 3 fires, and its conclusion “Go to third floor” is added to the database of facts. Presumably, this results in the elevator heading toward the third floor.
- Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.
- Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact Button pressed on first floor
- Is now added to the database, which results in Rule 4 firing.
- Now let us imagine that later in the day the facts database contains the following information:

  Fact 1

  At first floor

  Fact 2

  Button pressed on second floor

  Fact 3

  Button pressed on third floor

- In this case, two rules are triggered—Rules 2 and 3. In such cases where there is more than one possible conclusion, **conflict resolution** needs to be applied to decide which rule to fire.

- **Conflict Resolution**
  - In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire.
  - For example, consider the following set of rules:

    IF it is cold

    THEN wear a coat

    IF it is cold

THEN stay at home

IF it is cold

THEN turn on the heat

- If there is a single fact in the fact database, which is “it is cold,” then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible.
- In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:

  IF patient has pain

  THEN prescribe painkillers priority 10

  IF patient has chest pain

  THEN treat for heart disease priority 100

- Here, it is clear that treating possible heart problems is more important than just curing the pain.
- An alternative method is the **longest-matching strategy**. This method involves firing the conclusion that was derived from the longest rule.
- For example:

  IF patient has pain

  THEN prescribe painkiller

  IF patient has chest pain

  AND patient is over 60

  AND patient has history of heart conditions

  THEN take to emergency room

    - Here, if all the antecedents of the second rule match, then this rule's conclusion should be fired rather than the conclusion of the first rule because it is a more specific match.
    - A further method for conflict resolution is to fire the rule that has matched the facts most recently added to the database.
    - In each case, it may be that the system fires one rule and then stops, but in many cases, the system simply needs to choose a suitable ordering for the rules because each rule that matches the facts needs to be fired at some point.

- **Meta Rules**
    - In designing an expert system, it is necessary to select the conflict resolution method that will be used, and quite possibly it will be necessary to use different methods to resolve different types of conflicts.
    - For example, in some situations it may make most sense to use the method that involves firing the most recently added rules.
    - This method makes most sense in situations in which the timeliness of data is important. It might be, for example, that as research in a particular field of medicine develops, and new rules are added to the system that contradicts some of the older rules.
    - It might make most sense for the system to assume that these newer rules are more accurate than the older rules.
    - It might also be the case, however, that the new rules have been added by an expert whose opinion is less trusted than that of the expert who added the earlier rules.
    - In this case, it clearly makes more sense to allow the earlier rules priority.
    - This kind of knowledge is called **meta knowledge—**knowledge about knowledge. The rules that define how conflict resolution will be used, and how other aspects of the system itself will run, are called **meta rules**.
    - The knowledge engineer who builds the expert system is responsible for building appropriate meta knowledge into the system (such as "expert A is to be trusted more than expert B" or "any rule that involves drug X is not to be trusted as much as rules that do not involve X").
    - Meta rules are treated by the expert system as if they were ordinary rules but are given greater priority than the normal rules that make up the expert system.

In this way, the meta rules are able to override the normal rules, if necessary, and are certainly able to control the conflict resolution process.

- **Backward Chaining**
  - Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived, which is useful when a set of facts are present, but you do not know what conclusions you are trying to prove.
  - Forward chaining can be inefficient because it may end up proving a number of conclusions that are not currently interesting.
  - In such cases, where a single specific conclusion is to be proved, **backward chaining** is more appropriate.
  - 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 database.
  - The conclusion we are aiming to prove is called a **goal**, and so reasoning in this way is known as **goal-driven reasoning**.
  - Backward chaining is often used in formulating plans.
  - A plan is a sequence of actions that a program decides to take to solve a particular problem.
  - Backward chaining can make the process of formulating a plan more efficient than forward chaining.
  - Backward chaining in this way starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it.
  - For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table.
  - This action might not be possible from the start state, and so further actions need to be added before this action in order to reach it from the start state.
  - In this way, a plan can be formulated starting from the goal and working back toward the start state.
  - The benefit in this method is particularly clear in situations where the first state allows a very large number of possible actions.
  - In this kind of situation, it can be very inefficient to attempt to formulate a plan using forward chaining because it involves examining every possible

    action, without paying any attention to which action might be the best one to lead to the goal state.

  - Backward chaining ensures that each action that is taken is one that will definitely lead to the goal, and in many cases this will make the planning process far more efficient.

- **Comparing Forward and Backward Chaining**
  - Let us use an example to compare forward and backward chaining. In this case, we will revert to our use of symbols for logical statements, in order to clarify the explanation, but we could equally well be using rules about elevators or the weather.

Rules:

Rule 1 A ^ B → C

Rule 2 A → D

Rule 3 C ^ D → E

Rule 4 B ^ E ^ F → G

Rule 5 A ^ E → H

Rule 6 D ^ E ^ H → I

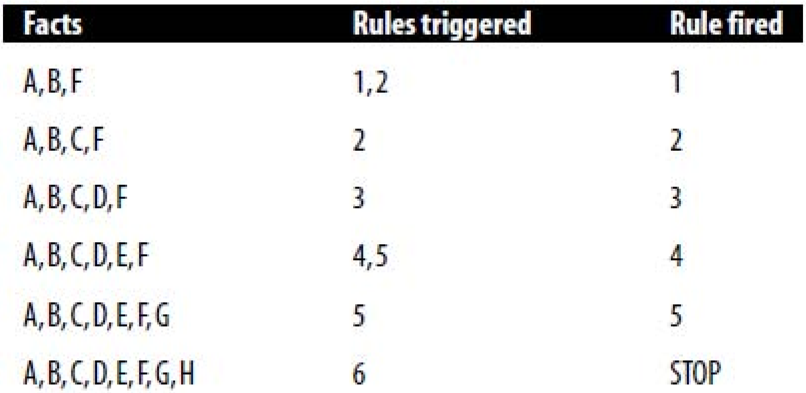
Facts:

Fact 1	A
Fact 2	B
Fact 3	F

Goal:

Our goal is to prove H.

- First let us use forward chaining. As our conflict resolution strategy, we will fire rules in the order they appear in the database, starting from Rule 1.
- In the initial state, Rules 1 and 2 are both triggered. We will start by firing Rule 1, which means we add C to our fact database. Next, Rule 2 is fired, meaning we add D to our fact database.
- We now have the facts A, B, C, D, F, but we have not yet reached our goal, which is G.
- Now Rule 3 is triggered and fired, meaning that fact E is added to the database.
- As a result, Rules 4 and 5 are triggered. Rule 4 is fired first, resulting in Fact G being added to the database, and then Rule 5 is fired, and Fact H is added to the database.
- We have now proved our goal and do not need to go on any further.
- This deduction is presented in the following table:

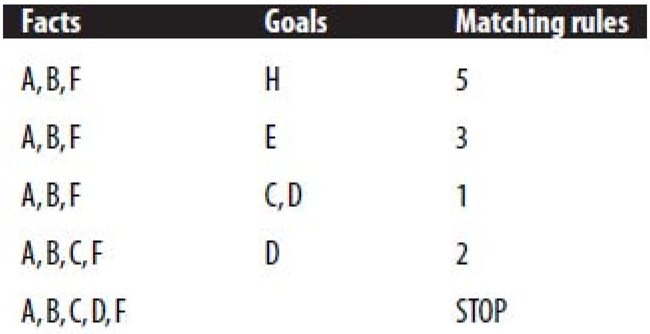


Facts	Rules triggered	Rule fired
A,B,F	1,2	1
A,B,C,F	2	2
A,B,C,D,F	3	3
A,B,C,D,E,F	4,5	4
A,B,C,D,E,F,G	5	5
A,B,C,D,E,F,G,H	6	STOP

- Now we will consider the same problem using backward chaining. To do so, we will use a goals database in addition to the rule and fact databases.
- In this case, the goals database starts with just the conclusion, H, which we want to prove. We will now see which rules would need to fire to lead to this conclusion.
- Rule 5 is the only one that has H as a conclusion, so to prove H, we must prove the antecedents of Rule 5, which are A and E.
- Fact A is already in the database, so we only need to prove the other antecedent, E. Therefore, E is added to the goal database. Once we have

proved E, we now know that this is sufficient to prove H, so we can remove H from the goals database.

- So now we attempt to prove Fact E. Rule 3 has E as its conclusion, so to prove E, we must prove the antecedents of Rule 3, which are C and D.
- Neither of these facts is in the fact database, so we need to prove both of them. They are both therefore added to the goals database. D is the conclusion of Rule 2 and Rule 2's antecedent, A, is already in the fact database, so we can conclude D and add it to the fact database.
- Similarly, C is the conclusion of Rule 1, and Rule 1's antecedents, A and B, are both in the fact database. So, we have now proved all the goals in the goal database and have therefore proved H and can stop.
- This process is represented in the table below:



Facts	Goals	Matching rules
A, B, F	H	5
A, B, F	E	3
A, B, F	C, D	1
A, B, C, F	D	2
A, B, C, D, F		STOP

- In this case, backward chaining needed to use one fewer rule. If the rule database had had a large number of other rules that had A, B, and F as their antecedents, then forward chaining might well have been even more inefficient.
- In general, backward chaining is appropriate in cases where there are few possible conclusions (or even just one) and many possible facts, not very many of which are necessarily relevant to the conclusion.
- Forward chaining is more appropriate when there are many possible conclusions.
- The way in which forward or backward chaining is usually chosen is to consider which way an expert would solve the problem. This is particularly appropriate because rule-based reasoning is often used in **expert systems**.

Rule-Based Expert Systems

- An expert system is one designed to model the behavior of an expert in some field, such as medicine or geology.
- Rule-based expert systems are designed to be able to use the same rules that the expert would use to draw conclusions from a set of facts that are presented to the system.
- **The People Involved in an Expert System**
  - The design, development, and use of expert systems involves a number of people.
  - The **end-user** of the system is the person who has the need for the system.
  - In the case of a medical diagnosis system, this may be a doctor, or it may be an individual who has a complaint that they wish to diagnose.
  - The **knowledge engineer** is the person who designs the rules for the system, based on either observing the expert at work or by asking the expert questions about how he or she works.
  - The **domain expert** is very important to the design of an expert system. In the case of a medical diagnosis system, the expert needs to be able to explain to the knowledge engineer how he or she goes about diagnosing illnesses.
- **Architecture of an Expert System**
  - Typical expert system architecture is shown in Figure 11.1.
  - The knowledge base contains the specific domain knowledge that is used by an expert to derive conclusions from facts.
  - In the case of a rule-based expert system, this domain knowledge is expressed in the form of a series of rules.
  - The explanation system provides information to the user about how the inference engine arrived at its conclusions. This can often be essential, particularly if the advice being given is of a critical nature, such as with a medical diagnosis system.

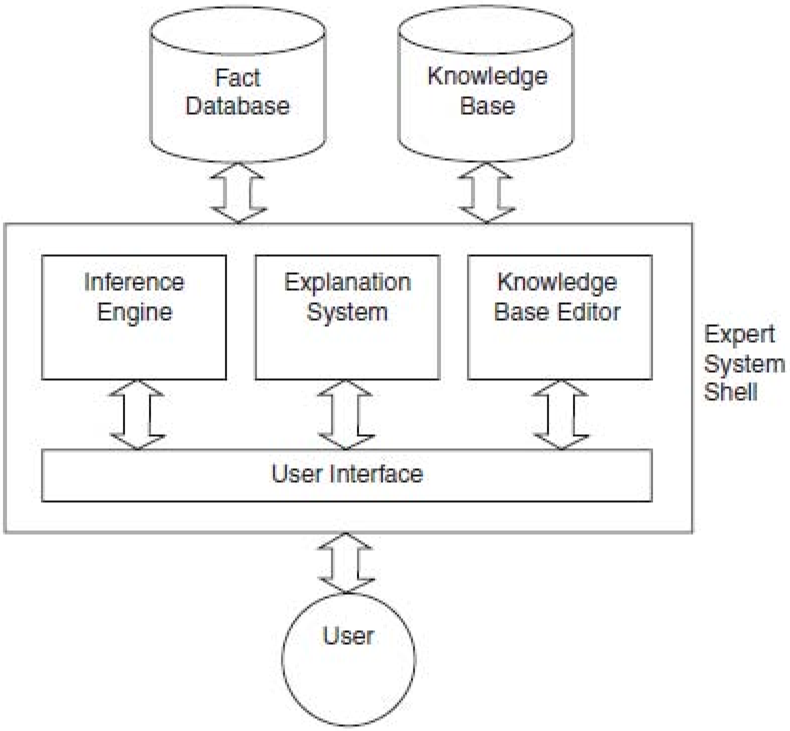


Fig Expert System Architecture

- If the system has used faulty reasoning to arrive at its conclusions, then the user may be able to see this by examining the data given by the explanation system.
- The fact database contains the case-specific data that are to be used in a particular case to derive a conclusion.
- In the case of a medical expert system, this would contain information that had been obtained about the patient’s condition.
- The user of the expert system interfaces with it through a user interface, which provides access to the inference engine, the explanation system, and the knowledge-base editor.
- The inference engine is the part of the system that uses the rules and facts to derive conclusions. The inference engine will use forward chaining, backward chaining, or a combination of the two to make inferences from the data that are available to it.
- The knowledge-base editor allows the user to edit the information that is contained in the knowledge base.

    - The knowledge-base editor is not usually made available to the end user of the system but is used by the knowledge engineer or the expert to provide and update the knowledge that is contained within the system.

- **The Expert System Shell**
    - Note that in Figure 11.1, the parts of the expert system that do not contain domain-specific or case-specific information are contained within the **expert system shell**.
    - This shell is a general toolkit that can be used to build a number of different expert systems, depending on which knowledge base is added to the shell.
    - An example of such a shell is CLIPS (C Language Integrated Production System). Other examples in common use include OPS5, ART, JESS, and Eclipse.
- **Knowledge Engineering**
    - Knowledge engineering is a vital part of the development of any expert system.
    - The knowledge engineer does not need to have expert domain knowledge but does need to know how to convert such expertise into the rules that the system will use, preferably in an efficient manner.
    - Hence, the knowledge engineer's main task is communicating with the expert, in order to understand fully how the expert goes about evaluating evidence and what methods he or she uses to derive conclusions.
    - Having built up a good understanding of the rules the expert uses to draw conclusions, the knowledge engineer must encode these rules in the expert system shell language that is being used for the task.
    - In some cases, the knowledge engineer will have freedom to choose the most appropriate expert system shell for the task.
    - In other cases, this decision will have already been made, and the knowledge engineer must work with what he is given.

CLIPS (C Language Integrated Production System)

- CLIPS is a freely available expert system shell that has been implemented in C.

- It provides a language for expressing rules and mainly uses forward chaining to derive conclusions from a set of facts and rules.
- The notation used by CLIPS is very similar to that used by LISP.
- The following is an example of a rule specified using CLIPS:

```
(defrule birthday

(firstname ?r1 John)

(surname ?r1 Smith)

(haircolor ?r1 Red)

=>

(assert (is-boss ?r1)))
```

- *?r1* is used to represent a variable, which in this case is a person.
- *Assert* is used to add facts to the database, and in this case the rule is used to draw a conclusion from three facts about the person:
- If the person has the first name John, has the surname Smith, and has red hair, then he is the boss.
- This can be tried in the following way:

```
(assert (firstname x John))

(assert (surname x Smith))

(assert (haircolor x Red))

(run)
```

- At this point, the command (facts) can be entered to see the facts that are contained in the database:

```
CLIPS> (facts)

f-0 (firstname x John)

f-1 (surname x Smith)

f-2 (haircolor x Red)
```

```
f-3 (is-boss x)
```

- So CLIPS has taken the three facts that were entered into the system and used the rule to draw a conclusion, which is that *x* is the boss.
- Although this is a simple example, CLIPS, like other expert system shells, can be used to build extremely sophisticated and powerful tools.
- For example, MYCIN is a well-known medical expert system that was developed at Stanford University in 1984.
- MYCIN was designed to assist doctors to prescribe antimicrobial drugs for blood infections.
- In this way, experts in antimicrobial drugs are able to provide their expertise to other doctors who are not so expert in that field. By asking the doctor a series of questions, MYCIN is able to recommend a course of treatment for the patient.
- Importantly, MYCIN is also able to explain to the doctor which rules fired and therefore is able to explain why it produced the diagnosis and recommended treatment that it did.
- MYCIN has proved successful: for example, it has been proven to be able to provide more accurate diagnoses of meningitis in patients than most doctors.
- MYCIN was developed using LISP, and its rules are expressed as LISP expressions.
- The following is an example of the kind of rule used by MYCIN, translated into English:

    IF the infection is primary-bacteria

    AND the site of the culture is one of the sterile sites

    AND the suspected portal of entry is the gastrointestinal tract

    THEN there is suggestive evidence (0.7) that infection is bacteroid

- The following is a very simple example of a CLIPS session where rules are defined to operate an elevator:

```
CLIPS> (defrule rule1

(elevator ?floor_now)

(button ?floor_now)
=>
(assert (open_door)))
CLIPS> (defrule rule2
(elevator ?floor_now)
(button ?other_floor)
=>
(assert (goto ?other_floor)))
CLIPS> (assert (elevator floor1))
==> f-0 (elevator floor1)
<Fact-0>
CLIPS> (assert (button floor3))
==> f-1 (button floor3)
<Fact-1>
<CLIPS> (run)
==>f-2 (goto floor3)
```

- The segments in **bold** are inputs by the knowledge engineer, and the plain text sections are CLIPS.
- Note that **?floor_now** is an example of a variable within CLIPS, which means that any object can match it for the rule to trigger and fire.
- In our example, the first rule simply says: If the elevator is on a floor, and the button is pressed on the same floor, then open the door.
- The second rule says: If the elevator is on one floor, and the button is pressed on a different floor, then go to that floor.
- After the rules, two facts are inserted into the database. The first fact says that the elevator is on floor 1, and the second fact says that the button has been pressed on floor 3.

- When the *(run)* command is issued to the system, it inserts a new fact into the database, which is a command to the elevator to go to floor 3.

Backward Chaining in Rule-Based Expert Systems

- A common method for building expert systems is to use a rule-based system with backward chaining. Typically, a user enters a set of facts into the system, and the system tries to see if it can prove any of the possible hypotheses using these facts.
- In some cases, it will need additional facts, in which case the expert system will often ask the user questions, to ascertain facts that could enable further rules to fire.
- The algorithm is applied as follows:
- To prove a conclusion, we must prove a set of hypotheses, one of which is the conclusion. For each hypothesis, H:

  If H is in the facts database, it is proved.

  Otherwise, if H can be determined by asking a question, then enter

  the user’s answer in the facts database. Hence, it can be determined

  whether H is true or false, according to the user’s answer.

  Otherwise, find a rule whose conclusion is H. Now apply this algorithm

  to try to prove this rule’s antecedents.

  If none of the above applies, we have failed to prove H.

- Typically, backward chaining is used in combination with forward chaining. Whenever a new fact is added to the database, forward chaining is applied to see if any further facts can be derived.
- Backward chaining is then used to try to prove each possible hypothesis.
- Let us imagine a simple medical expert system that is designed to diagnose and treat patients’ diseases. The rules might be as follows:

  Rule 1: If headache then prescribe painkiller

  Rule 2: If headache and sore throat and coughing then diagnose flu

  Rule 3: If tired and headache then diagnose glandular fever

Rule 4: If tired and sore throat then diagnose tonsillitis

Rule 5: If tired then diagnose stress

- Now we might imagine a session between the user (the patient) and the expert system as follows:

  Patient: I have a headache

  System: I suggest you take a painkiller. Do you have a sore throat?

  Patient: Yes, I do.

  System: Have you been coughing?

  Patient: No, why do you ask?

  System: If you had a headache and a sore throat and had been
  coughing, I would have concluded you might have flu. It
  doesn't sound as though you have flu. Have you been feeling
  tired recently?

  Patient: Yes, I have.

  System: It sounds as though you may have glandular fever or tonsillitis.

- This was a rather simplistic example, but it illustrates the way that the system uses forward chaining to derive new facts and asks questions in order to carry out its backward chaining analysis.
- For example, having been told that the patient has a headache, the system first applies forward chaining with Rule 1 and, as a result, adds "prescribe painkiller" to its database of facts.
- This fact has an action, or a recommendation, associated with it, and so the system advises the patient that she should take a painkiller.
- Next, the system tries to see if it can prove any other hypotheses. The possible hypotheses are flu, tonsillitis, glandular fever, and stress. First, the system uses backward chaining to try to prove the hypothesis that the patient has the flu.

- To prove this hypothesis, the antecedents of Rule 2 must be proved: that the patient has a headache and a sore throat and has been coughing. The patient has already said that she has a headache, so this fact is already in the fact database. Next, the system must establish whether the patient has a sore throat. She says that she does, so this fact is added to the fact database. She has not been coughing, though, so the system concludes that she does not have flu.
- At this point also note that the patient asks why the system asked the last question. The system is able to use its explanation facility to provide an explanation for why it asked the question and what conclusion it was able to draw from the answer.
- Finally, the patient says that she has been feeling tired, and as a result of this fact being added to the database, Rules 3, 4, and 5 are all triggered.
- In this case, conflict resolution has been applied in a rather simplistic way, such that Rules 3 and 4 both fire, but 5 does not.
- In a real medical expert system, it is likely that further questions would be asked, and more sophisticated rules applied to decide which condition the patient really had.

CYC

- CYC is an example of a frame-based representational system of knowledge, which is, in a way, the opposite of an expert system. Whereas an expert system has detailed knowledge of a very narrow domain, the developers of CYC have fed it information on over 100,000 different concepts from all fields of human knowledge. CYC also has information of over 1,000,000 different pieces of “common sense” knowledge about those concepts.
- The system has over 4000 different *types* of links that can exist between concepts, such as inheritance, and the “is–a” relationship that we have already looked at.
- The idea behind CYC was that humans function in the world mainly on the basis of a large base of knowledge built up over our lifetimes and our ancestors’ lifetimes.
- By giving CYC access to this knowledge, and the ability to reason about it, they felt they would be able to come up with a system with common sense. Ultimately, they predict, the system will be built into word processors.
- Then word processors will not just correct your spelling and grammar, but will also point out inconsistencies in your document.
- For example, if you promise to discuss a particular subject later in your document, and then forget to do so, the system will point this out to you. They also predict that

search engines and other information retrieval systems will be able to find documents even though they do not contain any of the words you entered as your query.

- CYC’s knowledge is segmented into hundreds of different contexts to avoid the problem of many pieces of knowledge in the system contradicting each other.
- In this way, CYC is able to know facts about Dracula and to reason about him, while also knowing that Dracula does not really exist.
- CYC is able to understand analogies, and even to discover new analogies for itself, by examining the similarities in structure and content between different frames and groups of frames. CYC’s developers claim, for example, that it discovered an analogy between the concept of “family” and the concept of “country.”