# Introduction to Artificial Intelligence and Expert Systems

## DCAP310

Edited by:
Dr. Anil Sharma

LOVELY PROFESSIONAL UNIVERSITY


# INTRODUCTION TO ARTIFICIAL INTELLIGENCE & EXPERT SYSTEMS

Edited By

Dr. Anil Sharma

Printed by
**EXCEL BOOKS PRIVATE LIMITED**
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

# SYLLABUS

## Introduction to Artificial Intelligence & Expert Systems

*Objectives:*

- To enable the student to understand technicalities of intelligence.
- To enable the student to understand techniques of capturing and generating knowledge.
- To enable the student to understand knowledge representation methodologies.
- To enable the student to learn Natural language processing.
- To enable the student to learn Fuzzy Logic with their applications.
- To enable the student to understand probabilistic reasoning.
- To enable the student to learn technicalities of expert system.
- To enable the student to understand Artificial intelligence language 'LISP" and 'Prolog'.

| Sr. No. | Description |
|---|---|
| 1. | **Overview of AI:** What is AI, Importance of AI, Early Work in AI, AI and Related Fields. |
| 2. | **Knowledge:** General Concepts, Introduction, Definition and Importance of Knowledge, Knowledge-based Systems, Representation of Knowledge, Knowledge Organization, Knowledge Manipulation, Acquisition of Knowledge. |
| 3. | **LISP and Other AI Programming Languages:** Introduction to LISP Syntax and Numeric Functions, Basic List Manipulation Functions in LISP, Functions, Predicates, and Conditionals, Input, Outputs and Local Variables, Iterations and Recursion, Property Lists and Arrays, PROLOG and Other AI Programming Languages. |
| 4. | **Formalized Symbolic Logics:** Introduction, Syntax and Semantics for Propositional Logic, Syntax and Semantics for FOPL, Properties of WFFs, Conversion to Clausal Form, Inference Rules, The Resolution Principle, Non-deductive Inference Methods, Representations using Rules Dealing with Inconsistencies and Uncertainties: Truth Maintenance System, Predicated Completion and Circumscription, Modal and Temporal Logics, Fuzzy Logic and Natural Language Computation. |
| 5. | **Probabilistic Reasoning:** Bayesian Probabilistic Inference, Possible World Representations, Dempster-Shafer Theory, Ad-Hoc Methods, Heuristic Reasoning Methods. Structured Knowledge: Associative Networks, Frame Structures, Conceptual Dependencies and Scripts. Object Oriented Representation: Overview of Object-Oriented Systems, Object, Classes, Messages and Methods. |
| 6. | **Search and Control Strategies:** Preliminary Concepts, Examples of Search Problems, Uninformed or Blind Search, Informed Search, Searching And-Or Graph. |
| 7. | **Matching Techniques:** Structures used in Matching, Measures for Matching, Matching Like Patterns, Partial Matching, Fuzzy Matching Algorithms, The RETE Matching Algorithm. Knowledge Organization and Management: Indexing and Retrieval Techniques, Integrating Knowledge in Memory. |
| 8. | **Natural Language Processing:** Overview of Linguistics, Grammars and Languages, Basic Parsing Techniques, Semantic Analysis and Representation Structures, Natural Language Generalization, Natural Language Systems, Recognition and Classification Process. |
| 9. | **Expert System Architecture:** Rule-Based Architecture, Nonproduction System Architectures, Dealing with Uncertainty, Knowledge Acquisition and Validation. |
| 10. | **Types of Learning, Knowledge Acquisition is Difficult, General Learning Model, Performance Measures, Knowledge System Building Tools, Learning by Induction:** Generalization and Specialization, Inductive Bias. Analogical Reasoning and Learning, Explanation based Learning. |

# CONTENT

# Unit 1: Overview of Artificial Intelligence

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Define Artificial Intelligence
- Describe the goals of AI and the importance of Artificial Intelligence
- Elaborate on the Early Work in Artificial Intelligence
- Discuss the Common Techniques Used in AI

Notes

# Introduction

Artificial Intelligence (AI) is the intelligence of machines or software and is also a branch of computer science that studies and develops intelligent machines and software. This field of computer science, AI, is defined as "the study and design of intelligent agents" where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. It is also defined as "the science and engineering of making intelligent machines".

AI research is highly technical and specialized, deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. There are subfields which are focused on the solution of specific problems, on one of several possible approaches, on the use of widely differing tools and towards the accomplishment of particular applications. There is no established unifying theory or paradigm that guides AI research. Researchers disagree about many issues A few of the most long standing questions that have remained unanswered are these: should artificial intelligence simulate natural intelligence by studying psychology or neurology?

The central problems (or goals) of AI research include reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects. General intelligence or "strong AI" is still among the field's long term goals. Currently popular approaches include statistical methods, computational intelligence and traditional symbolic AI. There are an enormous number of tools used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics, and many others. In this unit, you will be able to understand the meaning of Artificial Intelligence, its goals and importance, the early work and common techniques used in Artificial Intelligence.

## 1.1 What is AI?

Artificial intelligence (AI) is the science of programming computers to perform complex tasks that normally require human intelligence. Artificial intelligence systems used in information security applications include Artificial Neural Networks, expert systems and genetic algorithms.

AI is an emerging technology that has recently attracted considerable publicity. Many applications are now under development. One simple view of AI is that it is concerned with devising computer programs to make computers smarter. Thus, research in AI is focused on developing computational approaches to intelligent behavior. This research has two goals:

1. Making machines more useful.
2. Understanding intelligence.

The computer programs with which AI is concerned are primarily symbolic processes involving complexity, uncertainty, and ambiguity. These processes are usually those for which algorithmic solutions do not exist and search is required. Thus, AI deals with the types of problem-solving and decision-making that humans continually face in dealing with the world.

### 1.1.1 Goals of AI

The general problem of simulating (or creating) intelligence has been broken down into a number of specific sub-problems. These consist of particular traits or capabilities that researchers would like an intelligent system to display. The traits described below have received the most attention.

Notes

## Deduction, Reasoning and Problem Solving

Early AI researchers developed algorithms that imitated the step-by-step reasoning that humans use when they solve puzzles or make logical deductions. By the late 1980s and 1990s, AI research had also developed highly successful methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.

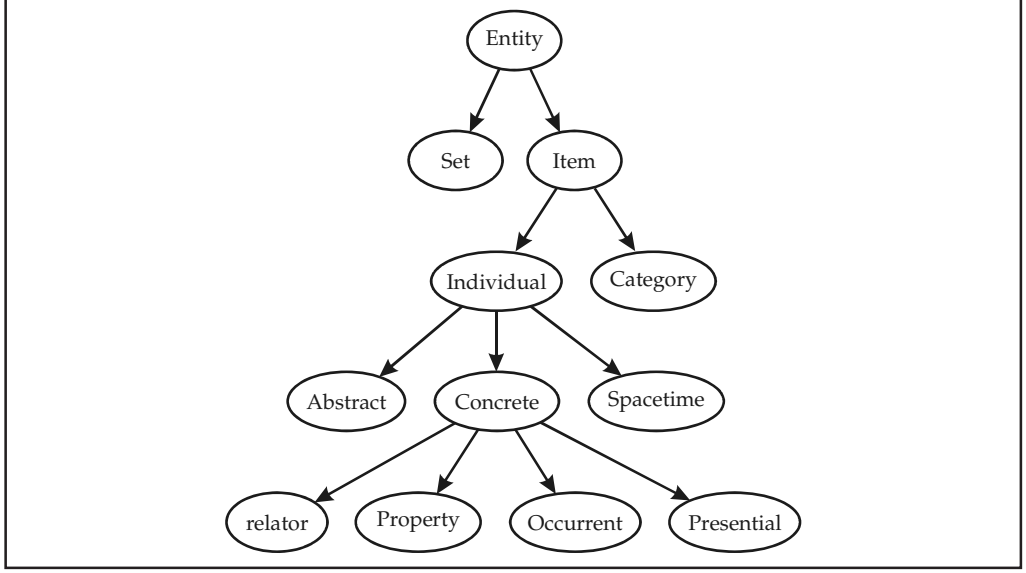
For difficult problems, most of these algorithms can require enormous computational resources – most experience a "combinatorial explosion": the amount of memory or computer time required becomes astronomical when the problem goes beyond a certain size. The search for more efficient problem-solving algorithms is a high priority for AI research.

Human beings solve most of their problems using fast, intuitive judgments rather than the conscious, step-by-step deduction that early AI research was able to model. AI has made some progress at imitating this kind of "sub-symbolic" problem solving: embodied agent approaches emphasize the importance of sensory motor skills to higher reasoning; neural net research attempts to simulate the structures inside the brain that give rise to this skill; statistical approaches to AI mimic the probabilistic nature of the human ability to guess.

## Knowledge Representation

Ontology represents knowledge as a set of concepts within a domain and the relationships between those concepts.

**Figure 1.1: Knowledge Representation**




Knowledge representation and knowledge engineering are central to AI research. Many of the problems machines are expected to solve will require extensive knowledge about the world. Among the things that AI needs to represent are: objects, properties, categories and relations between objects; situations, events, states and time; causes and effects; knowledge about knowledge and many other, less well researched domains. A representation of "what exists" is ontology: the set of objects, relations, concepts and so on that the machine knows about. The most general are called upper ontologism, which attempt provides a foundation for all other knowledge.

Notes

## 1.2 Importance of Artificial Intelligence

Artificial Intelligence deals with study and building of rational systems. The Artificial Intelligence has the following advantages:

(i) This method is more general as compared to the laws of thought method.

(ii) This method responds to science than to the methods based on human behaviour or thought.

(iii) The first computer program introduced to beat the champion chess player was Hitech. Limited intellectual skills are required for the chess strategies.

(iv) Artificial Intelligence is divided into many branches. Artificial Intelligence search programs to study the various probabilities which include chess game moves. Different ways are being set up to carry out more proficiently. Logical Artificial Intelligence deals with mathematical logical program. A program of pattern recognition studies and compares with a particular pattern. It matches the eyes and nose to the face of a police id. Epistemology deals with the study of different types of knowledge required to solve problems.

(v) Currently, voice recognition program failed to recognize every word but is slowly leading towards improvement. This idea guide disabled people to utilize computers.

(vi) Now computers perform many tasks that were beyond our imagination. By 2020, computers will be more intelligent than the mankind. Due to these advances we will not be inferior to computers as we can turn it off when it fails to perform.

### 1.2.1 Artificial Intelligence in Manufacturing

AI in manufacturing is wide. It can be described an interactive note-taking system for pen-based computers with two distinctive features. The system is an example of a learning apprentice software-agent. A machine learning component characterises the syntax and semantics of the users information. In machine learning that acquires knowledge for some of the higher level processing; it was found that performance equals that of a partially trainable discourse module requiring manual customisation for each domain. The central assumption of the model is that the learner is embedded within an environment of related learning tasks. Explicit bounds were also derived demonstrating that learning multiple tasks within an environment of related tasks could potentially give much better generalization than learning a single task.

Knowledge and Intelligence are as fundamental as the universe within which they exist, it may turn out that they are more fundamental. Enterprises that utilize AI-enhanced applications are expected to become more diverse, as the needs for the ability to analyze data across multiple variables, fraud detection and customer relationship management emerge as key business drivers to gain competitive advantage. Artificial Intelligence is a branch of Science which deals with helping machines, finds 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.

### 1.2.2 Artificial Intelligence in Financial Services

Banks use artificial intelligence systems to organize operations, invest in stocks and manage properties. In August 2001, robots beat humans in a simulated financial trading competition.

Financial institutions have long used artificial neural network systems to detect charges or claims outside of the norm, flagging these for human investigation.

Notes

Creative Virtual has deployed artificial intelligence customer support systems, or automated online assistants at HSBC and Lloyds Banking Group, to assist financial services customers with services such as checking an account balance, signing up for a new credit card or retrieving a forgotten password.

### 1.2.3 Artificial Intelligence in Marketing

Artificial intelligence is a field of study that "seeks to explain and emulate intelligent behavior in terms of computational processes" through performing the tasks of decision making, problem solving and learning. Unlike other fields associated with intelligence, Artificial intelligence is concerned with both understanding and building of intelligent entities, and has the ability to automate intelligent processes. It is evident that artificial intelligence is impacting on a variety of subfields and wider society. However, literature regarding its application to the field of marketing appears to be scarce. Advancements in Artificial intelligence's application to a range of disciplines have led to the development of Artificial intelligence systems which have proved useful to marketers. These systems assist in areas such as market forecasting, automation of processes and decision making and increase the efficiency of tasks which would usually be performed by humans. The science behind these systems can be explained through neural networks and expert systems which are computer programs that process input and provide valuable output for marketers. In the area of social networking, AI is used to Artificial intelligence systems stemming from Social computing technology can be applied to understand social networks on the Web. Data mining techniques can be used to analyze different types of social networks. This analysis helps a marketer to identify influential actors or nodes within networks, this information can then be applied to take a Societal marketing approach. Artificial intelligence has gained significant recognition in the marketing industry. However, ethical issues surrounding these systems and their potential to impact on the need for humans in the workforce, specifically marketing, is a controversial topic. AI-enhanced analytics programs also provide survival modeling capabilities suggesting changes to products based on use.

*Example:* Customer patterns are analyzed to learn ways to extend the life of light bulbs or to help decide the correct dosage for medications. High-tech data mining can give companies a precise view of how particular segments of the customer base react to a product or service and propose changes consistent with those findings.

### 1.2.4 Artificial Intelligence in HR

Artificial Intelligence (AI) has been used in business applications since the early eighties. As with all technologies, AI initially generated much interest, but failed to live up to the hype. However, with the advent of web-enabled infrastructure and rapid strides made by the AI development community, the application of AI techniques in real-time business applications has picked up substantially in the recent past.

Computers are fundamentally well suited to performing mechanical computations, using fixed programmed rules. This allows artificial machines to perform simple monotonous tasks efficiently and reliably, which humans are ill-suited to. For more complex problems, things get more difficult. Unlike humans, computers have trouble understanding specific situations, and adapting to new situations. Artificial Intelligence aims to improve machine behavior in tackling such complex tasks.

Together with this, much of AI research is allowing us to understand our intelligent behavior. Humans have an interesting approach to problem-solving, based on abstract thought, high-level deliberative reasoning and pattern recognition. Artificial Intelligence can help us understand

Notes

this process by recreating it, then potentially enabling us to enhance it beyond our current capabilities. The main purposes of the study are to discuss the appointment of managers in enterprises through fuzzy neural network, to construct a new model for evaluation of managerial talent, and accordingly to develop a decision support system in human resource selection. Therefore, the research methods of reviewing literature, in-depth interview, questionnaire survey, and fuzzy neural network are used in the study.

*Task* Go through various websites for the definition of AI.

*Caution* Have proper knowledge of all procedure before applying the AI.

### Self Assessment

State whether the following statements are true or false:

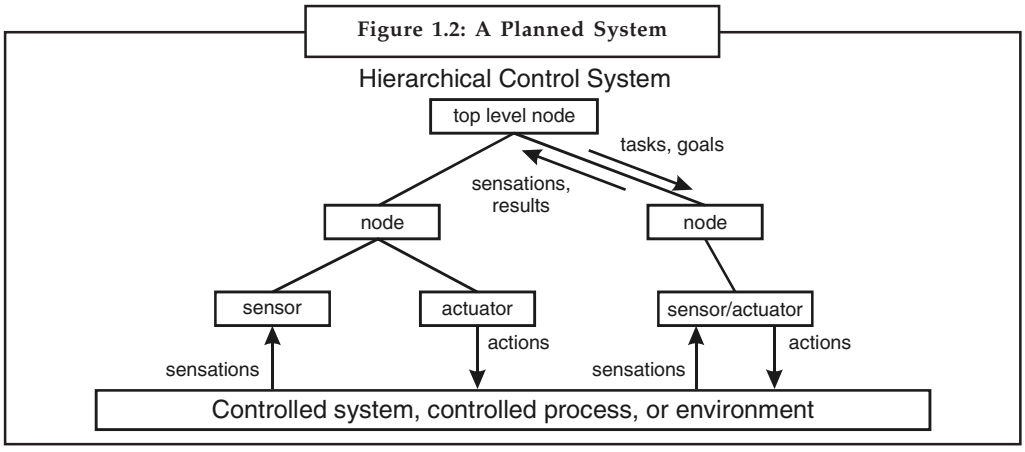
1. AI can be defined as "the science and engineering of making intelligent machines".
2. Knowledge representation and knowledge engineering are central to AI research.
3. Artificial Intelligence has no branch.
4. Banks use artificial intelligence systems to organize operations, invest in stocks and manage properties.
5. AI deals with the types of problem-solving and decision-making that humans continually face in dealing with the world.

## 1.3 Early Work in Artificial Intelligence

### 1.3.1 Expert Systems

A hierarchical control system is a form of control system in which a set of devices and governing software is arranged in a hierarchy. Intelligent agents must be able to set goals and achieve them. They need a way to visualize the future (they must have a representation of the state of the world and be able to make predictions about how their actions will change it) and be able to make choices that maximize the utility or "value" of the available choices.

**Figure 1.2: A Planned System**



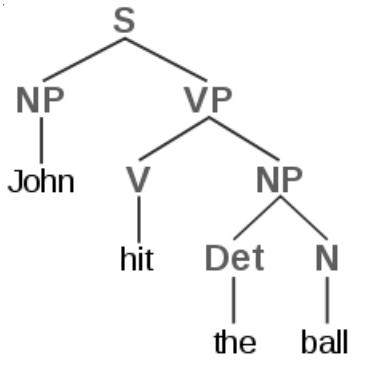
Notes

In classical planning problems, the agent can assume that it is the only thing acting on the world and it can be certain what the consequences of its actions may be. However, if the agent is not the only actor, it must periodically ascertain whether the world matches its predictions and it must change its plan as this becomes necessary, requiring the agent to reason under uncertainty.

### 1.3.2 Natural Language Processing (NLP)

Natural Language Processing gives machines the ability to read and understand the languages that humans speak. A sufficiently powerful natural language processing system would enable natural language user interfaces and the acquisition of knowledge directly from human-written sources, such as Internet texts.

**Figure 1.3: A NLP System**




A parse tree represents the syntactic structure of a sentence according to some formal grammar. Some straightforward applications of natural language processing include information retrieval (or text mining) and machine translation.

A common method of processing and extracting meaning from natural language is through semantic indexing. Increases in processing speeds and the drop in the cost of data storage makes indexing large volumes of abstractions of the users input much more efficient.

### 1.3.3 Computer Vision

Machine learning is the study of computer algorithms that improve automatically through experience and has been central to AI research since the field's inception. Unsupervised learning is the ability to find patterns in a stream of input. Supervised learning includes both classification and numerical regression. Classification is used to determine what category something belongs in, after seeing a number of examples of things from several categories. Regression is the attempt to produce a function that describes the relationship between inputs and outputs and predicts how the outputs should change as the inputs change. In reinforcement learning, the agent is rewarded for good responses and punished for bad ones. These can be analyzed in terms of decision theory, using concepts like utility. The mathematical analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory.

### 1.3.4 Intelligent Robots

Artificial Intelligence (AI) is a general term that implies the use of a computer to model and/or replicate intelligent behavior. Research in AI focuses on the development and analysis of algorithms that learn and/or perform intelligent behavior with minimal human intervention.

Notes

These techniques have been and continue to be applied to a broad range of problems that arise in robotics, e-commerce, medical diagnosis, gaming, mathematics, and military planning and logistics, to name a few.

Several research groups fall under the general umbrella of AI in the department, but are disciplines in their own right, including: robotics, NLP, computer vision, computational biology, and e-commerce. Specifically, research is being conducted in estimation theory, mobility mechanisms, multi-agent negotiation, natural language interfaces, machine learning, active computer vision, probabilistic language models for use in spoken language interfaces, and the modeling and integration of visual, haptic, auditory and motor information.

### 1.3.5 Industrial Applications

Artificial intelligence is implemented in automated online assistants that can be seen as avatars on web pages. It can avail for enterprises to reduce their operation and training cost. A major underlying technology to such systems is natural language processing. Similar techniques may be used in answering machines of call centers, such as speech recognition software to allow computers to handle first level of customer support, text mining and natural language processing to allow better customer handling, agent training by automatic mining of best practices from past interactions, support automation and many other technologies to improve agent productivity and customer satisfaction. Fuzzy logic controllers have been developed for automatic gearboxes in automobiles. Many telecommunications companies make use of heuristic search in the management of their workforces, for example BT Group has deployed heuristic search in a scheduling application that provides the work schedules of 20,000 engineers. The 1990s saw some of the first attempts to mass-produce domestically aimed types of basic Artificial Intelligence for education or leisure. This prospered greatly with the Digital Revolution and helped introduce people, especially children, to a life of dealing with various types of Artificial Intelligence, specifically in the form of Internet. The evolution of music has always been affected by technology.

### 1.3.6 Computer-aided Instruction (CAI)

With AI, scientists are trying to make the computer emulate the activities of the skillful musician. Composition, performance, music theory, sound processing are some of the major areas on which research in Music and Artificial Intelligence are focusing. The Air Operations Division uses AI for the rule based expert systems. The AOD has use for artificial intelligence for surrogate operators for combat and training simulators, mission management aids, support systems for tactical decision making, and post processing of the simulator data into symbolic summaries.

To truly learn is to digest and make the material one's own by updating one's internal models and using them in new applications. Real-time interaction with a computer providing immediate feedback and individual guidance is particularly appropriate to this goal. Thus, as computer hardware costs continue to tumble, the nature of the entire present educational system may be radically changed.

### 1.3.7 Learning by Computers

Computers can support the variety of ways learners construct their own understanding. Students who gather information from the Internet can be self-directed and independent. They can choose what sources to examine and what connections to pursue. Depending on the parameters set by teachers, the students may be in complete control of their topics and their explorations.

Students can work through a computer-based activity at their own pace. Rather than 25 individuals working together on one activity, technology allows independent completion of work. Those

Notes

who begin to fall behind can receive an instructor's individualized attention while others can begin to tackle more complex tasks.

Computer software can mix text, pictures, sound, and motion to provide a variety of options for learners. Multimedia software will not be the only classroom resource, but it can contribute richness and variety to student work.

Students can build on their own understanding by using computers as resource tools, as work stations for individual learning, or as communication channels to share their ideas with other learners. Individual understanding and experiences must be shared and compared to curriculum content. By uncovering students' individual understandings, teachers can determine the influence of students' prior knowledge and further their education through new experience.

Computers can be used to assist active experiences – gathering data and resources, conversing with colleagues, struggling through a challenging puzzle or application – or they can assist in reflection. For example, while an online conversation through e-mail is an active event, such discussions usually prompt reflection. They help us think about ideas and check our understanding. In another reflective application, teachers can enlist computers as authoring tools for students' journals which are excellent vehicles for thoughtful examination of experience.

*Notes* Introducing technology into the learning environment can encourage cooperative learning and student collaboration. If they are allowed to converse, most students like to talk about their computer work and share their strategies. Classroom activities that are structured so that computers encourage collaboration build on learners' desire to communicate and share their understanding. It takes planning and intervention to build successful cooperative groups with or without computers, but groups that use computers as teamwork tools have a better start toward collaborative work.

Beyond the classroom, computer networking allows students to communicate and collaborate with content experts and with fellow students around the globe. Communication tools like e-mail, bulletin boards, and chat groups allow teachers to exchange lesson plans and teaching strategies and create a professional community.

The use of real world tools, relevant experiences, and meaningful data inject a sense of purpose to classroom activity. Part of the mission of educational institutions is to produce workforce-ready graduates who can, among other things, manipulate and analyze raw data, critically evaluate information, and operate hardware and software. This technological literacy imparts a very important set of vocational skills that will serve students well in the working world.

Technology has allowed schools to provide greater assistance to traditionally underserved populations. Assistive technology such as voice recognition systems, dynamic Braille displays, speech synthesizers, and talking books provide learning and communication alternatives for those who have developmental or physical disabilities. Research has also shown that computer-mediated communication can ease the social isolation that may be experienced by those with disabilities. Computers have proved successful in increasing academic motivation and lessening anxiety among low ability students and learning disabled students, many of whom simply learn in a manner different from that practiced in a traditional, non-technological classroom.

### 1.3.8 Voice Recognition (VR)

In computer science, voice recognition is the translation of spoken words into text. It is also known as "automatic speech recognition", "ASR", "computer speech recognition", "speech to

Notes

text", or just "STT". Some SR systems use "training" where an individual speaker reads sections of text into the SR system. These systems analyze the person's specific voice and use it to fine tune the recognition of that person's speech, resulting in more accurate transcription. Systems that do not use training are called "Speaker Independent" systems. Systems that use training are called "Speaker Dependent" systems. Speech recognition applications include voice user interfaces such as voice dialing (e.g. "Call home"), call routing (e.g. "I would like to make a collect call"), demotic appliance control, search (e.g. find a podcast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g. a radiology report), speech-to-text processing (e.g., word processors or emails), and aircraft (usually termed Direct Voice Input). The term "voice recognition" refers to finding the identity of "who" is speaking, rather than what they are saying. Recognizing the speaker can simplify the task of translating speech in systems that have been trained on specific person's voices or it can be used to authenticate or verify the identity of a speaker as part of a security process.

### Self Assessment

State whether the following statements are true or false:

6. Natural Language Processing (NLP) gives machines the ability to read and understand the languages that humans speak.
7. Unsupervised learning includes both classification and numerical regression.
8. The Air Operations Division uses AI for the rule based expert systems.
9. Voice Recognition (VR) is also known as "automatic speech recognition" or "speech to text".

## 1.4 AI and Related Fields

The AI may be defined as a branch of Computer Science that is concerned with automation of intelligent behavior. Its principles include the data structures used and knowledge representation, the algorithms needed to apply the knowledge, and language and programming techniques used in their implementation.

### 1.4.1 Common Techniques Used in AI

***Neural Networks:*** Neural networks are structures that can be "trained" to recognize patterns in inputs. Neural networks typically take a vector of input values and produce a vector of output values. Inside, they train weights of "neurons". Neural networks use supervised learning, in which inputs and outputs are known and the goal is to build a representation of a function that will approximate the input to output mapping.

They are a way to implement function approximation: given $y_1 = f(x_1)$, $y_2 = f(x_2)$, …, $y_n = f(x_n)$, construct a function f' that approximates f. The approximate function f' is typically *smooth*: for x' close to x, we will expect that f'(x') is close to f'(x). Function approximation serves two purposes:

1. ***Size:*** The representation of the approximate function can be significantly smaller than the true function.
2. ***Generalization:*** The approximate function can be used on inputs for which we do not know the value of the function.

In path finding, the function is f(start, goal) = path. We do not already know the output paths. We could compute them in some way, perhaps by using A*. But if we are able to compute a path given (start, goal), then we already know the function f, so why bother approximating it?

Notes

There is no use in generalizing f because we know it completely. The only potential benefit would be in reducing the size of the representation of f. The representation of f is a fairly simple algorithm, which takes little space, so I don't think that's useful either. In addition, neural networks produce a fixed-size output, whereas paths are variable sized.

***Genetic Algorithms:*** Genetic Algorithms allow to explore a space of parameters to find solutions that score well according to a "fitness function". They are a way to implement function optimization: given a function g(x) (where x is typically a vector of parameter values), find the value of x that maximizes (or minimizes) g(x). This is an unsupervised learning problem. The right answer is not known beforehand. For path finding, given a starting position and a goal, x is the path between the two and g(x) is the cost of that path. Simple optimization approaches like hill-climbing will change x in ways that increase g(x). Unfortunately, in some problems, we reach "local maxima", values of x for which no nearby x has a greater value of g, but some faraway value of x is better. Genetic algorithms improve upon hill-climbing by maintaining multiple x, and using evolution-inspired approaches like mutation and crossover to alter x. Both hill-climbing and genetic algorithms can be used to learn the best value of x. For path finding, however, we already have an algorithm (A*) to find the best x, so function optimization approaches are not needed.

Genetic Programming takes genetic algorithms a step further, and treats programs as the parameters. For example, we would breed path finding algorithms instead of paths, and your fitness function would rate each algorithm based on how well it does. For path finding, we already have a good algorithm and we do not need to evolve a new one.

It may be that as with neural networks, genetic algorithms can be applied to some portion of the path finding problem.

***Reinforcement Learning:*** Like genetic algorithms, Reinforcement Learning is an unsupervised learning problem. However, unlike genetic algorithms, agents can learn during their lifetimes; it's not necessary to wait to see if they "live" or "die". Also, it's possible for multiple agents experiencing different things to share what they've learned. Reinforcement learning has some similarities to the core of A*. In A*, reaching the end goal is propagated back to mark all the choices that were made along the path; other choices are discarded. In reinforcement learning, every state can be evaluated and its reward (or punishment) is propagated back to mark all the choices that were made leading up to that state. The propagation is made using a value function, which is somewhat like the heuristic function in A*, except that it's updated as the agents try new things and learn what works. One of the key advantages of reinforcement learning and genetic algorithms over simpler approaches is that there is a choice made between exploring new things and exploiting the information learned so far. In genetic algorithms, the exploration via mutation; in reinforcement learning, the exploration is via explicitly allowing the probability of choosing new actions. As with genetic algorithms, we don't believe reinforcement learning should be used for the path finding problem itself, but instead as a guide for teaching agents how to behave in the game world.

### 1.4.2 Related Fields of AI

Following may be the related fields of AI.

(a) Robotics

(b) Computer Vision

(c) Image Processing

(d) Voice Recognition

(e) Neural Networks

**Notes**

(a) ***Artificial Intelligence – Robotics:*** Robots are a practical application of artificial intelligence of rapidly growing importance for industrial, domestic, entertainment and military tasks. The capabilities of today's robots go far beyond the factory robots of the last century, whose operations depended on the strictly controlled conditions of an assembly line. Today's field and service robots must be able operate in the unstructured world of a shop floor, a home, a school or a battlefield. Furthermore, the software controlling such machines must be far more flexible, robust and autonomous than the robot programming languages of the past. They must be able to be commanded to go about their tasks far more readily than is today possible. In order to solve such demanding problems, it is not only necessary to build and test new kinds machines, but also to create new and powerful classes of algorithms, the general applicability of which can be demonstrated by the successful operation of those machines on challenging problems. We invite research students to join us in this important work; skills in electronics, mechanics or software development are welcome, but an interest in robots is essential!

(b) ***Artificial Intelligence – Vision:*** Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

Applications range from tasks such as industrial machine vision systems which, say, inspect bottles speeding by on a production line, to research into artificial intelligence and computers or robots that can comprehend the world around them. The computer vision and machine vision fields have significant overlap. Computer vision covers the core technology of automated image analysis which is used in many fields. Machine vision usually refers to a process of combining automated image analysis with other methods and technologies to provide automated inspection and robot guidance in industrial applications.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner.

As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

- Controlling processes, *e.g.*, an industrial robot;
- Navigation, *e.g.*, by an autonomous vehicle or mobile robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Organizing information, *e.g.*, for indexing databases of images and image sequences;
- Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;
- Interaction, *e.g.*, as the input to a device for computer-human interaction, and
- Automatic inspection, *e.g.*, in manufacturing applications.

Notes

(c) ***Artificial Intelligence – Image Processing:*** We can apply artificial intelligence techniques in digital image fundamentals, image enhancement, image restoration, morphological image processing, image segmentation and edge detection, object recognition, image representation and description, colour image processing, wavelets and multi resolution processing, image compression.

(d) ***Artificial Intelligence – Voice Recognition:*** Voice recognition has existed in some form or another since the 1950s, Since then, many other companies have toyed with voice recognition, including Dragon Dictation, which launched the first speech recognition software for the PC. Soon after, the telecom industry began creating voice portals, which were intended to replace customer service representatives by giving information through voice-activated menus. Instead, they became a nuisance to cell phone users looking for answers about their overage charges.

Despite these advances, the ability to truly communicate with machines through conversation remained confined to the realm of science fiction.

The latest advances in voice recognition software have found a niche in the automotive department. Ford's SYNC feature debuted in 2007 and has now become a popular offering across its product line. In last year's Super Bowl, Chevrolet advertised On Star's ability to read live Facebook feeds aloud.

AI for speech recognition involves two basic ideas. Firstly, it involves studying the thought processes of human beings. Secondly, it deals with representing those processes via machines (like computers, robots, etc.). AI is behavior of a machine, which, if performed by a human being, would be called intelligence. It makes machines smarter and more useful, and is less expensive than natural intelligence. NLP, refers to artificial intelligence methods of communicating with a computer in a natural language like English. The main objective of a NLP program is to understand input and initiate action. The input words are scanned and matched against internally stored known words. Identification of a keyword causes some action to be taken. In this way, one can communicate with the computer in one's language.

(e) ***Artificial Intelligence – Neural Network:*** A neural network is, in essence, an attempt to simulate the brain. Neural network theory revolves around the idea that certain key properties of biological neurons can be extracted and applied to simulations, thus creating a simulated (and very much simplified) brain. The first important thing to understand then is that the components of an artificial neural network are an attempt to recreate the computing potential of the brain. The second important thing to understand, however, is that no one has ever claimed to simulate anything as complex as an actual brain. Whereas the human brain is estimated to have something on the order of ten to a hundred billion neurons, a typical Artificial Neural Network (ANN) is not likely to have more than 1,000 artificial neurons. While many types of artificial neural nets exist, most are organized according to the same basic structure. There are three components to this organization: a set of input nodes, one or more layers of 'hidden' nodes, and a set of output nodes. The input nodes take in information, and are akin to sensory organs. Whether the information is in the form of a digitized picture, or a series of stock values, or just about any other form that can be numerically expressed, this is where the net gets its initial data. The information is supplied as activation values, that is, each node is given a number, higher numbers representing greater activation. This is just like human neurons except that rather than conveying their activation level by firing more frequently, as biological neurons do, artificial neurons indicate activation by passing this activation value to connected nodes. After receiving this initial activation, information is then passed through the network.

Notes

*Did u know?* The philosophy of artificial intelligence can determine the following:

- A machine can act intelligently? Can it solve *any* problem that a person would solve by thinking?
- A human intelligence and machine intelligence are the same? Is the human brain essentially a computer?
- A machine has a mind, mental states and consciousness in the same sense humans can do? Can it *feel how things are*?
- The art of creating machines that perform functions that require intelligence when performed by people'' (Kurzweil, 1990)

*Task* Create a list of software that can solve the problems intelligently.

## Self Assessment

State whether the following statements are true or false:

10. Neural networks do not use supervised learning.
11. Genetic Programming takes genetic algorithms and treats programs as the parameters.
12. Robots are one of the applications of AI.
13. Human brain is estimated to have something on the order of hundred billion neurons called artificial neural network (ANN).

*Lab Exercise*

1. Create a list of five existing expert systems and their work domain.
2. Draw a diagram showing knowledge in a system and the place of AI.

*Case Study*

## Application Areas of AI Technology

Artificial intelligence is an integrated part of our daily life and of many fields in research. In archaeology, however, it does not (yet) play an important role. In the past twenty years archaeologists have discussed the potentials of, in particular, expert systems. They have developed some valuable systems, but the general impression is that archaeology is not a suitable host discipline for knowledge-based approaches. In Archaeology and the Application of Artificial Intelligence: case studies on use-wear analysis of prehistoric flint tools, Dr. M. H. van den Dries sets out to validate this negative conclusion. She states that since most archaeological applications were mere prototypes and have never been subjected to objective tests, there is hardly any ground for this rather radical inference. In order to ground her conclusion objectively, Van den Dries has built two applications, an expert system and a neural network. She used use-wear analysis of prehistoric tools as the application area. The main objective of the project was to develop

*Contd...*

Notes

a practical training tool for students. As Van den Dries' aim was to demonstrate the practical applicability of both applications, they were exposed to two objective tests in which replicated stone tools as well as prehistoric artefacts were involved. In one test both experience use-wear analysts and students participated. The outcome of this trail has been compared with the results of all other blind tests that human use-wear analysts have carried out hitherto. An important conclusion of Van den Dries' study is that both applications perform well, but that the expert system is better equipped for educational tasks, while the neural network approach mainly suits research purposes. Therefore, the expert system application, called WAVES, has been made operational. It already supports students of several archaeology departments around the world in learning use-wear analysis. Based on her findings Van den en Dries subsequently advises archaeologists to exploit better the merits that artificial intelligence offers them, because it is a means to record unique and valuable expert knowledge, to obtain objective analysis results and to democratize archaeological knowledge.

**Questions:**

1. What are the application areas of AI ?
2. What is WAVES ?

## 1.5 Summary

- AI is a branch of science which deals with helping machines find solution to complex solutions.
- AI is generally associated with computer science but it has many important links with other fields such as Mathematics, Psychology and Biology etc.
- It is more general than the "laws of thought" approach, because correct inference is only a useful mechanism for achieving rationality, and not a necessary one. Second, it is more amenable to scientific development than approaches based on human behavior or human thought, because the standard of rationality is clearly defined and completely general.

## 1.6 Keywords

***Artificial Intelligence:*** Artificial Intelligence is one of the newest disciplines. It was formally initiated in 1956, when the name was coined, although at that point work had been under way for about five years.

***Expert System*:** An expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, like an expert, and not by following the procedure of a developer as is the case in conventional programming.

***Natural Language:*** A natural language (or ordinary language) is any language which arises in an unpremeditated fashion as the result of the innate facility for language possessed by the human intellect. A natural language is typically used for communication, and may be spoken, signed, or written.

## 1.7 Review Questions

1. Explain artificial intelligence with an example.
2. What are the advantages of AI?
3. How does AI help in solving complex problems of computer science?

Notes

4. Discuss the role of AI in police interrogation.
5. Explain the working of any expert systems.
6. What are the different techniques used in AI?
7. Can AI software help in determining human behavior?

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | True |
| 3. | False | 4. | True |
| 5. | True | 6. | True |
| 7. | False | 8. | True |
| 9. | True | 10. | False |
| 11. | True | 12. | True |
| 13. | true | | |

## 1.8 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie).* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

aima.cs.berkeley.edu/

http://interests.caes.uga.edu/eai/ai.html

http://www.ucs.louisiana.edu/~isb9112/dept/phil341/wisai/WhatisAI.html

http://www-formal.stanford.edu/jmc/whatisai/

https://www.ai-class.com/

# Unit 2: Knowledge

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Discuss the general concepts of Knowledge
- Explain the definition and importance of Knowledge
- Describe the concept of Knowledge-based Systems (KBS)

## Introduction

In the previous unit, we dealt with the meaning of Artificial intelligence, its goals and importance, the early work and common techniques used in Artificial Intelligence. Knowledge is a familiarity with someone or something, which can include facts, information, descriptions or skills acquired through experience or education. It can refer to the theoretical or practical understanding of a subject. It can be implicit (as with practical skill or expertise) or explicit (as with the theoretical understanding of a subject); it can be more or less formal or systematic. In philosophy, the study of knowledge is called epistemology; the philosopher Plato famously defined knowledge as

Notes

"justified true belief." However, no single agreed upon definition of knowledge exists, though there are numerous theories to explain it.

The following quote from Bertrand Russell's "Theory of Knowledge" illustrates the difficulty in defining knowledge: "The question how knowledge should be defined is perhaps the most important and difficult of the three with which we shall deal. This may seem surprising: at first sight it might be thought that knowledge might be defined as belief which is in agreement with the facts. The trouble is that no one knows what a belief is, no one knows what a fact is, and no one knows what sort of agreement between them would make a belief true. Let us begin with belief." The definition of knowledge is a matter of ongoing debate among philosophers in the field of epistemology. The classical definition, described but not ultimately endorsed by Plato specifies that a statement must meet three criteria in order to be considered knowledge: it must be justified, true, and believed. Some claim that these conditions are not sufficient, as Gettier case examples allegedly demonstrate.

There are a number of alternatives proposed, including Robert Nozick's arguments for a requirement that knowledge 'tracks the truth' and Simon Blackburn's additional requirement that we do not want to say that those who meet any of these conditions 'through a defect, flaw, or failure' have knowledge. Richard Kirkham suggests that our definition of knowledge requires that the evidence for the belief necessitates its truth. In this unit, you will be able to understand the concepts of knowledge progression and model, importance of knowledge, characteristics and structure of Characteristics of KBS.

## 2.1 General Concepts of Knowledge

The theory of knowledge and creativity is an important department of philosophy. It arose historically with philosophy, as its core, around which everything else was built. This department of philosophy considers a wide range of problems: the relationship between knowledge and reality, its sources and driving forces, its forms and levels, the principles and laws of cognitive activity, and the trends of its development. Philosophy analyses the criteria of the authenticity of knowledge, its veracity, and also the causes of error, the problems of the practical application of knowledge.

As selective reflection of the world cognition expresses the highest creative aspirations of human reason and constitutes the crown of human achievement. Throughout the millennia of its development humanity has travelled a long road, from the primitive and limited to an increasingly profound and comprehensive understanding of the essence of existence.

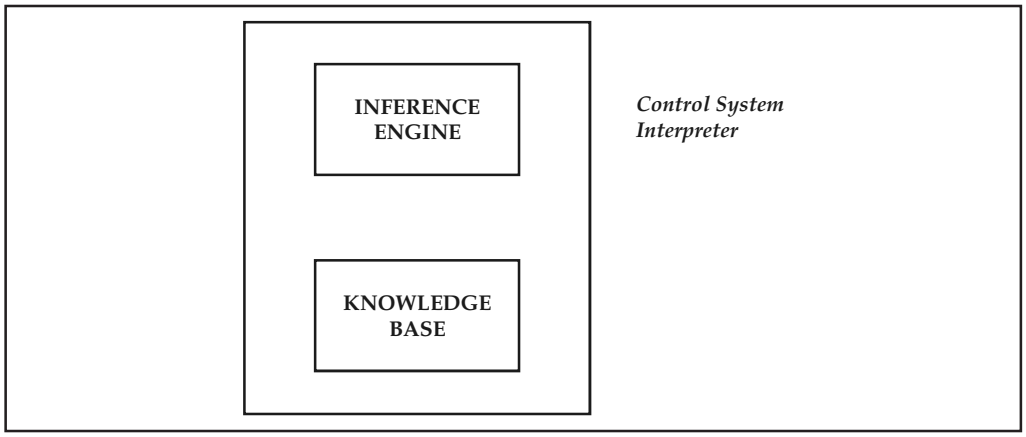
*Caution* This difficult path has led us to the discovery of innumerable facts, properties and laws of nature, of social life and man himself, to the building of an extremely complex and almost unencompassable scientific picture of the world, to the highly sophisticated sphere of art, to the achievements of modern technology.

Humanity has always striven to acquire new knowledge. The process of mastering the secrets of existence continues unceasingly and its vector is oriented on the infinite vistas of the future. The pace and scale of cognitive activity are constantly increasing. Every day is marked by intellectual advances in a constant quest, which ever more widely and vividly illuminates the remote horizons of the as yet invisible. We are deluged with new discoveries.

The path travelled by science convinces us that the possibilities of human cognition are limitless. Our reason perceives the laws of the universe in order to bring them under man's control, in

Notes

order to refashion the world in the interests of man and society. Human knowledge is a highly complex system, a social memory whose wealth is passed on from generation to generation by means of social heredity.




## 2.1.1 Knowledge Progression

The concept can be understood by the following figure:

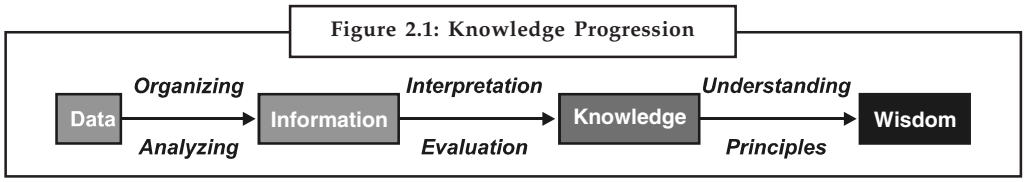



**Figure 2.1: Knowledge Progression**

***Data:*** Data is viewed as collection of disconnected facts.

*Example:* It is raining.

***Information*** emerges when relationships among facts are established and understood;

Provides answers to "who", "what", "where", and "when".

*Example:* The temperature dropped 15 degrees and then it started raining.

***Knowledge*** emerges when relationships among patterns are identified and understood;

Provides answers as "how".

*Example:* If the humidity is very high and the temperature drops substantially, then the atmosphere is unlikely to hold the moisture, so it rains.

***Wisdom*** is the pinnacle of understanding, uncovers the principles of relationships that describe patterns;

Provides answers as "why".

*Example:* Encompasses understanding of all the interactions that happen between raining, evaporation, air currents, temperature gradients, changes, and raining.

Notes

## 2.1.2 Knowledge Model

Knowledge modeling is a process of creating a computer interpretable model of knowledge or standard specifications about a kind of process and/or about a kind of facility or product. The resulting knowledge model can only be computer interpretable when it is expressed in some knowledge representation language or data structure that enables the knowledge to be interpreted by software and to be stored in a database or data exchange file.

Knowledge-based engineering or knowledge-aided design is a process of computer-aided usage of such knowledge models for the design of products, facilities or processes. The design of products or facilities then uses the knowledge model to guide the creation of the facility or product that need to be designed. In other words, it used knowledge about a kind of object to create a product model of an (imaginary) individual object. Similarly, the design of a particular process implies the creation of a process model, which design activity can be guided by the knowledge that is contained in a knowledge model about such a kind of process. The resulting process model, product model or facility model is typically also stored in a database.

Usually, the knowledge representation language only allows to represent knowledge (about kinds of things), whereas another language or data structure is required to represent and store the information models about individual things.

*Did u know?* If the knowledge representation language enables to express both, then the knowledge model and the information model can be expressed in the same language (or data structure).

The basis of a knowledge model of an assembly physical object is a decomposition structure that specifies the components of the assembly and possible the sub-components of the components.

*Example:* Knowledge about a compressor system includes that a compressor system consists of a compressor, a lubrication system, etc., whereas a lubrication system consists of a pump system, etc. Assume that this knowledge is expressed in a knowledge representation language that expresses knowledge as a collection of relations between two kinds of things, whereas in that language a relation type is defined that is called <shall have as part a>. Then a part of a knowledge model about a compressor system will consist of the following expressions of knowledge facts:

- compressor system shall have as part a compressor
- compressor system shall have as part a lubrication system
- lubrication system shall have as part a pump system

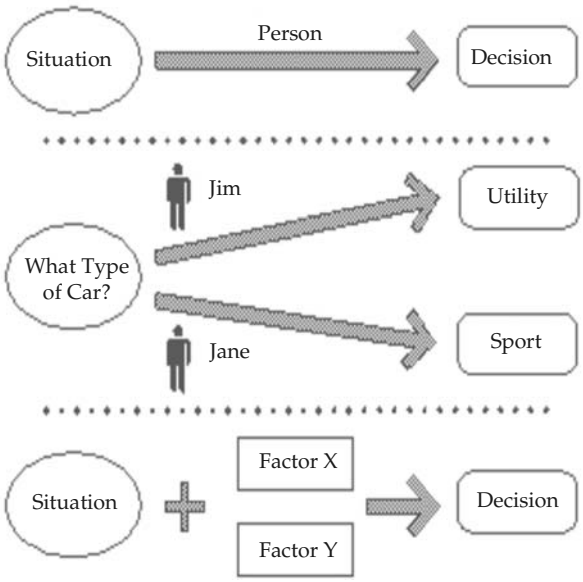
In everyday situations, people make a variety of decisions to act upon. In turn, these decisions vary based on one's preferences, objectives and habits. The following example, Figure 2.2 – Situational Effects, highlights how gender and age play a role in the decision-making process.

As such, many models, like the example of Jim and Jane, can only be executed after having a profile assigned. A profile is defined as the personnel interpretation of inputs to a model.

KCM incorporate the quantitative and qualitative use of information, and processes tangible and intangible attributes that contribute to end result. The bridging together of quantitative and qualitative methods enables KCM to incorporate subjectivity, which is the main differentiator between information and knowledge.

Notes

**Figure 2.2: Example of Knowledge-based System**




## Self Assessment

State whether the following statements are true or false:

1. The theory of knowledge and creativity is an important department of philosophy.
2. Humanity has always striven to acquire old knowledge.
3. KCM incorporate the quantitative and qualitative use of information.
4. The basis of a knowledge model of an assembly physical object is a decomposition structure.
5. The path travelled by science convinces us that the possibilities of human cognition are not limitless.

## 2.2 Definition and Importance of Knowledge

Knowledge encompasses the implicit and explicit restrictions placed upon objects (entities), operations, and relationships along with general and specific heuristics and inference procedures involved in the situation being modeled.

Knowledge is to represent reality in thought or experience the way it really is on the basis of adequate grounds. Knowledge is a description of the world. It determines a system's competence by what it knows.

*Notes* To know something is to think of or experience it as it really is on a solid basis of evidence, experience, intuition and so forth. Little can be said in general about what counts as "adequate grounds." The best one can do is to start with specific cases of knowledge and its absence in, say, art, chemistry, memory, scripture and logic, and formulate helpful descriptions of "adequate grounds" accordingly.

Notes

"Adequate grounds" may be a variety of things, including scientific evidence, widespread perception of beauty and excellence in a piece of art, clarity and vividness in recalling something in the past, and so forth. Different areas of knowledge will have different kinds of evidence relevant to those areas, and when the evidence reaches a certain level, it becomes adequate to provide the kind of support one needs to have knowledge of something.

Again, little more can be said in general about what counts as "adequate." Only by focusing carefully on particular cases can "adequate" be clarified.

### 2.2.1 Three Kinds of Knowledge

In addition to these three observations about knowledge, there are three different kinds of knowledge:

**Knowledge by Acquaintance**

This happens when we are directly aware of something; e.g. when we see an apple directly before must or pay attention to our inner feelings, we know these things by acquaintance. One does not need a concept of an apple or knowledge of how to use the word "apple" in English to have knowledge by acquaintance with an apple. A baby can see and recognize an apple without having the relevant concept or linguistic skills. Knowledge by acquaintance is sometimes called "simple seeing"—being directly aware of something.

**Propositional Knowledge**

This is knowledge that an entire proposition is true. For example, knowledge that "the object there is an apple" requires having a concept of an apple and knowing that the object under consideration satisfies the concept. Propositional knowledge is justified true belief; it believes something that is true on the basis of adequate grounds.

**Know-how**

This is the ability to do certain things; e.g. to use apples for certain purposes. We may distinguish mere know-how from genuine know-how or skill. The latter is know-how based on knowledge and insight and is characteristic of skilled practitioners in some field. Mere know-how is the ability to engage in the correct behavioural movements, say, by following the steps in a manual, with little or no knowledge of why one is performing these movements.

### 2.2.2 Uses of Knowledge

It's the information age, as the popular magazines keep reminding us. Our ability to generate, classify, collect, and exploit data has grown exponentially with the advent of computers and other technologies.

But how is this information to be used? After the supermarkets have scanned our rewards cards to gather data on our buying habits, and the Web site purveyors have posted gigabytes of facts, and the database services have made available every article written in the last 10 years on every conceivable subject, what do we really know?

The Markkula Center for Applied Ethics has been grappling with these questions as part of larger preparations for the conference "Ethics and Technology: Access, Accountability, and Regulation," held at Santa Clara University June 5 and 6.

Notes

These issues in ethics reflects those discussions, exploring the uses and limits of knowledge from several perspectives. "Little Brother is watching You" describes the potential for employers to read employee e-mail and monitor Web site browsing, and it lays out the pros and cons of such electronic surveillance.

"When What We Know Outstrips What We Can Do" looks at the issue in a medical context. How can ethics guide us as science increases our ability to diagnose genetic illnesses for which no cure exists? To what end should the growing body of information about the human genome be put?

Our Thinking Ethically piece argues that it is not enough simply to fill students' heads with knowledge. "Like a Bear Robbed of Her Cubs" explores how educating students for compassion helps them put their knowledge to use.

Among our regular features, we have "The Case of the Million-Dollar Decision," which examines the ethical implications for companies expanding into foreign markets where payoffs are an accepted part of doing business.

### 2.2.3 Knowledge-base

A knowledge-base is a database used for knowledge sharing and management. It promotes the collection, organization and retrieval of knowledge. Many knowledge-bases are structured around artificial intelligence which not only store data but find solutions for further problems using data from previous experience stored as part of the knowledge-base. Knowledge management systems depend on data management technologies ranging from relational databases to data warehouses.

A knowledge-base is not merely a space for data storage, but can be an artificial intelligence tool for delivering intelligent decisions. Various knowledge representation techniques, including frames and scripts, represent knowledge. The services offered are explanation, reasoning and intelligent decision support. Knowledge-based Computer-aided Systems Engineering (KB-CASE) tools assist designers by providing suggestions and solutions, thereby helping to investigate the results of design decisions. The knowledge-base analysis and design allows users to frame knowledge-bases, from which informative decisions are made.

*Did u know?* Two major types of knowledge-bases are human readable and machine readable. Human readable knowledge-bases enable people to access and use the knowledge. They store help documents, manuals, troubleshooting information and frequently answered questions. They can be interactive and lead users to solutions to problems they have, but rely on the user providing information to guide the process.

Machine readable knowledge-bases store knowledge, but only in system readable forms. Solutions are offered based upon automated deductive reasoning and are not as interactive as this relies on query systems that have software that can respond to the knowledge-base to narrow down a solution. This means that machine readable knowledge-base information shared to other machines is usually linear and is limited in interactivity, unlike the human interaction which is query based.

*Caution* Knowledge from any text must be well refined for applying the concept of AI.

*Task* Visit www.knowledge.com for various issues involved with knowledge and AI.

Notes

### Self Assessment

State whether the following statements are true or false:

6. Knowledge determines a system's competence by what it knows.
7. Different areas of knowledge will have same kinds of evidence relevant to those areas, and when the evidence reaches a certain level.
8. Propositional Knowledge is knowledge that an entire proposition is false.
9. Machine readable knowledge-bases store knowledge, but only in system readable forms.
10. Solutions are offered based upon automated deductive.

## 2.3 Knowledge-based Systems (KBS)

A knowledge-based system is a computer program that reasons and uses knowledge to solve complex problems. Knowledge is acquired and represented using various knowledge representation techniques rules, frames and scripts. The basic advantages offered by such system are documentation of knowledge, intelligent decision support, self learning, reasoning and explanation. Knowledge-based systems are systems based on the methods and techniques of Artificial Intelligence. Their core components are:

- Knowledge base
- Acquisition mechanisms
- Inference mechanisms

KBS Consist of following things:

- ***Symbolic:*** It incorporates knowledge that is symbolic [as well as numeric].
- ***Heuristic:*** It reasons with judgmental, imprecise, and qualitative knowledge as well as with formal knowledge of established theories.
- ***Transparent:*** Its knowledge is simply and explicitly represented in terms familiar to specialists, and is separate from its inference procedures. It provides explanations of its line of reasoning and answers to queries about its knowledge.
- ***Flexible:*** It is incrementally re-finable and extensible. More details can be specified to refine its performance; more concepts and links among concepts can be specified to broaden its range of applicability.

It is an expert system if it provides expert-level solutions. The power lies in task-specific knowledge.

### 2.3.1 Characteristics of KBS

Following are the characteristics of the knowledge-based systems:

1. Expand the knowledge-base for the domain of interest.
2. Support for heuristics analysis.
3. Application of search techniques.
4. It is having capability to infer new knowledge from existing knowledge.

Notes

5. Use the symbol processing approach.

6. Ability to explain its own reasoning.

The KBS can help acting as intelligent assistants to human experts or, in certain cases, even replacing the human experts.

### 2.3.2 Structure of a Knowledge-based System

There are three principle roles for those who work with KBS:

(i) ***End user:*** Doctor, factory manager, person of the street, etc.

(ii) ***Knowledge Engineer:*** The person who designs, classifies, organizes, builds a representation of the knowledge)

(iii) ***Constructor of the Tools for the KBS:*** Typically a programmer

The major components of an Expert System are high level Knowledge-base, Inference Engine and User Interface.

***Knowledge-base:*** An expert system can give intelligent answers to sufficient knowledge about the field. The component of the 'expert system' that contains the system's knowledge is called the knowledge-base. It is a vital component of KBS. The knowledge-base consists of the declarative knowledge and procedural knowledge.

Declarative knowledge consists of facts about objects, events and situations. Procedural knowledge consists of courses of action. Knowledge representation is a process of putting the knowledge into the systems knowledge-base in the form of facts by using the above two types while KBS uses reasoning to draw the conclusions from stored facts.

***Inference Engine:*** If the system has knowledge, it must be capable of using the knowledge in appropriate manner. Systems must know how and when to apply the knowledge, i.e. Inference Engine works as a control programmed to decide the direction of the search in KBS.

The KBS uses different types of search techniques to find the solutions of given problems. Search is the name given to the process of shifting through alternative solutions to reach the Goal state. The search is carried through search space. Problem is moved from Initial State to Goal State through state space. The Inference Engine 'decides' which heuristic search techniques are used to determine and how the rules in the knowledge-base are to be applied to the problem. Inference Engine is independent of the knowledge-base.

***User Interface:*** User must have communicated with the system. KBS helps its user to communicate with it is known as 'user interface' in form of bi-directional communication. The system should be able to ask much information to arrive at a solution or the user want to know the reasoning about the facts. Thus, it is important to have a user interface that can be used by common people.

### 2.3.3 KBS Development

The development of a KBS may require a team of several people working together.

There are two types of people involved in the development of the expert system.

1. ***Domain Experts:*** Domain Experts provide the information for the Knowledge-base.

2. ***Knowledge Engineers:*** Knowledge Engineer develops the Knowledge-based System.

Notes

The following are the different stages in the development of KBS:

- Identification
- Conceptualization
- Formalization
- Implementation

### 2.3.4 Testing

Testing is the process in which the knowledge-base systems are checked. Following are the phases in the testing process:

***Identification:*** In this phase, Knowledge Engineer and Domain Expert work together closely to describe the problem that the KBS expected to solve. Such interactive procedure is typical of the entire KBS development process. Additional resources, such as other experts and knowledge engineers and reference journals are also identified in the identification stage.

***Conceptualization:*** This stage involves analyzing the problem. Knowledge Engineer represents graphical representation of the relationship between the objects and the process in the problem domain. The problem is decomposed in to sub-problems and their interrelationships are properly conceptualized. Then the identification stage is revised again. Such interactive process can occur in any stage of development.

***Formalization:*** Both identification and conceptualization are concerned on understanding the problem. The process of formalization means that the problem is connected to its KBS, by analyzing the relation mentioned in conceptualization. Knowledge Engineer selects the development techniques that are appropriate to required KBS. It must be familiar with different types of AI techniques, i.e. heuristic search techniques and knowledge representation mechanisms that are used in the development of KBS.

***Implementation:*** During the implementation stage of KBS, Knowledge Engineer determines whether correct techniques were chosen or not. Otherwise the knowledge engineer has tore formalize the concepts or use new development tools.

***Testing:*** Testing provides an opportunity to the knowledge engineer to identify the strengths and weaknesses of the system that will lead to identify and see whether any modifications are required.

This structure of Expert Systems is most closely matched by the structure of logical programming (its computational model). In a logic programming language such as LISP and PROLOG, PROLOG statements are relations of a restricted form called "Clauses" and the execution of such program is a suitably controlled logic deduction from the Clauses forming the program. A Clause is a well formed Formula consisting of Conjunction and Disjunction of Literals.

The following logic program for family has three Conditions of four Clauses:

- Father (Bill, John)
- Father (John, Tom)
- Grandfather (X, Z): father (X, Y), mother (Y, Z)
- Grandfather (X, Z): father (X, Y), father (Y, Z)

The first two clauses define that Bill is the father of John, second two clauses use the variables X, Y and Z to represent (express) the rule that if X is the grandfather of Z, if X is the father of Y and

Notes

Y is either the mother or father of Z. Such a program can be asked a range of questions—from "Is John, the father of Tom?" [Father (John, Tom)?] To "Is there any A who is the grandfather of C?" [Grandfather (A, C)?] .

The possible operation of computer based on logic is illustrated in the following using the family tree program. Execution of, for example, "Grandfather (Bill, R)?" will match each "Grandfather ()"Clause.

Grandfather (X = Bill, Z = R): father (Bill, Y), mother (Y, R).

Grandfather (X = Bill, Z = R): father (Bill, Y), father (Y, R).

Both clauses will attempt in parallel to satisfy their Goals, such a concept is called OR – Parallelism. The first clause will fail being unable to satisfy its goal, search will continues to the second clause, i.e. called OR – Parallelism.

The first clause will fail being unable to satisfy the "Mother ()" goal from the program. The second goal has "Father ()", "Mother ()", which is an attempt to solve in parallel: such a concept is called AND parallelism. The latter concept involves Pattern Matching methods and substitution to satisfy both the individual goals.

Grandfather (X = Bill, Z = R)

: father (Bill, Y), father (Y, R).

: father (Bill, Y = John), father (Y = Bill, R = John).

And the Overall Consistency

: father (Bill, Y = John), father (Y = John, R = Tom).

Computers Organization Supporting Expert Systems are a highly microprogrammed (Control Flow Based). PROLOG machines analogous to current LISP machines through we can expect a number of such designs in the near feature. The PROLOG machines are not TRUE Logic Machines, just as LISP Machines are not considered reduction machines linked by a Common Logic Machine language and architecture.

### 2.3.5 Why Build a Knowledge-based System?

Following are the reasons to build knowledge-based systems:

- To decrease cost or increase quality of goods and services.
- Magnify availability of expertise.
- Provide expertise to less experienced personnel.
- Avoid delays when expertise is needed.
- Provide expertise in locations where it is not available.
- Fuse different sources of knowledge.
- Encode corporate knowledge.
- Provide consistency and availability over time.
- Automate some routine decision-making or bookkeeping tasks.
- Keep records of decisions and actions.
- Provide a reliable database for later analysis.

Notes

### Self Assessment

State whether the following statements are true or false:

11. Knowledge-based systems are systems which are not based on the methods and techniques of Artificial Intelligence.
12. The development of a KBS may require a team of several people working together.
13. Testing is the process in which the knowledge-base systems are checked.
14. End User are the person who designs, classifies, organizes, builds a representation of the knowledge.
15. Domain Experts provide the information for the Knowledge-base.

## 2.4 Summary

- Knowledge is a progression that starts with data which is of limited utility.
- Data and information deal with the past; they are based on the gathering of facts and adding context.
- Knowledge encompasses the implicit and explicit restrictions placed upon objects (entities), operations, and relationships along with general and specific heuristics and inference procedures involved in the situation being modelled.
- A knowledge-base is a database used for knowledge sharing and management.
- The Knowledge-based System division carries out research and development in selected sub fields of Artificial Intelligence.
- Knowledge-based System is a computer program designed to act as an expert in specific field of knowledge.

## 2.5 Keywords

***Acquaintance:*** Acquaintance knowledge by acquaintance is obtained through a direct causal (experience-based) interaction between a person and the object that person perceives.

***Heuristic Methods:*** Heuristic method refers to experience-based techniques for problem-solving, learning, and discovery.

***Inferential Knowledge:*** Inferential knowledge is based on reasoning from facts or from other inferential knowledge such as a theory. Such knowledge may or may not be verifiable by observation or testing.

***LISP:*** LISP is a family of computer programming languages with a long history and a distinctive, fully parenthesized Polish prefix notation.

***Procedural Knowledge:*** Also known as imperative knowledge, it is the knowledge exercised in the performance of some task.

***PROLOG:*** PROLOG is a general purpose logic programming language associated with artificial intelligence and computational linguistics.

## 2.6 Review Questions

Notes

1. Explain knowledge with respect to Artificial Intelligence.
2. Why should knowledge be extracted carefully?
3. What are the different types of knowledge?
4. What are the applications of knowledge?
5. How is the KBS established?
6. What are the necessary actions for KBS development?

### Answers: Self Assessment

1. True
2. False
3. True
4. True
5. False
6. True
7. False
8. False
9. True
10. True
11. False
12. True
13. True
14. False
15. True

## 2.7 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://arantxa.ii.uam.es/~modonnel/IC/02_Structure.pdf

http://www.jblearning.com/catalog/9780763776473/

http://www.mels.gouv.qc.ca/progression/index_en.asp

http://www.reidgsmith.com/Knowledge-Based_Systems_-_Concepts_Techniques_Examples_08-May-1985.pdf

http://www.tmrfindia.org/eseries/ebookV1-C1.pdf

Notes

# Unit 3: Representation of Knowledge

**CONTENTS**



## Objectives

After studying this unit, you will be able to:

- Identify Knowledge Representation Techniques
- Discuss the concept of Knowledge Organization
- Describe Knowledge Manipulation
- Explain Knowledge Acquisition

## Introduction

Notes

In the previous unit, we dealt with the concepts of knowledge progression and model, importance of knowledge, characteristics and structure of characteristics of KBS.

Knowledge Representation (KR) is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inferencing from those knowledge elements, creating new elements of knowledge. The KR can be made to be independent of the underlying knowledge model or KBS such as a semantic network.

Knowledge Representation (KR) research involves analysis of how to reason accurately and effectively and how best to use a set of symbols to represent a set of facts within a knowledge domain. A symbol vocabulary and a system of logic are combined to enable inferences about elements in the KR to create new KR sentences. Logic is used to supply formal semantics of how reasoning functions should be applied to the symbols in the KR system. Logic is also used to define how operators can process and reshape the knowledge. Examples of operators and operations include negation, conjunction, adverbs, adjectives, quantifiers and modal operators. The logic is interpretation theory. These elements – symbols, operators, and interpretation theory – are what give sequences of symbols meaning within a KR.

A key parameter in choosing or creating a KR is its expressivity. The more expressive a KR, the easier and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR. However, more expressive languages are likely to require more complex logic and algorithms to construct equivalent inferences. A highly expressive KR is also less likely to be complete and consistent. Less expressive KRs may be both complete and consistent. Autoepistemic temporal modal logic is a highly expressive KR system, encompassing meaningful chunks of knowledge with brief, simple symbol sequences (sentences). Propositional logic is much less expressive but highly consistent and complete and can efficiently produce inferences with minimal algorithm complexity. Nonetheless, only the limitations of an underlying knowledge base affect the ease with which inferences may ultimately be made (once the appropriate KR has been found). This is because a knowledge set may be exported from a knowledge model or KBS into different KRs, with different degrees of expressiveness, completeness, and consistency. If a particular KR is inadequate in some way, that set of problematic KR elements may be transformed by importing them into a KBS, modified and operated on to eliminate the problematic elements or augmented with additional knowledge imported from other sources, and then exported into a different, more appropriate KR. In this unit, you will understand the concepts of knowledge representation techniques along with the knowledge organization, manipulation and acquisition.

## 3.1 Knowledge Representation Techniques

In applying KR systems to practical problems, the complexity of the problem may exceed the resource constraints or the capabilities of the KR system. Recent developments in KR include the concept of the Semantic Web, and development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

There are several KR techniques such as frames, rules, tagging, and semantic networks which originated in cognitive science. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to facilitate reasoning, inferencing, or drawing conclusions. A good KR must be both declarative and procedural knowledge. What is knowledge

Notes

representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand:

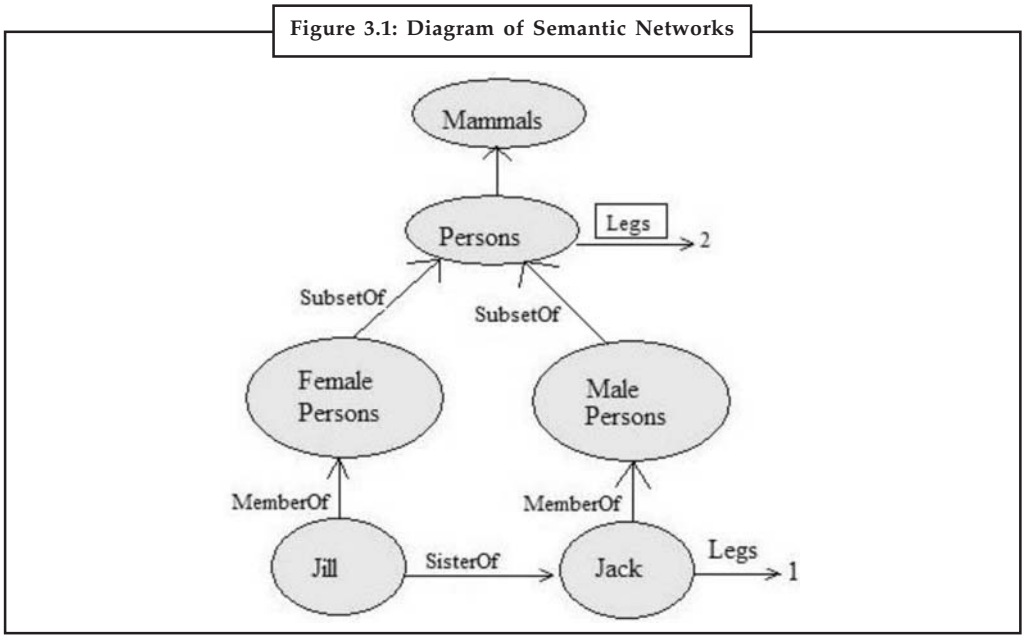
- KR is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world."

### 3.1.1 Semantic Nets

A semantic net (or semantic network) is a knowledge representation technique used for propositional information. So it is also called a propositional net. Semantic nets convey meaning. Semantic nets are two dimensional representations of knowledge. Mathematically, a semantic net can be defined as a labeled directed graph.

Semantic nets consist of nodes, links (edges) and link labels. In the semantic network diagram, nodes appear as circles or ellipses or rectangles to represent objects such as physical objects, concepts or situations. Links appear as arrows to express the relationships between objects, and link labels specify particular relations. Relationships provide the basic structure for organizing knowledge. The objects and relations involved need not be so concrete. As nodes are associated with other nodes semantic nets are also referred to as associative nets.

**Figure 3.1: Diagram of Semantic Networks**



Notes

In the Figure 3.1 all the objects are within ovals and connected using labelled arcs. Note that there is a link between Jill and FemalePersons with label MemberOf. Similarly, there is a MemberOf link between Jack and MalePersons and SisterOf link between Jill and Jack. The MemberOf link between Jill and FemalePersons indicates that Jill belongs to the category of female persons.

### Inheritance Reasoning

Unless, there is a specific evidence to the contrary, it is assumed that all members of a class (category) will inherit all the properties of their superclasses. So semantic network allows us to perform inheritance reasoning. For example, Jill inherits the property of having two legs as she belongs to the category of FemalePersons which, in turn, belongs to the category of Persons which has a boxed Legs link with value 2. Semantic nets allows multiple inheritance. So an object can belong to more than one category and a category can be a subset of more than one another category.

### Inverse Links

Semantic network allows a common form of inference known as inverse links. For example, we can have a HasSister link which is the inverse of SisterOf link. The inverse links make the job of inference algorithms much easier to answer queries such as who the sister of Jack is. On discovering that HasSister is the inverse of SisterOf the inference algorithm can follow that link HasSister from Jack to Jill and answer the query.

### Disadvantage of Semantic Nets

One of the drawbacks of semantic network is that the links between the objects represent only binary relations. For example, the sentence Run(ChennaiExpress, Chennai,Bangalore,Today) cannot be asserted directly.

There is no standard definition of link names.

### Advantages of Semantic Nets

Semantic nets have the ability to represent default values for categories. In the figure, Jack has one leg while he is a person and all persons have two legs. So persons have two legs has only default status which can be overridden by a specific value.

- Semantic nets convey some meaning in a transparent manner.
- Semantic nets are simple and easy to understand.
- Semantic nets are easy to translate into PROLOG.

## 3.1.2 Frames and Scripts

Two other popular knowledge representation formalisms are frames and scripts. Although, these were developed independently (both originating in the early 1970s), and are different in important ways, they have sufficient similarities to be considered together. One influential proponent of frame-based systems is Marvin Minsky (1975); a champion of script-based systems is Roger Schank (see Schank and Abelson, 1977). The key idea involved in both frames and scripts is that our knowledge of concepts, events, and situations is organized around expectations of key features of those situations.

Notes

*Example:* Consider a stereotypical situation, such as going to hear a lecture. One's knowledge of what might go on during such an event is based on assumptions. For instance, it can be assumed that the person who actually delivers the lecture is likely to be identical with the person advertised; that the lecturer's actual time of arrival is not more than a few minutes after the advertised start; that the duration of the lecture is unlikely to exceed an hour and a half at the maximum; and so on. These and other expectations can be encoded in a generic 'lecture frame' to be modified by what actually occurs during a specific lecture. This frame will include various *slots*, where specific values can be entered to describe the occasion under discussion. For example, a lecture frame may include slots for 'room location', 'start time', 'finish time', and so on.

Scripts were developed in the early AI work by Roger Schank, Robert P. Abelson and their research group, and are a method of representing procedural knowledge. They are very much like frames, except the values that fill the slots must be ordered. A script is a structured representation describing a stereotyped sequence of events in a particular context. Scripts are used in natural language understanding systems to organize a knowledge base in terms of the situations that the system should understand.

### 3.1.3 Logic

#### Propositional Logic

The following is a brief summary of propositional logic, intended only as a reminder to those who have taken a course in elementary logic.

#### Language

Propositional logic is the logic of propositional formulas. Propositional formulas are constructed from a set var of elementary or 'atomic' propositional variables p, q, and so on, with the connectives:

(negation, 'not'), ^ (conjunction, 'and'), _ (disjunction, 'or'), ! (implication, 'if . . . then'), and $ (equivalence, 'if and only if'). If ' and are formulas, then so are:' and:, ('^), ('_), (' !), and (' $). So p is a formula,:(p ^ q) and q _ (q ^:(r !:p)) are formulas, but pq is not a formula, and neither are p ^ q ! and p: _ r. We add the simple symbol ? which is called the falsum. We write this definition in Backus-Naur Form (BNF) notation, as follows:

[Lprop] '::= p j ? j:' j (' ^ ') j (' _ ') j (' ! ') j (' $ ')

This means that a formula ' can be an atom p, the falsum ?, or a complex expression of the other forms, whereby its sub-expressions themselves must be formulas. The language of propositional logic is called Lprop.

Brackets are important to ensure that formulas are unambiguous. The sentence p _ q ^ r could be understood to mean either (p _ q) ^ r or p _ (q ^ r), which are quite different insofar as their meaning is concerned. We omit the outside brackets, so we do not write ((p _ q) ^ r).

The symbols ';; _;::: are formula variables. So, if it is claimed that the formula ' _:' is a tautology, it means that every propositional formula of that form is a tautology. This includes p _:p, (p !:q) _:(p !:q) and any other such formula. In a similar way, we formulate axiom schemata and inference rules by means of formula variables. If ' ! (! ') is an axiom scheme, then every formula of that form is an axiom, such as (p ^ q) ! (:q ! (p ^ q)). And an inference rule that allows us to infer ' _ from ' allows us to infer p _ (q $ r) from p, and so on.

**Truth Values and Truth Tables**

**Notes**

In propositional logic, the models are (truth) valuations. A truth valuation determines which truth value the atomic propositions get. It is a function v: var ! f0; 1g, where var is the non-empty set of atomic propositional variables. A proposition is true if it has the value 1, and false if it has the value 0. The truth value of falsum is always 0, so v(?) = 0 for every valuation v.

Whether a complex propositional formula is true in a given model (valuation) can be calculated by means of truth functions, that take truth values as input and give truth values as output. To each logical connective corresponds a truth function. They are defined as follows:

f:(1) = 0 f:(0) = 1

f^(1; 1) = 1 f^(1; 0) = 0 f^(0; 1) = 0 f^(0; 0) = 0

f_(1; 1) = 1 f_(1; 0) = 1 f_(0; 1) = 1 f_(0; 0) = 0

f!(1; 1) = 1 f!(1; 0) = 0 f!(0; 1) = 1 f!(0; 0) = 1

f$(1; 1) = 1 f$(1; 0) = 0 f$(0; 1) = 0 f$(0; 0) = 1

*Example:* If v(p) = 1 and v(q) = 0, then the propositional formula (p ! q) ^:q is false. To see that this is the case, firstly, we calculate the truth value of p ! q, which is f!(v(p); v(q)) = f!(1; 0) = 0, so p ! q is false. Secondly, we calculate the truth value of :q, which is f:(v(q)) = f:(0) = 1, so:q is true.

A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be atoms (in this case, these atoms are treated as constants), numbers, variables or lists. Arguments are separated by commas.

If we consider the arguments in a fact to be objects, then the predicate of the fact describes a property of the objects. In a Prolog program, a presence of a fact indicates a statement that is true. An absence of a fact indicates a statement that is not true.

### 3.1.4 Facts and Rules

A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true. It consists of two parts. The first part is similar to a fact (a predicate with arguments). The second part consists of other clauses (facts or rules which are separated by commas) which must all be true for the rule itself to be true. These two parts are separated by ":-". You may interpret this operator as "if" in English.

A program describes the relationships of the members in a family

```
father(jack, susan). /* Fact 1 */
father(jack, ray).                                  /* Fact 2 */
father(david, liza).                                /* Fact 3 */
father(david, john).                                /* Fact 4 */
father(john, peter).                                /* Fact 5 */
father(john, mary).                                 /* Fact 6 */
mother(karen, susan).                               /* Fact 7 */
mother(karen, ray).                                 /* Fact 8 */
mother(amy, liza).                                  /* Fact 9 */
mother(amy, john).                                 /* Fact 10 */
mother(susan, peter).                              /* Fact 11 */
mother(susan, mary).                               /* Fact 12 */

parent(X, Y):- father(X, Y).                        /* Rule 1 */
parent(X, Y):- mother(X, Y).                        /* Rule 2 */
grandfather(X, Y):- father(X, Z), parent(Z, Y).     /* Rule 3 */
```

Notes

```
grandmother(X, Y):- mother(X, Z), parent(Z, Y).          /* Rule 4 */
grandparent(X, Y):- parent(X, Z), parent(Z, Y).          /* Rule 5 */
yeye(X, Y):- father(X, Z), father(Z, Y).                 /* Rule 6 */
mama(X, Y):- mother(X, Z), father(Z, Y).                 /* Rule 7 */
gunggung(X, Y):- father(X, Z), mother(Z, Y).             /* Rule 8 */
popo(X, Y):- mother(X, Z), mother(Z, Y).                 /* Rule 9 */
```

Take Rule 3 as an example. It means that "`grandfather(X, Y)`" is true if both "`father(X, Z)`" and "`parent(Z, X)`" are true. The comma between the two conditions can be considered as a logical-AND operator.

You may see that both Rules 1 and 2 start with "`parent(X, Y)`". When will "`parent(X, Y)`" be true? The answer is any one of these two rules is true. This means that "`parent(X, Y)`" is true when "`father(X, Y)`" is true, or "`mother(X, Y)`" is true.

### 3.1.5 Statistics

Statistics is the study of the collection, organization, analysis, interpretation, and presentation of data. It deals with all aspects of this, including the planning of data collection in terms of the design of surveys and experiments. The word statistics, when referring to the scientific discipline, is singular, as in "Statistics is an art." This should not be confused with the word statistic, referring to a quantity (such as mean or median) calculated from a set of data, whose plural is statistics.

Some consider statistics a mathematical body of science that pertains to the collection, analysis, interpretation or explanation, and presentation of data, while others consider it a branch of mathematics concerned with collecting and interpreting data. Because of its empirical roots and its focus on applications, statistics is usually considered a distinct mathematical science rather than a branch of mathematics.

*Notes* Much of statistics is non-mathematical: ensuring that data collection is undertaken in a way that produces valid conclusions; coding and archiving data so that information is retained and made useful for international comparisons of official statistics; reporting of results and summarised data (tables and graphs) in ways comprehensible to those who must use them; implementing procedures that ensure the privacy of census information.

Statisticians improve data quality by developing specific experiment designs and survey samples. Statistics itself also provides tools for prediction and forecasting the use of data and statistical models. Statistics is applicable to a wide variety of academic disciplines, including natural and social sciences, government, and business. Statistical consultants can help organizations and companies that don't have in-house expertise relevant to their particular questions.

### 3.1.6 Neural Networks

The term "neural network" was traditionally used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus, the term may refer to either biological neural networks, made up of real biological neurons, or artificial neural networks, for solving artificial intelligence problems.

Neural networks are a form of multiprocessor computer system, with

- simple processing elements
- a high degree of interconnection

Notes

- simple scalar messages
- adaptive interaction between elements

A biological neuron may have as many as 10,000 different inputs, and may send its output (the presence or absence of a short-duration spike) to many other neurons. Neurons are wired up in a 3-dimensional pattern.

*Did u know?* Real brains are orders of magnitude more complex than any artificial neural network so far considered.

### 3.1.7 Mixed Approaches

Certainly, all previous mentioned ways can be combined. This combination represents a powerful but complicated solution. The problem is the data exchange between the different methods. Here potent interfaces must be created.

### Self Assessment

State whether the following statements are true or false:

1. The KR cannot be made to be independent of the underlying knowledge model or knowledge base system (KBS).
2. A knowledge representation (KR) is most fundamentally a surrogate.
3. Semantic nets are simple and easy to understand.

## 3.2 Knowledge Organization

Knowledge management has become such a hot topic that it has been dubbed the business mantra of the 1990s (Halal, 1998). It primarily addresses the growing importance of knowledge management for private sector organizations, but clearly knowledge-generating organizations such as federal science management and research agencies can not only benefit from this literature but also play a leadership role in furthering theory and practice in this area. Although, these knowledge-oriented organizations have been in the business of creating and furthering knowledge development, they have not necessarily developed and articulated a systemic approach to knowledge management. This is a critical omission that should be corrected. Of all the management topics of potential relevance to public science organizations, this may be one of the most useful areas to pursue. Knowledge management is central to public science organizations. Although, knowledge management has become a highly prominent topic, the term remains rather ambiguous and controversial, impeding progress in articulating what knowledge management entails and what knowledge-based organizations will look like.

*Caution* Many have questioned whether knowledge management is, or will ever become, a useful concept with practical application; others proclaim it is already the pivotal driver of organizational success and will only become more important in the future. The latter point of view is persuasive, but there is a long way to go in clarifying and articulating the concept of knowledge management.

Notes

The belief that knowledge management is destined to become the key to future economic success is based on the following logic:

(i) Many prominent scholars note that a new economic era, referred to as the knowledge-based economy, is already underway. In this new economy, knowledge is the source of wealth. It is assumed, therefore, that knowledge management will be the new work of organizations.

(ii) Knowledge management represents a logical progression beyond information management. Information technologies, at long last, have demonstrated a notable impact on organizational performance. Many believe that the next generation of information technology/artificial intelligence products will increasingly enable knowledge management, in contrast to information management, and, as such, will have a far bigger impact on organizational performance.

(iii) Knowledge management can also be seen as representing a culmination and integration of many earlier organization development ideas, e.g. total quality, re-engineering, organizational learning, benchmarking, competitive intelligence, innovation, organizational agility, asset management, supply chain management, change management, etc.). It encapsulates these concepts into a larger, more holistic perspective that focuses on effectively creating and applying knowledge.

### 3.2.1 Factors

MYCIN was an early expert system that used artificial intelligence to identify bacteria causing severe infections, such as bacteremia and meningitis, and to recommend antibiotics, with the dosage adjusted for patient's body weight—the name derived from the antibiotics themselves, as many antibiotics have the suffix "-mycin". The Mycin system was also used for the diagnosis of blood clotting diseases.

MYCIN was developed over five or six years in the early 1970s at Stanford University. It was written in Lisp as the doctoral dissertation of Edward Shortliffe under the direction of Bruce Buchanan, Stanley N. Cohen and others. It arose in the laboratory that had created the earlier Dendral expert system.

MYCIN was never actually used in practice but research indicated that it proposed an acceptable therapy in about 69% of cases, which was better than the performance of infectious disease experts who were judged using the same criteria.

MYCIN was never actually used in practice. This wasn't because of any weakness in its performance. As mentioned, in tests it outperformed members of the Stanford medical school faculty. Some observers raised ethical and legal issues related to the use of computers in medicine—if a program gives the wrong diagnosis or recommends the wrong therapy, who should be held responsible? However, the greatest problem, and the reason that MYCIN was not used in routine practice, was the state of technologies for system integration, especially at the time it was developed. MYCIN was a stand-alone system that required a user to enter all relevant information about a patient by typing in response to questions that MYCIN would pose. The program ran on a large time-shared system, available over the early Internet (ARPANet), before personal computers were developed. In the modern era, such a system would be integrated with medical record systems, would extract answers to questions from patient databases, and would be much less dependent on physician entry of information. In the 1970s, a session with MYCIN could easily consume 30 minutes or more—an unrealistic time commitment for a busy clinician.

Notes

### 3.2.2 Chaining

Chaining is an instructional procedure used in behavioral psychology, experimental analysis of behavior and applied behavior analysis. It involves reinforcing individual responses occurring in a sequence to form a complex behavior. It is frequently used for training behavioral sequences (or "chains") that are beyond the current repertoire of the learner.

The chain of responses is broken down into small steps using task analysis. Parts of a chain are referred to as links. The learner's skill level is assessed by an appropriate professional and is then either taught one step at a time while being assisted through the other steps forward or backwards or if the learner already can complete a certain percentage of the steps independently, the remaining steps are all worked on during each trial total task. A verbal stimulus or prompt is used at the beginning of the teaching trial. The stimulus change that occurs between each response becomes the reinforcer for that response as well as the prompt/stimulus for the next response without requiring assistance from the teacher.

*Example:* In purchasing a soda you pull the money out of your pocket and see the money in your hand and then put the money in the machine. Seeing the money in your hand both was the reinforcer for the first response (getting money out of pocket) and was what prompted you to do the next response (putting money in machine).

As small chains become mastered, i.e. are performed consistently following the initial discriminative stimulus prompt, they may be used as links in larger chains. (e.g. teach hand washing, tooth brushing, and showering until mastered and then teach morning hygiene routine which includes the mastered skills). Chaining requires that the teachers present the training skill in the same order each time and is most effective when teachers are delivering the same prompts to the learner. The most common forms of chaining are backward chaining, forward chaining, and total task presentation.

### 3.2.3 Software Architecture

Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. Their definition is:

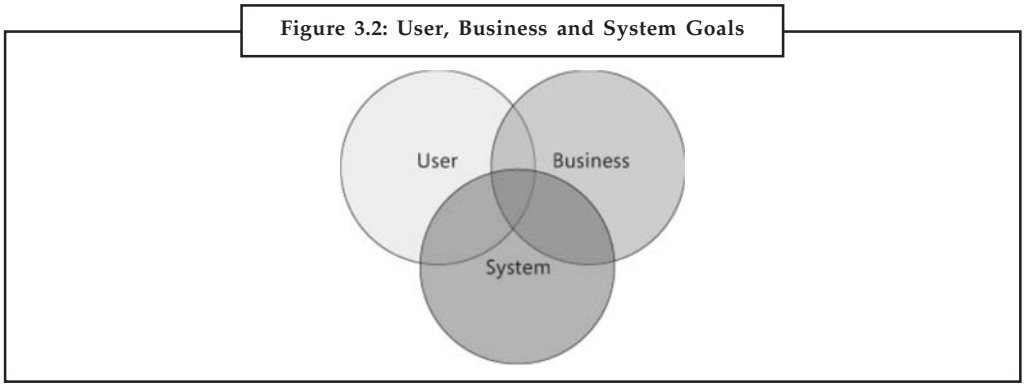
*"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, trade-offs and aesthetic concerns."*

Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

Notes

Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals. For each of these areas, you should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.

**Figure 3.2: User, Business and System Goals**




Trade-offs are likely, and a balance must often be found between competing requirements across these three areas. For example, the overall user experience of the solution is very often a function of the business and the IT infrastructure, and changes in one or the other can significantly affect the resulting user experience. Similarly, changes in the user experience requirements can have significant impact on the business and IT infrastructure requirements. Performance might be a major user and business goal, but the system administrator may not be able to invest in the hardware required to meet that goal 100 percent of the time. A balance point might be to meet the goal only 80 percent of the time.

Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application. The selection of data structures and algorithms or the implementation details of individual components are design concerns. Architecture and design concerns very often overlap. Rather than use hard and fast rules to distinguish between architecture and design, it makes sense to combine these two areas. In some cases, decisions are clearly more architectural in nature. In other cases, the decisions are more about design, and how they help you to realize that architecture.

### 3.2.4 Theories of Knowledge is Represented and Organized

Knowledge acquisition is the process of absorbing and storing new information in memory, the success of which is often gauged by how well the information can later be remembered (retrieved from memory). The process of storing and retrieving information depends heavily on the representation and organization of the information. Moreover, the utility of knowledge can also be influenced by how the information is structured. For example, a bus schedule can be represented in the form of a map or a timetable. On the one hand, a timetable provides quick and easy access to the arrival time for each bus, but does little for finding where a particular stop is situated. On the other hand, a map provides a detailed picture of each bus stop's location, but cannot efficiently communicate bus schedules. Both forms of representation are useful, but it is important to select the representation most appropriate for the task at hand. Similarly, knowledge acquisition can be improved by considering the purpose and function of the desired information.

#### Knowledge Representation and Organization

There are numerous theories of how knowledge is represented and organized in the mind, including rule-based production models, distributed networks, and propositional models.

Notes

However, these theories are all fundamentally based on the concept of semantic networks. A semantic network is a method of representing knowledge as a system of connections between concepts in memory.

### Semantic Networks

According to semantic network models, knowledge is organized based on meaning, such that semantically related concepts are interconnected. Knowledge networks are typically represented as diagrams of nodes (i.e., concepts) and links (i.e., relations). The nodes and links are given numerical weights to represent their strengths in memory. The node representing DOCTOR is strongly related to SCALPEL, whereas NURSE is weakly related to SCALPEL. These link strengths are represented here in terms of line width. Concepts such as DOCTOR and BREAD are more memorable because they are more frequently encountered than concepts such as SCALPEL and CRUST.

Mental excitation, or activation, spreads automatically from one concept to another related concept. For example, thinking of BREAD spreads activation to related concepts, such as BUTTER and CRUST. These concepts are primed, and thus more easily recognized or retrieved from memory. For example, in David Meyer and Roger Schvaneveldt's 1976 study (a typical semantic priming study), a series of words (e.g., BUTTER) and non-words (e.g., BOTTOR) are presented, and participants deter mine whether each item is a word. A word is more quickly recognized if it follows a semantically related word. For example, BUTTER is more quickly recognized as a word if BREAD precedes it, rather than NURSE. This result supports the assumption that semantically related concepts are more strongly connected than unrelated concepts.

A good knowledge representation covers six basic characteristics:

- Coverage, which means the KR covers a breadth and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
- Understandable by humans. KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (Polar bears are bears, which are animals). It should also have simple primitives that combine in complex forms.
- Consistency. If John closed the door, it can also be interpreted as the door was closed by John. By being consistent, the KR can eliminate redundant or conflicting knowledge.
- Efficient
- Easiness for modifying and updating.
- Supports the intelligent activity which uses the knowledge base

To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopedia (e.g. Wikipedia) is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

## 3.2.5 Participants

Rules of inference are syntactical transform rules which one can use to infer a conclusion from a premise to create an argument. A set of rules can be used to infer any valid conclusion if it is complete, while never inferring an invalid conclusion, if it is sound. A sound and complete set of rules need not include every rule in the following list, as in logic, a rule of inference, inference

Notes

rule, or transformation rule is the act of drawing a conclusion based on the form of premises interpreted as a function which takes premises, analyzes their syntax, and returns a conclusion (or conclusions). For example, the rule of inference modus ponens takes two premises, one in the form of "If p then q" and another in the form of "p" and returns the conclusion "q". The rule is valid with respect to the semantics of classical logic (as well as the semantics of many other non-classical logics), in the sense that if the premises are true (under an interpretation) then so is the conclusion.

Typically, a rule of inference preserves truth, a semantic property. In many-valued logic, it preserves a general designation. But a rule of inference's action is purely syntactic, and does not need to preserve any semantic property: any function from sets of formulae to formulae counts as a rule of inference. Usually only rules that are recursive are important; i.e. rules such that there is an effective procedure for determining whether any given formula is the conclusion of a given set of formulae according to the rule. An example of a rule that is not effective in this sense is the infinitary rule.

*Task* Critically examine the importance of chaining in behavioral psychology.

### Self Assessment

State whether the following statements are true or false:

4. Knowledge management represents a logical progression beyond information management.
5. MYCIN is not an expert system.
6. A good knowledge representation covers intelligent activity which uses the knowledge base.

## 3.3 Knowledge Manipulation

The Knowledge Query and Manipulation Language, or KQML, is a language and protocol for communication among software agents and knowledge-based systems. It was developed in the early 1990s part of the DARPA knowledge Sharing Effort, which was aimed at developing techniques for building large-scale knowledge bases which are shareable and reusable. While originally conceived of as an interface to knowledge based systems, it was soon repurposed as an Agent communication language.

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 sub-goals 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.

### 3.3.1 The Turing Test

The Turing test is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of an actual human. In the original illustrative example, a human

Notes

judge engages in a natural language conversation with a human and a machine designed to generate performance indistinguishable from that of a human being. All participants are separated from one another. If the judge cannot reliably tell the machine from the human, the machine is said to have passed the test. The test does not check the ability to give the correct answer; it checks how closely the answer resembles typical human answers. The conversation is limited to a text-only channel such as a computer keyboard and screen so that the result is not dependent on the machine's ability to render words into audio. The test was introduced by Alan Turing in his 1950 paper "Computing Machinery and Intelligence," which opens with the words: "I propose to consider the question, 'Can machines think?'" Since "thinking" is difficult to define, Turing chooses to "replace the question by another, which is closely related to it and is expressed in relatively unambiguous words." Turing's new question is: "Are there imaginable digital computers which would do well in the imitation game?" This question, Turing believed, is one that can actually be answered. In the remainder of the paper, he argued against all the major objections to the proposition that "machines can think".

### Self Assessment

State whether the following statements are true or false:

7. KQML, is a language and protocol for communication among software agents and knowledge-based systems.
8. The manipulations are the computational equivalent of reasoning.
9. The searching and matching operations consume least amount of computation time in AI systems.

## 3.4 Knowledge Acquisition

Knowledge acquisition is the process of extracting, structuring and organizing knowledge from one source, usually human experts, so it can be used in software such as an ES. This is often the major obstacle in building an Expert System (ES).

There are three main topic areas central to knowledge acquisition that require consideration in all ES projects. Firstly, the domain must be evaluated to determine if the type of knowledge in the domain is suitable for an ES. Secondly, the source of expertise must be identified and evaluated to ensure that the specific level of knowledge required by the project is provided. Thirdly, if the major source of expertise is a person, the specific knowledge acquisition techniques and participants need to be identified. An ES attempts to replicate in software the reasoning/pattern-recognition abilities of human experts who are distinctive because of their particular knowledge and specialized intelligence. ES should be heuristic and readily distinguishable from algorithmic programs and databases. Further, ES should be based on expert knowledge, not just competent or skillful behavior.

### Domains

Several domain features are frequently listed for consideration in determining whether an ES is appropriate for a particular problem domain. Several of these caveats relate directly to knowledge acquisition. Firstly, bona fide experts, people with generally acknowledge expertise in the domain, must exist. Secondly, there must be general consensus among experts about the accuracy of solutions in a domain. Thirdly, experts in the domain must be able to communicate the details of their problem solving methods. Fourthly, the domain should be narrow and well defined and solutions within the domain must not require common sense.

**Notes**

Following points are remembered about knowledge acquisition:

I. 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.

II. A knowledge representation language is defined by two aspects:

(i) *Syntax:* The syntax of a language defines which configurations of the components of the language constitute valid sentences.

(ii) *Semantics:* The semantics defines which facts in the world the sentences refer to, and hence the statement about the world that each sentence makes.

III. 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

IV. The requirements of a knowledge representation are as follows:

(i) *Representational Adequacy:* The ability to represent all the different kinds of knowledge that might be needed in that domain.

(ii) *Inferential Adequacy:* The ability to manipulate the representational structures to derive new structures (corresponding to new knowledge) from existing structures.

(iii) *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.

(iv) *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.

V. In practice, the theoretical requirements for good knowledge representations can usually be achieved by dealing appropriately with a number of practical requirements:

(i) The representations need to be complete – so that everything that could possibly need to be represented can easily be represented.

(ii) They must be computable – implementable with standard computing procedures.

(iii) 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.

(iv) They should suppress irrelevant detail – so that rarely used details don't introduce unnecessary complications, but are still available when needed.

(v) They should expose any natural constraints – so that it is easy to express how one object or relation influences another.

(vi) They should be transparent – so you can easily understand what is being said.

(vii) The implementation needs to be concise and fast – so that information can be stored, retrieved and manipulated rapidly.

VI. The four fundamental components of a good representation:

(i) The lexical part – that determines which symbols or words are used in the representation's vocabulary.

(ii) The structural or syntactic part – that describes the constraints on how the symbols can be arranged, i.e. a grammar.

Notes

(iii) The semantic part – that establishes a way of associating real world meanings with the representations.

(iv) 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.

VII. Knowledge Representation in Natural Language

Advantages of natural language are:

(i) It is extremely expressive – we can express virtually everything in natural language (real world situations, pictures, symbols, ideas, emotions, reasoning).

(ii) Most humans use it most of the time as their knowledge representation of choice.

Disadvantages of natural language are:

(i) Both the syntax and semantics are very complex and not fully understood.

(ii) There is little uniformity in the structure of sentences.

(iii) It is often ambiguous – in fact, it is usually ambiguous.

### 3.4.1 Types of Learning

1. ***Perceptual Learning:*** Ability to learn to recognize stimuli that have been seen before:
   - ❖ Primary function is to identify and categorize objects and situations
   - ❖ Changes within the sensory systems of the brain
2. ***Stimulus-response Learning:*** Ability to learn to perform a particular behavior when a certain stimulus is present
   - ❖ Establishment of connections between sensory systems and motor systems
   - ❖ Classical conditioning – association between two stimuli
     - ♦ Unconditioned Stimulus (US), Unconditioned Response (UR), Conditioned Stimulus (CS), Conditioned Response (CR)
     - ♦ Hebb rule – if a synapse repeatedly becomes active at about the same time that the postsynaptic neuron fires, changes will take place in the structure or chemistry of the synapse that will strengthen it
     - ♦ Rabbit experiment – tone paired with puff of air
   - ❖ Instrumental conditioning – association between a response and a stimulus; allows an organism to adjust its behavior according to the consequences of that behavior
     - ♦ Reinforcement – positive and negative
     - ♦ Punishment
3. ***Motor Learning:*** Establishment of changes within the motor system
4. ***Relational Learning:*** Involves connections between different areas of the association cortex
5. ***Spatial Learning:*** Involves learning about the relations among many stimuli
6. ***Episodic Learning:*** Remembering sequences of events that we witness
7. ***Observational Learning:*** Learning by watching and imitation other people

Notes

### 3.4.2 Knowledge Acquisition Problem

The preceding section provided an idealized version of how ES projects might be conducted. In most instances, the above suggestions are considered and modified to suit the particular project. The remainder of this section will describe a range of knowledge acquisition techniques that have been successfully used in the development of ES.

#### Operational Goals

After an evaluation of the problem domain shows that an ES solution is appropriate and feasible, then realistic goals for the project can be formulated. An ES's operational goals should define exactly what level of expertise its final product should be able to deliver, who the expected user is and how the product is to be delivered. If participants do not have a shared concept of the project's operational goals, knowledge acquisition is hampered.

#### Pre-training

Pre-training the knowledge engineer about the domain can be important. In the past, knowledge engineers have often been unfamiliar with the domain. As a result, the development process was greatly hindered. If a knowledge engineer has limited knowledge of the problem domain, then pre-training in the domain is very important and can significantly boost the early development of the ES.

#### Knowledge Document

Once development begins on the knowledge base, the process should be well documented. In addition to tutorial a document, a knowledge document that succinctly state the project's current knowledge base should be kept. Conventions should be established for the document such as keeping the rules in quasi-English format, using standard domain jargon, giving descriptive names to the rules and including supplementary, explanatory clauses with each rule.

*Caution* The rules should be grouped into natural subdivisions and the entire document should be kept current.

#### Scenarios

An early goal of knowledge acquisition should be the development of a series of well developed scenarios that fully describe the kinds of procedures that the expert goes through in arriving at different solutions. If reasonably complete case studies do not exist, then one goal of pre-training should be to become so familiar with the domain that the interviewer can compose realistic scenarios. Anecdotal stories that can be developed into scenarios are especially useful because they are often examples of unusual interactions at the edges of the domain. Familiarity with several realistic scenarios can be essential to understanding the expert in early interviews and the key to structuring later interviews. Finally, they are ultimately necessary for validation of the system.

#### Interviews

Experts are usually busy people and interviews held in the expert's work environment are likely to be interrupted. To maximize access to the expert and minimize interruptions it can be

Notes

helpful to hold meetings away from the expert's workplace. Another possibility is to hold meetings after work hours and on weekends. At least initially, audiotape recordings ought to be made of the interviews because often times notes taken during an interview can be incomplete or suggest inconsistencies that can be clarified by listening to the tape. The knowledge engineer should also be alert to fatigue and limit interviews accordingly. In early interviews, the format should be unstructured in the sense that discussion can take its own course.

*Did u know?* The knowledge engineer should resist the temptation to impose personal biases on what the expert is saying.

During early discussions, experts are often asked to describe the tasks encountered in the domain and to go through example tasks explaining each step. An alternative or supplemental approach is simply to observe the expert on the job solving problems without interruption or to have the expert talk aloud during performance of a task with or without interruption. These procedures are variations of protocol analysis and are useful only with experts that primarily use verbal thought processes to solve domain problems. For shorter term projects, initial interviews can be formalized to simplify rapid prototyping. One such technique is a structured interview in which the expert is asked to list the variables considered when making a decision. Next the expert is asked to list possible outcomes (solutions) from decision making. Finally, the expert is asked to connect variables to one another, solutions to one another and variables to solutions through rules.

A second technique is called twenty questions. With this technique, the knowledge engineer develops several scenarios typical of the domain before the interview. At the beginning of the interview, the expert asks whatever questions are necessary to understand the scenario well enough to determine the solution. Once the expert begins the questions, the expert is asked to explain why each question is asked. When the interviewer perceives a rule, he interrupts and restates the rule to ensure that it is correct.

A third technique is card sorting. In this procedure, the knowledge engineer prepares a stack of cards with typical solutions to problems in the domain. The expert is asked to sort the cards according to some characteristic important to finding solutions to the problem. After each sort, the expert is asked to identify the sorting variable. After each sort, the expert is asked to repeat the process based on another variable.

*Notes* During interviews, it may be helpful to work at a whiteboard to flexibly record and order the exact phraseology of rules or other representations. It may also be helpful to establish recording conventions for use such as color coding different aspects of a rule and using flags to note and defer consideration of significant but peripheral details. Structured interviews should direct the course of a meeting to accomplish specific goals defined in advance.

*Example:* Once a prototypic knowledge base is developed, the expert can be asked to evaluate it line by line. Other less obvious structures can be imposed on interviews, such as asking the expert to perform a task with limited information or during a limited period of time. Even these structured interviews can deviate from the session's intended goals. Sometimes such deviations show subtleties in the expert's procedures and at other times the interview simply becomes sidetracked, requiring the knowledge engineer to redirect the session.

Notes

### 3.4.3 Knowledge Acquisition Represented and Organized

The knowledge-acquisition process includes five major activities:

1. ***Knowledge Acquisition:*** Knowledge acquisition involves the acquisition of knowledge from human experts, books, documents, sensors, or computer files. The knowledge may be specific to the problem domain or to the problem-solving procedures, it may be general knowledge (e.g., knowledge about business), or it may be meta knowledge (knowledge about knowledge). (By meta knowledge, we mean information about how experts use their knowledge to solve problems and about problem-solving procedures in general.) Byrd (1995) formally verified that knowledge acquisition is the bottleneck in ES development today; thus, much theoretical and applied research is still being conducted in this area. An analysis of more than 90 ES applications and their knowledge acquisition techniques and methods is available in Wagner et al. (2003).
2. ***Knowledge Representation:*** Acquired knowledge is organized so that it will be ready for use, in an activity called knowledge representation. This activity involves preparation of a knowledge map and encoding of the knowledge in the knowledge base.
3. ***Knowledge Validation:*** Knowledge validation (or verification) involves validating and verifying the knowledge (e.g., by using test cases) until its quality is acceptable. Testing results are usually shown to a domain expert(s) to verify the accuracy of the ES.
4. ***Inferencing:*** This activity involves the design of software to enable the computer to make inferences based on the stored knowledge and the specifics of a problem. The system can then provide advice to non-expert users.
5. ***Explanation and Justification:*** This step involves the design and programming of an explanation capability (e.g., programming the ability to answer questions such as why a specific piece of information is needed by the computer or how a certain conclusion was derived by the computer).

*Task*

1. Create a structure for a knowledge organization.
2. Prepare and defined the neural network architecture.

## Self Assessment

State whether the following statements are true or false:

10. There are six main topic areas central to knowledge acquisition that require consideration in all ES projects.
11. Experts in the domain must not be able to communicate the details of their problem solving methods.
12. ES should be heuristic and readily distinguishable from algorithmic programs and databases.
13. Episodic learning refers to remembering sequences of events that we witness.
14. Relational learning is the establishment of changes within the motor system.
15. Unstructured interviews should direct the course of a meeting to accomplish specific goals defined in advance.

Notes

## 3.5 Summary

- Knowledge acquisition is integrally tied to how the mind organizes and represents information.
- Learning can be enhanced by considering the fundamental properties of human knowledge as well as by the ultimate function of the desired information.
- The most important property of knowledge is that it is organized semantically; therefore, learning methods should enhance meaningful study of new information.
- Learners should also create as many links to the information as possible.
- Learning methods should be matched to the desired outcome. Just as using a bus timetable to find a bus-stop location is ineffective, learning to recognize information will do little good on an essay exam.

## 3.6 Keywords

***Declarative Knowledge:*** It refers to one's memory for concepts, facts, or episodes, whereas procedural knowledge refers to the ability to perform various tasks.

***Inference Rule:*** An inference rule is a conditional statement with two parts: an *if clause* and a *then clause*.

***Knowledge Acquisition:*** It is the process of adding new knowledge to a knowledge-base and refining or otherwise improving knowledge that was previously acquired.

***Knowledge Representation:*** Knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e. by reasoning about the world rather than taking action in it.

## 3.7 Review Questions

1. Explain semantic network and classify it with frame based representation.
2. Describe the different types of knowledge representation techniques. Explain propositional logic.
3. Explain differences between declarative and procedural knowledge. Explain with the help of examples.
4. How the knowledge representation is defined?

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | False | 2. | True |
| 3. | True | 4. | True |
| 5. | True | 6. | True |
| 7. | True | 8. | True |
| 9. | False | 10. | False |
| 11. | False | 12. | True |
| 13. | True | 14. | False |
| 15. | False | | |

Notes

## 3.8 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://intelligence.worldofcomputing.net/category/knowledge-representation

http://itforum.coe.uga.edu/AECT_ITF_PDFS/paper109.pdf

http://opencourseware.kfupm.edu.sa/colleges/ccse/ics/ics381/files/2_Lectures%209-10-Knowledge%20Representation%20Methods.pdf

http://www.cs.vassar.edu/~weltyc/papers/phd/HTML/dissertation-14.html

http://www.springer.com/engineering/computational+intelligence+and+complexity/book/978-3-540-33518-4

http://www.stanford.edu/class/cs227/Lectures/lec01.pdf

http://www.techfak.uni-bielefeld.de/ags/wbski/lehre/digiSA/WS0506/MDKI/Vorlesung/vl11_knowledgerep.pdf

# Unit 4: LISP

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Identify LISP Applications
- Discuss the Syntax and Numeric Functions
- Explain Basic List Manipulation Functions in LISP
- Describe the concept of Functions and its Procedure

## Introduction

LISP (historically, LISP) is a family of computer programming languages with a long history and a distinctive, fully parenthesized Polish prefix notation. Originally specified in 1958, LISP is the second-oldest high-level programming language in widespread use today; only Fortran is older (by one year). Like Fortran, LISP has changed a great deal since its early days, and a number of dialects have existed over its history. Today, the most widely known general-purpose LISP dialects are Common LISP and Scheme. LISP was originally created as a practical mathematical notation for computer programs, influenced by the notation of Alonzo Church's lambda calculus. It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, LISP pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, and the self-hosting compiler.

The name LISP derives from "List Processing". Linked lists are one of LISP language's major data structures, and LISP source code is itself made up of lists. As a result, LISP programs can

Notes

manipulate source code as a data structure, giving rise to the macro systems that allow programmers to create new syntax or even new domain-specific languages embedded in LISP. LISP was invented by John McCarthy in 1958 while he was at the Massachusetts Institute of Technology (MIT). McCarthy published its design in a paper in Communications of the ACM in 1960, entitled "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I" ("Part II" was never published). He showed that with a few simple operators and a notation for functions, one can build a Turing-complete language for algorithms. Information Processing Language was the first AI language, from 1955 or 1956, and already included many of the concepts, such as list-processing and recursion, which came to be used in LISP. LISP was first implemented by Steve Russell on an IBM 704 computer. Russell had read McCarthy's paper, and realized (to McCarthy's surprise) that the LISP eval function could be implemented in machine code. The result was a working LISP interpreter which could be used to run LISP programs, or more properly, 'evaluate LISP expressions.' The first complete LISP compiler, written in LISP, was implemented in 1962 by Tim Hart and Mike Levin at MIT. This compiler introduced the LISP model of incremental compilation, in which compiled and interpreted functions can intermix freely. The language used in Hart and Levin's memo is much closer to modern LISP style than McCarthy's earlier code.

LISP was a difficult system to implement with the compiler techniques and stock hardware of the 1970s. Garbage collection routines, developed by then-MIT graduate student Daniel Edwards, made it practical to run LISP on general-purpose computing systems, but efficiency was still a problem. This led to the creation of LISP machines: dedicated hardware for running LISP environments and programs. Advances in both computer hardware and compiler technology soon made LISP machines obsolete. During the 1980s and 1990s, a great effort was made to unify the work on new LISP dialects (mostly successors to Mac lisp like Zeta LISP and NIL (New Implementation of LISP)) into a single language. The new language, Common LISP, was somewhat compatible with the dialects it replaced (the book Common LISP the Language notes the compatibility of various constructs). In 1994, ANSI published the Common LISP standard, "ANSI X3.226-1994 Information Technology Programming Language Common LISP."

## 4.1 LISP Application

LISP is an expression-oriented language. Unlike most other languages, no distinction is made between "expressions" and "statements"; all code and data are written as expressions. When an expression is evaluated, it produces a value (in Common LISP, possibly multiple values), which then can be embedded into other expressions. Each value can be any data type. McCarthy's 1958 paper introduced two types of syntax: S-expressions (Symbolic expressions, also called "sexps"), which mirror the internal representation of code and data; and M-expressions (Meta Expressions), which express functions of S-expressions.

*Caution* M-expressions never found favor, and almost all Lisps today use S-expressions to manipulate both code and data.

The use of parentheses is Lisp's most immediately obvious difference from other programming language families. As a result, students have long given LISP nicknames such as Lost In Stupid Parentheses, or Lots of Irritating Superfluous Parentheses. However, the S-expression syntax is also responsible for much of Lisp's power: the syntax is extremely regular, which facilitates manipulation by computer. However, the syntax of LISP is not limited to traditional parentheses notation. It can be extended to include alternative notations.

*Example:* XM LISP, for instance, is a Common LISP extension that employs the meta object-protocol to integrate S-expressions with the Extensible Markup Language (XML).

Notes

The reliance on expressions gives the language great flexibility. Because LISP functions are themselves written as lists, they can be processed exactly like data. This allows easy writing of programs which manipulate other programs (meta programming).

Many LISP dialects exploit this feature using macro systems, which enables extension of the language almost without limit. A LISP list is written with its elements separated by whitespace, and surrounded by parentheses. The empty list () is also represented as the special atom nil. This is the only entity in LISP which is both an atom and a list.

The following industrial LISP users are grouped by industry:

- Aerospace/Airlines
- Animation/Graphics
- Automotive
- Computers and Computer Tools
- Computer-aided Design
- Consulting/Research
- Databases
- Expert Systems and Knowledge-based Engineering Tools
- Education
- Electronics
- Energy
- Finance
- Government
- Groupware/Project Management
- Internet/World-Wide-Web
- Publishing
- Mathematics
- Medicine/Biotechnology
- Music
- Telecommunications
- Other

## Self Assessment

State whether the following statements are true or false:

1. LISP is an expression-oriented language.
2. McCarthy's 1958 paper introduced four types of syntax.
3. A LISP list is written with its elements separated by whitespace, and surrounded by parentheses.
4. The empty list () is not represented as the special atom nil.

Notes

## 4.2 Syntax and Numeric Functions

The two fundamental pieces of LISP syntax are the idea of a list of things and the dotted pair:

- For any set of strings X, list_of(X)::=”(“ # X “)”,
- For any set of strings X, dotted_pair(X)::=”(“ X dot X “)”.

A dotted pair is the external image of a node in an internal data structure with two components called “car” and “cdr” for historical reasons.

*Did u know?* Each component in the pair points at an atomic item or another list.

Each list is coded internally as dotted pairs:

```
(A)  =  (A.NIL).
(A  B)  =  (A.(B.NIL)).
(A  B  C)  =  (A.(B.(C.NIL))).
(A  B  C  D  ...  Y  Z)  =  (A.(B.(C.(.D  ...  (Y.  (Z.  NIL))  ...).
```

After the lexical analysis the interpreter parses input as a list_structure before evaluating it. A list_structure is empty, or a dotted_pair, or an atom, or any number of lists inside a pair of parenthesis. We often call a list structure a list for short. But always be careful to distinguishing ‘a list of numbers’ from a ‘list structure of numbers’

- list_structure::=atom | dotted_pair(list_structure) | list_of(list_structure).

*Notes* In old LISP, commas were optional separators in lists but not in XLISP and Scheme. The separating spaces, tabs, etc. in a list are all sieved out by the lexical scan. This scan also removes comment lines that start with a semicolon [ comment in lisp.lexemes ]

- atom::= [ atom in lisp. lexemes ]

The “Unknowns” below indicate that an entry is incomplete.

- either the entry exist in the language,
- either the entry doesn’t exist in the language. The entry will be marked as such and won’t appear as missing anymore.
- ***Category:*** Object Oriented, Dynamically typed, Functional
- ***Various***

| | |
|---|---|
| `nothing needed` | breaking lines (useful when end-of-line and/or indentation has a special meaning) |
| `#| ... |#(1)` | commenting (nestable) |
| `;` | commenting (until end of line) |
| `< > <= >=` | Comparison |
| `min/max` | comparison (min/max (binary or more)) |
| `(defun f (para1 para2) "..." ...)` | documentation comment |
| `Equal` | equality/inequality (deep) |
| `Equalp` | equality/inequality (deep) |
| `eq, eql` | equality/inequality (shallow) |

*Contd...*

Notes

| | |
|---|---|
| `(ext:gc)` | force garbage collection |
| `Eval` | runtime evaluation |
| `case-insensitive` | tokens (case-sensitivity (keywords, variable identifiers...)) |
| `anything without a space and is not a number` | tokens (variable identifier regexp) |
| `hyphens` | tokens (what is the standard way for scrunching together multiple words) |
| `Setq` | variable assignment or declaration (assignment) |
| `Setf` | variable assignment or declaration (assignment) |
| `Set` | variable assignment or declaration (assignment) |
| `let let*` | variable assignment or declaration (declaration) |
| `flet labels defun defmethod defvar defparameter defsetf ..` | variable assignment or declaration (declaration) |

- *Functions*

| | |
|---|---|
| `(lambda (a b) ...)` | anonymous function |
| `(f a b ...) (apply f l)` | function call |
| `(funcall f a b ...)` | function call |
| `(f)` | function call (with no parameter) |
| `(funcall f)` | function call (with no parameter) |
| `no-applicable-method` | function called when a function is not defined (in dynamic languages) |
| `(defun f (para1 para2) ...)` | function definition |
| `Return` | function return value (breaks the control flow) |
| `Return-from xxx` | function return value (breaks the control flow) |
| `no syntax needed` | function return value (function body is the result) |
| `identity` | identity function |

- *Control Flow*

| | |
|---|---|
| `/return-from xxx or return` | breaking control flow (continue/break) |
| `go throw` | breaking control flow (goto (unconditional jump)) |
| `Return` | breaking control flow (returning a value) |
| `Return-from xxx` | breaking control flow (returning a value) |
| `handler-bind handler-case ignore-errors` | exception (catching) |
| `Unwind-protect` | exception (cleanup: code executed before leaving) |
| `Error` | exception (throwing) |
| `Signal` | exception (throwing) |
| `Cerror warn` | exception (throwing) |
| `(if c ...)` | if_then |
| `(if c b1 b2)` | if_then_else |
| `(cond (c b1) (c2 b2) (t b3))` | if_then_else |

*Contd...*

Notes

| | |
|---|---|
| `(loop do ... until c)` | loop (do something until condition) |
| `(loop with VAR = INITIAL-VALUE ... while CONDITION finally INCREMENT ...)` | loop (for "a la C" (while + initialisation)) |
| `(loop for i from 1 to 10 by -1 do ...)` | loop (for each value in a numeric range, 1 decrement) |
| `(loop for i from 1 to 10 do ...)` | loop (for each value in a numeric range, 1 increment (see also the entries about ranges)) |
| `(loop for i from 1 to 10 by 2 do ...)` | loop (for each value in a numeric range, free increment) |
| `(loop do ...)` | loop (forever loop) |
| `(loop while c do ...)` | loop (while condition do something) |
| `(case val`<br>`((v1) ...)`<br>`((v2) ...)`<br>`(otherwise ...))` | multiple selection (switch) |
| `(progn ...) (prog1 ...) (prog2 ...)` | Sequence |

- ***Types***

| | |
|---|---|
| `(declare (t v))` | annotation (or variable declaration) |
| `(deftype n () 't)` | Declaration |

- ***Object Oriented & Reflexivity***

| | |
|---|---|
| `call-next-method` | accessing parent method |
| `defclass defstruct` | class declaration |
| `dispatching parameter` | current instance |
| `type-of` | get the type/class corresponding to an object/instance/value |
| `find-method` | has the method |
| `(defclass child (parent) ...)` | Inheritance |
| `(method object para)` | method invocation |
| `(method object)` | method invocation (with no parameter) |
| `(make-instance class_name ...)` | object creation |
| `Typep` | testing class membership |

- ***Package, Module***

| | |
|---|---|
| `(defpackage p ...)` | Declare |
| `(export 'name1 'name2)` | declare (selective export) |
| `(use-package 'p)` | import (everything into current namespace) |
| `(require 'p)` | import (package (ie. load the package)) |
| `(import '(p:name1 p:name2))` | import (selectively) |
| `:::(5)` | package scope |

- ***Strings***

| | |
|---|---|
| `char, aref, schar, svref` | accessing n-th character |
| `Aref` | accessing n-th character |
| `code-char` | ascii to character |

*Contd...*

Notes

| | |
|---|---|
| `#\z` | character "z" |
| `char-code` | character to ascii |
| `(coerce e 'string)` | convert something to a string (see also string interpolation) |
| `Subseq` | extract a substring |
| `Search` | locate a substring |
| `(search substring string:from-end t)` | locate a substring (starting at the end) |
| `all strings allow multi-line strings` | multi-line |
| `(with-standard-io-syntax (write obj stream))` | serialize (marshalling) |
| `Write` | simple print (on any objects) |
| `Print` | simple print (on any objects) |
| `princ prin1` | simple print (on any objects) |
| `write-string` | simple print (on strings) |
| `Format` | simple print (printf-like) |
| `Format` | sprintf-like |
| `concatenate` | string concatenation |
| `equal, equalp` | string equality & inequality |
| `Length` | string size |
| `"~%"` | strings (end-of-line (without writing the real CR or LF character)) |
| `"..."` | strings (with no interpolation of variables) |
| `(with-standard-io-syntax (read obj stream))` | unserialize (un-marshalling) |
| `char-upcase/char-downcase` | upper/lower case character |
| `string-upcase/string-downcase` | uppercase/lowercase/capitalized string |

- ***Booleans***

| | |
|---|---|
| `Nil` | false value |
| `not` | logical not |
| `or/and` | logical or/and (short circuit) |
| `T` | true value |
| `anything not false` | true value |
| `boolean` | type name |

- ***Bags and Lists***

| | |
|---|---|
| `Cons` | adding an element at the beginning (list cons) (return the new list (no side-effect)) |
| `Push` | adding an element at the beginning (list cons) (side-effect) |
| `Cdr` | all but the first element |
| `Reduce` | f(... f(f(init, e1), e2) ..., en) |
| `(reduce f '(e1 e2 ... en):from-right t:initial-value init)` | f(e1, f(e2, ... f(en, init) ...)) |
| `Find` | find an element |

*Contd...*

Notes

| | |
|---|---|
| find-if | find an element |
| Car | first element |
| (dolist (v l) ...) (loop for v in l do ...) mapc | for each element do something |
| Pop | get the first element and remove it |
| Member | is an element in the list |
| Some | is the predicate true for an element |
| Every | is the predicate true for every element |
| (loop for v in l as i upfrom 0 do ...) | iterate with index |
| Remove-if-not delete-if-not | keep elements (matching) |
| Remove-if delete-if | keep elements (non matching) |
| (car (last l)) | last element |
| Nconc | list concatenation |
| Append | list concatenation |
| '(a b c) | list constructor |
| List | list constructor |
| Pairlis | list of couples from 2 lists |
| Length | list size |
| Nth | list/array indexing |
| Aref | list/array indexing |
| Assoc | lookup an element in a association list |
| Delete-duplicates | remove duplicates |
| Remove-duplicates | remove duplicates |
| Reverse | Reverse |
| min/max | smallest/biggest element |
| sort | Sort |
| split-sequence | split a list (into sublists delimited by elements matching a predicate) |
| Mapcar | transform a list (or bag) in another one |
| Mapcar | transform two lists in parallel |

- *Unknown:*
  - list flattening adding an element at index get the last element and remove it partition a list: elements matching, elements non matching join a list of strings in a string using a glue string
- *Various Data Types*

| | |
|---|---|
| (gethash k h) | dictionary (access: read/write) |
| (gethash k h) | dictionary (has the key ?) |
| remhash | dictionary (remove by key) |
| Nil | optional value (null value) |
| V | optional value (value) |
| Normal function call | record (selector) |
| (cons a b) | tuple constructor |

Notes

- *Unknown:*
  - computable tuple (these are a kind of immutable lists playing a special role in parameter passing) (empty tuple) computable tuple (these are a kind of immutable lists playing a special role in parameter passing) (1-uple) computable tuple (these are a kind of immutable lists playing a special role in parameter passing) (using a tuple for a function call) optional value (null coalescing) dictionary (constructor) dictionary (merge) range
- *Mathematics*

| | |
|---|---|
| `+/-/*//` | addition/subtraction/multiplication/division |
| `Logand/logior/logxor` | bitwise operators (and/or/xor) |
| `Lognot` | bitwise operators (bitwise inversion) |
| `(ash x positive-integer)/(ash x negative-integer)/` | bitwise operators (left shift/right shift/unsigned right shift) |
| `Floor` | euclidian division (both quotient and modulo) |
| `Expt` | exponentiation (power) |
| `(log x 10)` | logarithm (base 10) |
| `Log` | logarithm (base e) |
| `Mod` | modulo (modulo of -3/2 is 1) |
| `-` | Negation |
| `1000.0, 1E3` | numbers syntax (floating point) |
| `#b1, #o7, #xf` | numbers syntax (integers in base 2, octal and hexadecimal) |
| `#2r1, #8r7, #16rf` | numbers syntax (integers in base 2, octal and hexadecimal) |
| `1000, 1000.` | numbers syntax (integers) |
| `Random` | random (random number) |
| `make-random-state` | random (seed the pseudo random generator) |
| `sqrt/exp/abs` | square root/e-exponential/absolute value |
| `sin/cos/tan` | trigonometry (basic) |
| `asin/acos/atan` | trigonometry (inverse) |
| `truncate/round/floor/ceiling` | truncate/round/floor/ceil |

## Self Assessment

State whether the following statements are true or false:

5. A dotted pair is the internal image of a node in an internal data structure with two components.
6. In LISP, commas were optional separators in lists.
7. LISP can be used in finding the human behavior.
8. Each list is coded internally as dotted pairs.

Notes

## 4.3 Basic List Manipulation Functions in LISP

Basic List Manipulation Functions in LISP are as follows:

### 4.3.1 List Manipulation—Step 1

As you are probably well aware, LISP stands for "List Processing". (Not "Lost in Stupid Parenthesis"). A list is a group of elements consisting of any data type and is stored as a single variable. A list can contain any number of Reals, Integers, Strings, Variables and even other Lists.

Let's have a look at a list. Type this:

```
(setq pt1 (getpoint "\nChoose a Point: "))
```

AutoLisp should return something like this:

```
(127.34 35.23 0.0)
```

Fine, you say, I've got a list but what do I do with it? AutoLisp has many functions available to manipulate lists. Let's have a look at them.

#### Car

The primary command for taking a list apart is the "Car" function. This function returns the first element of a list. (The x coordinate.)

For example,

```
(setq a (car pt1))
```

Would return:

```
(127.34)
```

#### Cdr

This function returns the second element plus the remaining elements of a list. For example,

```
(setq b (cdr pt1))
```

Would return:

```
(35.23 0.0)
```

But what if we only wanted the second element? We could write:

```
(setq b (car (cdr pt1)))
```

But there is a better way. AutoLisp has provided the "Cadr" function which is basically an abbreviation of a nested command.

#### Cadr

This returns the second element of a list. (The y coordinate)

```
(setq b (cadr pt1))
```

This would return:

```
(35.23)
```

Likewise, there is another abbreviated function to return the third element.

Notes

### Caddr

This returns the third element of a list. (The z coordinate)

```
(setq c (caddr pt1))
```

Would return:

```
(0.0)
```

AutoLisp has other functions that will retrieve values from lists of more than three elements. (Caar, cadar, etc.). You can, though, us another function to access any element of a list. This is the "nth" function.

### nth

The syntax for the nth function is as follows:

```
(nth num list)
```

"num" is the number of the element to return. Just remember that zero is the first element. For example given the list:

```
(setq d '("M10" "M20" "M30" 10.25))
(setq e (nth 0 d))
```

Would return:

```
("M10")
```

And likewise:

```
(setq f (nth 3 d))
```

Would return:

```
(10.25)
```

### Syntax Synergy

The thing that I think scares most everyday hackers away from lisp is the (comparatively) very odd syntax it's written in. One of the simple functions we wrote while working through the first couple units looks like this:

```
(defun list-copy (lst)
  (if (atom lst)
      lst
      (cons (car lst) (list-copy (cdr lst)))))
```

To someone who's used to the simula syntax languages (as most programmers are), this can be a very challenging paradigm shift. All those parentheses hovering around, seemingly providing random groupings of keywords. It's like this for a pragmatic reason, though, and in the end it's actually simpler than the languages you might be used to. You don't have to know where to use braces, brackets, parentheses, or dots.

*Notes* Everything is grouped by the same syntax, and everything is a list. It takes some getting used to, but if you work with it for a couple days your brain adapts and starts to actually compose your code ideas this way.

Notes

### Simple Math

Step one when picking up a language is usually to work with the simple things that are easily defined. Namely, mathematics. It's an easy way to break into an unfamiliar workspace since there's usually a lot of commonality across languages and it can be a more gentle introduction than jumping right into a catalogue of basic functions. This is true when moving to LISP, but perhaps less so than other languages because most languages stick to the elementary school standard "5 + 4" for doing simple arithmetic operations. LISP, true to form (List Processing language), treats arithmetic operators just like any other function call. Standard syntax is something like this (the carrot is actually the prompt in the interpreter and is not part of the language):

```
>(function  arg1  arg2)
result
```

so addition is done like this:

```
>(+  5  4)
9
```

This can feel strange since we're used to a different order even in verbalizing the expression (Five Plus Four), but think about the benefit of prefix notation when you start to add extra arguments. Most languages chain together addition like this:

```
5 + 4 + 3 + 2 + 1
```

Notice any redundancy? We've used the "+" operator 4 times in one "sum" operation. Using lisp, this same expression becomes:

```
>(+  5  4  3  2  1)
15
```

More concise, and actually more natural when you think about the way you do your sums on paper when you have no calculator:

```
  10
  14
  12
+ 8
  44
```

why express over and over that you are STILL doing addition? why not just use the operator once? Subtraction is done the same way, but is a little more awkward since it's not commutative the way addition is, and since we usually think of a dash at the front of an expression when we are signifying a "not" operation on the result.

```
>(-  5  4)
1
```

and multiplication and division are in the same vein:

```
>(*  10  5  2)
100
>(/30  6)
5
```

Pretty simple, but enough to get your feet wet and to start practicing with a new syntax. Next time we'll go just a little deeper and look at list manipulation.

## 4.3.2 List Manipulation—Step 2

Each item in the list has two things: a value (the item ON the stump) and a link to the rest of the list (the string leading from the branch to the next stump). Therefore, you're "list" of 10 items on

Notes

tree stumps is actually a "pair" of things: an item on the first tree stump, and a pointer leading to a list of 9 other tree-stumps with things on them. So what is that 9-tree-stump list? It is ALSO a pair of items: the item on the first tree stump, and a pointer to a list of 8 other tree-stumps with things on them. This subdivision goes all the way down to the last tree stump which is STILL a pair of items: the item on the tree stump, and a pointer to nothing (NIL).

Now, why is this distinction important? Because the three basic list-manipulation functions in lisp are based around this paradigm. Look at the following function call:

```
> (cons 'B '(O R D E R))
(B O R D E R)
```

It takes two arguments (an element and a list) and puts them together as a list. Basically, it's the same as putting something on another tree-stump ('B) and tying that tree-stump's branch to the first stump in the original list. How about these two:

```
> (car '(L I S P))
L
> (cdr '(L I S P))
(I S P)
```

The CAR function takes a list as a parameter, and returns the first part of the pair (the thing on the first tree stump). The CDR function takes a list as well, and it returns the SECOND part of the pair (the list of other tree stumps that the first one points to). If you think about it, you'll see why this is important: you can do pretty much anything to a list with those three functions. They are the building blocks of any functionality you need regarding a list. Here is a good example, it's one of the practice exercises from Paul Graham's "ANSI Common LISP":

```
(defun our-list-copy (lst)
  (if (atom lst)
    lst
    (cons (car lst) (list-copy (cdr lst)))))
```

We've just written a new function called "our-list-copy" that takes a list as a parameter and returns a copy of it by recursively traveling through the list and "CONS"-ing each first-element (CAR) with a copy of the rest of the list (CDR).

*Did u know?* This kind of development is core to what lispers describe as bottom up programming. You start out writing simple, minimalist functions. As your needs become specific you combine these low-level functions into more powerful functions, which are in turn used to write domain-specific functions, which in turn become a kind of language for the construction of your top-level program. Basically, you're adding to the language as you go.

### 4.3.3 List Manipulation—Step 3

Most programs cannot be written without branching at some point. It's all well and good to be able to execute various statements, but it doesn't really become useful until the program can decide to take a certain action based on some evaluation of a condition (i.e., if it's below 40 degrees, I will wear a coat outside).

When writing lisp, you base all programmatic logic on predicates (technically this is true for any language, but because of the prefix syntax all predicates FEEL like predicate functions rather than X == Y). In mathematics, a predicate is either a relation or the boolean-valued function that amounts to the characteristic function or the indicator function of such a relation. This is essentially true in lisp as well: a predicate is a function that returns a true or false value regarding a certain characteristic of it's parameter(s). False is represented by NIL, and true is anything that is Not-NIL (most commonly represented by T).

Notes

There are two basic types of predicates in LISP: Data-type predicates and equality predicates. Data-type predicates take one argument and determine whether or not that argument qualifies as the specified datatype. Here are some examples:

```
> (numberp 5)
T
> (numberp 'FIVE)
NIL
> (stringp "hello, world")
T
> (stringp 100)
NIL
```

"numberp" is a predicate function that returns T if the argument is a number, and NIL if it is not. Likewise, "stringp" determines whether or not the argument is a string. Most data-type predicates are written with the convention seen above; namely, the data-type name followed by a "p" for predicate. The two common exceptions are "null" which determines whether or not an expression evaluates to NIL, and "atom" which determines whether or not the argument is an atom (a single value, indivisible into smaller parts...basically, not a list). Here are some common data-type predicates built into the lisp language:

```
null
symbolp
atom
consp
listp
numberp
integerp
rationalp
floatp
complexp
characterp
stringp
vectorp
arrayp
functionp
```

### Self Assessment

State whether the following statements are true or false:

9. The primary command for taking a list apart is the "Cdr" function.
10. Zero is the first element.
11. Most programs can be written without branching at some point.
12. Most data-type predicates are written with the convention.

## 4.4 Functions

LISP is a language that is usually associated with the "Functional Programming" paradigm. For those of you who aren't familiar with that term, functional programming takes a problem and models it as mathematical functions for evaluation. This is in contrast to imperative programming where "state" is the focus, and the outcome of a program is modeled in changes of state. As a primarily-functional programming language, the most important building block of a LISP program is the function, so here we're going to talk about the specifics of defining new functions and using them.

Notes

### 4.4.1 Defining Functions

If we have used to using one of the more common enterprise languages like Java or C#.NET, the following probably looks pretty familiar (example code will be written in java):

```
public class MyMath
{
  public static int AddTogether(int oneNumber,int otherNumber)
  {
    return oneNumber + otherNumber;
  }
}
```

This is a simple function for adding two numbers together. Now, here's the same function, but written in LISP:

```
(defun addTogether (oneNumber otherNumber)
  (+ oneNumber otherNumber))
```

that's a little smaller, isn't it? Let's examine some of the differences. First, the java example has this "public static" at the beginning. In an imperative language, these are important specifiers. "public" says that any code can use that function, both code within the MyMath class structure and code elsewhere. Other modifiers like "protected" and "private" would restrict access to that function to a subset of code (child classes, and only the defining class, respectively). "static" means that this is a "class-level" function, or in other words, you don't need to have an instance of the MyMath class already constructed to use this function. We can call it without instantiating any objects. If "static" were omitted, we would have to first instantiate a "MyMath" object, and then make the function call.

So how is this information conveyed in LISP? Simple: it doesn't need it. Access modifiers like "public" have to do with whether code outside the class can use the function or not. Functional programming doesn't use classes, the AddTogether function is just a piece of code that adds two numbers together and returns the result, and once it has been loaded into the runtime it can be called from any other piece of code. The same goes for "static": there aren't any "objects" or "classes" in functional programming, so there's no need to specify whether the function belongs to the class or to instances of the class; it just exists.

### 4.4.2 Defining Procedures

So far we've just used the Listener as a calculator, to evaluate expressions. In this section, we're going to make a huge leap forward and show you how to define your own procedures. Once you've defined a procedure, it has the same status as the built-in procedures, like list and +. Let's define a procedure to return the average of two numbers.

In words, the procedure for finding the average of two numbers is:

- Add the first number to the second number.
- Divide the sum by 2.

#### Defining a Procedure – Defun

To define a procedure we use the special operator defun. This is an abbreviation for define function, but LISP procedures are not strictly functions in the mathematical sense. We can write the average procedure as follows:

```
(defun average (1st-number 2nd-number)
  (/(+ 1st-number 2nd-number) 2))
```

The first argument to defun gives the name of the procedure, which we've chosen as average.

**Notes**

The second argument, (1st-number 2nd-number), is a list of what are called the parameters. These are symbols representing the numbers we are going to refer to later in the procedure.

The rest of the definition, (/(+ 1st-number 2nd-number) 2), is called the body of the procedure. It tells LISP how to calculate the value of the procedure, in this case sum the numbers and divide by 2.

The words you use for the parameters are arbitrary, as long as they match the words you use in the procedure definition. So you could equally well have written the **average** procedure as follows:

```
(defun average (a b)
  (/(+ a b) 2))
```

To define it we can type the definition at the LISP prompt:

```
CL-USER > (defun average (1st-number 2nd-number)
  (/(+ 1st-number 2nd-number) 2))
AVERAGE
```

Alternatively you could type the definition into the LISP Editor and select **Compile Buffer** to evaluate it.

We can try it out with:

```
CL-USER 13 > (average 7 9)
8
```

or:

```
CL-USER 14 > (average (+ 2 3) (+ 4 5))
7
```

Remember that the arguments are evaluated before they are given to the procedure.

### Procedures Without Parameters

A procedure doesn't have to have any parameters. Here's a procedure **dice** that returns a random dice throw from 1 to 6:

```
(defun dice ()
  (+ 1 (random 6)))
```

To call the procedure you simply write:

```
CL-USER 10 > (dice)
5
```

*Example 1:*

***Objective:*** Write a program showing use of LISP arithmetic functions.

***Solution:***

| **Function Call** | **Value Returned** |
|---|---|
| (+ 3 5 8 4) | 20 |
| (- 10 12) | -2 |
| (* 2 3 4) | 24 |
| (/25 2) | 12.5 |

Notes

*Example 2:*

***Objective:*** Write a program for implementation of LISP manipulation functions.

***Solution:***

| **Function Call** | **Value Returned** |
|---|---|
| (Car '(a b c)) | A |
| (Cdr '(a b c)) | (B C) |
| (Cons 'a '(b c)) | (A B C) |
| (List 'a '(b c)) | (A (B C)) |
| (Append '(a) '(b c)) | (A B C) |
| (Last '(a b c d)) | (D) |
| (Member 'b '(a b d)) | (B D) |
| (Reverse '(a (b c) d)) | (D (B C) A) |

*Example 3:*

***Objective:*** Write a program for implementation of LISP Boolean function.

***Solution:***

| **Function Call** | **Value Returned** |
|---|---|
| (Atom 'aabb) | t |
| (Equal 'a (car '(a b)) | t |
| (Evenp 3) | nil |
| (Number 10ab) | nil |
| (Oddp 3) | t |
| (Zerop .000001) | nil |
| (Greater 2 4 27) | t |
| (Lessp 5 3 1 2) | nil |
| (Listp '(a)) | t |
| (Null nil) | t |

*Example 4:*

***Objective:*** Write a LISP Program calculating average of three numbers.

***Solution:***

**AVERAGE OF THREE NUMBERS**

➔ (defun averagethree(n1 n2 n3) (/(+ n1 n2 n3)3))

AVERAGETHREE

**OUTPUT**

(Averagethree 10 20 30)

Notes

*Example 5:*

***Objective:*** Write a program in LISP showing implementation of condition.

***Solution:***

**Maximum of Two Numbers**

:->(defun maximum2 (a b) (cond ((>a b) a) (t b)))

MAXIMUM2

**OUTPUT**

(maximum2 234 320)

:-> 320

**Maximum of Three Numbers**

(defun maximum3(a b c)(cond((>a b)(cond((>a c)a)(t c)))((>b c)b)(t c)))

MAXIMUM3

**OUTPUT**

(maximum3 20 30 25)

:-> 30

*Task* Write a LISP program for login and password checking.

## Self Assessment

State whether the following statements are true or false:

13. LISP is a language that is usually associated with the Functional Programming paradigm.
14. To define a procedure we use the special operator average.
15. The words you use for the parameters are arbitrary.
16. A procedure does have to have any parameters.

## 4.5 Summary

- LISP programs are made up of expressions, which are lists or single atoms.
- Lists are made up of zero or more atoms or inner lists, separated by white space and surrounded by parentheses. A list can be empty.
- Atoms are multi-character symbols, like forward-paragraph, single character symbols like +, strings of characters between double quotation marks, or numbers.
- A number evaluates to itself.
- A string between double quotes also evaluates to itself.
- When you evaluate a symbol by itself, its value is returned.

Notes

- When you evaluate a list, the LISP interpreter looks at the first symbol in the list and then at the function definition bound to that symbol. Then the instructions in the function definition are carried out.
- A single quotation mark, ', tells the LISP interpreter that it should return the following expression as written, and not evaluate it as it would if the quote were not there.
- Arguments are the information passed to a function. The arguments to a function are computed by evaluating the rest of the elements of the list of which the function is the first element.
- A function always returns a value when it is evaluated (unless it gets an error); in addition, it may also carry out some action called a "side effect". In many cases, a function's primary purpose is to create a side effect.

## 4.6 Keywords

***CMUCL:*** The CMUCL is a high-performance, free common LISP implementation.

***Common LISP Implementation:*** It is licensed under a LISP-specific variant of the LGPL, and derived from Digitool's MCL product.

***FORTRAN:*** It has been used for such projects as the design of bridges and aeroplane structures; it is used for factory automation control, for storm drainage design, analysis of scientific data and so on.

***Functional Calculus:*** It is a theory allowing one to apply mathematical functions to mathematical operators. It is now a branch (more accurately, several related areas) of the field of functional analysis, connected with spectral theory.

***Working Knowledge:*** It is a social enterprise dedicated to opening the eyes of business to young talent available locally.

## 4.7 Review Questions

1. Define LISP?
2. What is the different syntax and numeric functions of LISP?
3. Explain manipulation functions.
4. Abbreviate and meaning of Car, Cdr, Cadr, Caddr.
5. Write the difference between Cadr and Caddr.
6. What are LISP functions? Describe the LISP functions.
7. Explain the procedures with examples.
8. Write the examples for the uses of syntax in the LISP.

### Answers: Self Assessment

1. True
2. False
3. True
4. False
5. False
6. True
7. True
8. True

Notes

9. False
10. True
11. False
12. True
13. True
14. False
15. True
16. False

## 4.8 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://blog.ethanvizitei.com/2008/01/lisp-ing-part-2-basic-list-manipulation.html

http://people.cs.pitt.edu/~milos/courses/cs2740/Lectures/LispTutorial.pdf

http://www.cs.iusb.edu/~danav/teach/c311/c311_elisp2.html

http://www.cs.sfu.ca/CourseCentral/310/pwfong/LISP/1/tutorial1.html

http://www.n-a-n-o.com/lisp/cmucl-tutorials/LISP-tutorial-24.html

http://www.worldclasscad.com/lisp_pdf/chapter_08_lisp_ manipulation_functions.pdf

www.apl.jhu.edu/~hall/lisp-books.html

www.engin.umd.umich.edu/CIS/course.des/cis400/lisp/lisp.html

# Unit 5: Artificial Intelligence Programming Language

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Discuss the AI Programming Languages
- Explain the Predicates and Conditionals in AI
- Describe Input, Output, and Local Variables
- Explain the Iteration and Recursion in AI
- Describe the Property Lists and Arrays in AI
- Explain Prolog and Other AI Programming Languages

## Introduction

Programming languages in AI are the major tool for exploring and building computer programs that can be used to simulate intelligent processes such as learning, reasoning and understanding symbolic information in context. Although, in the early days of computer language design, the primarily use of computers was for performing calculations with numbers, it was also found out quite soon that strings of bits could represent not only numbers but also features of arbitrary objects. Operations on such features or symbols could be used to represent rules for creating, relating or manipulating symbols. This led to the notion of symbolic computation as an appropriate means for defining algorithms that processed information of any type, and thus

Notes

could be used for simulating human intelligence. Soon it turned out that programming with symbols required a higher level of abstraction than was possible with those programming languages which were designed especially for number processing, e.g., Fortran.

## 5.1 AI Programming Languages

In AI, the automation or programming of all aspects of human cognition is considered from its foundations in cognitive science through approaches to symbolic and sub-symbolic AI, natural language processing, computer vision, and evolutionary or adaptive systems. It is inherent to this very complex problem domain that in the initial phase of programming a specific AI problem, it can only be specified poorly. Only through interactive and incremental refinement does more precise specification become possible. This is also due to the fact that typical AI problems tend to be very domain specific; therefore heuristic strategies have to be developed empirically through generate-and-test approaches (also known as rapid prototyping). In this way, AI programming notably differs from standard software engineering approaches where programming usually starts from a detailed formal specification. In AI programming, the implementation effort is actually part of the problem specification process. Due to the "fuzzy" nature of many AI problems, AI programming benefits considerably if the programming language frees the AI programmer from the constraints of too many technical constructions (e.g., low-level construction of new data types, manual allocation of memory).

*Did u know?* A declarative programming style is more convenient using built-in high-level data structures (e.g., lists or trees) and operations (e.g., pattern matching) so that symbolic computation is supported on a much more abstract level than would be possible with standard imperative languages, such as Fortran, Pascal or C. Of course, this sort of abstraction does not come for free, since compilation of AI programs on standard Von Neumann computers cannot be done as efficiently as for imperative languages.

However, once a certain AI problem is understood (at least partially), it is possible to reformulate it in form of detailed specifications as the basis for re-implementation using an imperative language. From the requirements of symbolic computation and AI programming, two new basic programming paradigms emerged as alternatives to the imperative style: the functional and the logical programming style. Both are based on mathematical formalisms, namely recursive function theory and formal logic. The first practical and still most widely used AI programming language is the functional language Lisp developed by John McCarthy in the late 1950s. Lisp is based on mathematical function theory and the lambda abstraction. A number of important and influential AI applications have been written in Lisp so we will describe this programming language in some detail in this article. During the early 1970s, a new programming paradigm appeared, namely logic programming on the basis of predicate calculus. The first and still most important logic programming language is Prolog, developed by Alain Colmerauer, Robert Kowalski and Phillippe Roussel. Problems in Prolog are stated as facts, axioms and logical rules for deducing new facts.

*Notes* Prolog is mathematically founded on predicate calculus and the theoretical results obtained in the area of automatic theorem proving in the late 1960s.

### Self Assessment

Notes

State whether the following statements are true or false:

1. AI programming differs from standard software engineering approaches where programming usually ends with a detailed formal specification.
2. A declarative programming style is more convenient using built-in high-level data structures.
3. Lisp is not based on mathematical function theory and the lambda abstraction.

## 5.2 Predicates and Conditionals

Predicate logic builds heavily upon the ideas of proposition logic to provide a more powerful system for expression and reasoning. As we have already mentioned, a predicate is just a function with a range of two values, say false and true. We use predicates routinely in programming, e.g. in conditional statements of the form

```
if(p(...args  ...))
```

Here, we are using the two possibilities for the return value of p, (true or false). We also use the propositional operators to combine predicates, such as in:

```
if(p(....)  &&  (!q(....)  ||  r(....)))
```

Predicate logic deals with the combination of predicates using the propositional operators we have already studied. It also adds one more interesting element, the "quantifiers". The meaning of predicate logic expressions is suggested by the following:

Expression + Interpretation + Assignment = Truth Value

Now, we explain this equation.

An interpretation for a predicate logic expression consists of: a domain for each variable in the expression a predicate for each predicate symbol in the expression a function for each function symbol in the expression. Note that the propositional operators are not counted as function symbols in the case of predicate logic, even though they represent functions. The reason for this is that we do not wish to subject them to interpretations other than the usual propositional interpretation. Also, we have already said that predicates are a type of function. However, we distinguish them in predicate logic so as to separate predicates, which have truth values used by propositional operators, from functions that operate on arbitrary domains.

Furthermore, as with proposition logic, the stand-alone convention applies with predicates. We do not usually explicitly indicate == 1 when a predicate expression is true; rather we just write the predicate along with its arguments, standing alone. An assignment for a predicate logic expression consists of a value for each variable in the expression. Given an assignment, a truth value is obtained for the entire expression in the natural way.

*Example:*

Consider the expression:

```
x < y || (y < z && z < x)
^ ^ ^ predicate symbols
```

Here || and && are propositional operators and < is a predicate symbol (in infix notation). An assignment is a particular predicate, say the less_than predicate on natural numbers, and values for x, y, and z, say 3, 1, and 2. With respect to this assignment then, the value is that of

**Notes**

```
3 < 1 || (1 < 2 && 2 < 3)
```

which is false || (true && true) i.e. true.

With respect to the same assignment for <, but 3, 2, 1 for x, y, z, the value would be that of

3 < 2 || (2 < 1 && 1 < 3) which would be false. As long as we have assigned meanings to all variables and predicates in the expression, we can derive a false or true value. Now, we give an example where function symbols, as well as predicate symbols, are present.

((u + v) < y) || ((y < (v + w)) && v < x) ^ ^ function symbols would be an example of an expression with both function and predicate symbols. If we assign + and < their usual meanings and u, v, w, x, y the values 1, 2, 3, 4, 5 respectively, this would evaluate to the value of ((1 + 2) < 4) || ((4 < (2 + 3)) && 2 < 4) which is, of course, true. It is common to be concerned with a fixed interpretation (of domains, predicates, and functions) and allow the assignment to vary over individuals in a domain. If a formula evaluates to true for all assignments, it is called valid with respect to the interpretation.

If a formula is valid with respect to every interpretation, it is called valid. A special case of validity is where sub-expressions are substituted for proposition symbols in a tautology. These are also called tautologies.

The traditional conditional construct in Lisp is `cond`. However, `if` is much simpler and is directly comparable to conditional constructs in other programming languages, so it is considered to be primitive in Common Lisp and is described first. Common Lisp also provides the dispatching constructs `case` and `typecase`, which are often more convenient than `cond`.

*Special Form*

`if` *test then* [else]

The `if` special form corresponds to the if-then-else construct found in most algebraic programming languages. First the form test is evaluated. If the result is not `nil`, then the form then is selected; otherwise the form else is selected. Whichever form is selected is then evaluated, and `if` returns whatever is returned by evaluation of the selected form.

```
(if test then else) == (cond (test then) (t else))
```

but `if` is considered more readable in some situations.

The else form may be omitted, in which case if the value of test is `nil` then nothing is done and the value of the `if` form is `nil`. If the value of the `if` form is important in this situation, then the `and` construct may be stylistically preferable, depending on the context. If the value is not important, but only the effect, then the `when` construct may be stylistically preferable.

### Self Assessment

State whether the following statements are true or false:

4. Predicate logic builds heavily upon the ideas of proposition logic to provide a more powerful system for expression and reasoning.
5. Predicate logic does not deal with the combination of predicates using the propositional operators.
6. The traditional conditional construct in Lisp is cond.

## 5.3 Input, Output and Local Variables

Variables in programming languages are simply containers which hold values. Access to variables and the contents of them is through their name, also know as their identifiers.

Notes

Inputs are variables that are created and filled with values when a procedure is invoked. The names of the variables, are given on the procedure's title line. These identifiers can be used in the instructions which make up the body of the procedure. Inputs are also called local variables – they are local to the procedure in which they are defined.

Just to refresh your memory, given:

```
to box:size
repeat 4 [ forward:size right 90 ]
  end
box 100
```

The first three lines are the definition of the procedure box, which includes an input, :size. A value is placed in the variable (think container) named size when the procedure box is invoked. The last line in this example shows an invocation of box. In this case, 100 is the value that's being supplied for the*: size* variable.

### Self Assessment

State whether the following statements are true or false:

7. Variables in programming languages are simply containers which hold values.
8. Outputs are variables that are created and filled with values when a procedure is invoked.
9. Inputs are also called local variables.

## 5.4 Iteration and Recursion

The fact is that recursion is rarely the most efficient approach to solving a problem, and iteration is almost always more efficient. This is because there is usually more overhead associated with making recursive calls due to the fact that the call stack is so heavily used during recursion. This means that many computer programming languages will spend more time maintaining the call stack then they will actually performing the necessary calculations. Generally speaking, yes it does. This is because of the extensive use of the call stack. Recursion is generally used because of the fact that it is simpler to implement, and it is usually more 'elegant' than iterative solutions. Remember that anything that's done in recursion can also be done iteratively, but with recursion there is generally a performance drawback. But, depending on the problem that you are trying to solve, that performance drawback can be very insignificant – in which case it makes sense to use recursion. With recursion, you also get the added benefit that other programmers can more easily understand your code – which is always a good thing to have.

#### Example of Recursion in Java – the Factorial

We take a simple example of recursion to best illustrate how it works: the factorial is probably the most commonly used example. What is a factorial? Well, any number written like this: "x!" is said to be "the factorial of x". A factorial of a number "x" is just the product of all integers between 1 and x. So, if x is equal to the number 5, then the factorial of x would be 5*4*3*2*1, which equals 120. We could also say that the factorial of is equal to 5 multiplied by the factorial of 4, which would be 5 * 4!, or 5*4*3*2*1 So, the factorial of any number "x" could also be defined as:

```
x! = x * (x - 1)!
```

And, something else that is important to know is the fact that the factorial of 0 is equal to 1 as is the factorial of 1.

```
0! = 1! = 1
```

Notes

### Self Assessment

State whether the following statements are true or false:

10. Recursion is rarely the most efficient approach to solving a problem.
11. With recursion, you also get the added benefit that other programmers can more easily understand.
12. A factorial of a number "x" is just the product of all integers between 0 and x.

## 5.5 Property Lists and Arrays

Arrays are extremely useful when you have several similar objects that you want to be able to process the same way or to be able to process as a group rather than individually. The Array object itself provides a number of methods that make manipulation of the whole group of objects or specific objects within the group much easier. The following methods are provided by Java script to assist in the manipulation of arrays:

- ***concat(secondarray)*** will concatenate the elements of the second array onto the end of the original array effectively joining two arrays together.
- ***join(separator)*** converts the array into a string by placing the contents of the elements one after the other with the specified separator in between. If no separator is specified then a comma will be used.
- ***pop()*** removes the last element from the array making the array smaller. The removed element is returned from the method call allowing it to be processed.
- ***push(element)*** adds an element to the end of the array.
- ***reverse()*** reverses the order of the elements within the array.
- ***shift()*** similar to pop() but removes and returns the first element in the array rather than the last.
- ***slice(start,end)*** creates a new array containing a subset of the elements from the original array. The start and end values are numbers indicating which elements to extract counting from the start of the array if the numbers are positive or from the end of the array if they are negative. If an end value is omitted then the elements from the start position to the end of the array will be extracted.
- ***sort(compareFunction)*** will sort the elements in the array into any desired order. If no compare function is specified then the elements will be treated as strings and sorted into ascending order. By providing an appropriate compare function you can set whatever sort order you require.
- ***splice(position,number,element1,...)*** allows you to insert, delete, or replace elements from anywhere within the array. The specified number of elements from the specified position will be removed from the array and will be replaced with the elements specified in the third and subsequent parameters.
- ***unshift(element)*** similar to push except that the new element gets added to the start rather than the end of the array.

All of the above methods can be used for conventional arrays where the elements of the arrays are numbered. Such conventional arrays also have a length property that can be used to determine the number of elements that are currently allocated to the array. There is a second type of array known as an associative array where the elements of the array are named rather than numbered.

Notes

An associative array in Java script is primarily an alternative notation for being able to reference an Object and so the above methods and property cannot be used with an associative array. As the Array class is based on the Object class in Java script all of the methods and properties of the Object class are also available to use with Arrays and function the same way for Arrays as they do for Objects.

*Caution* Have proper knowledge of array and its applications.

*Task* Go through various websites for the iteration and recursion related with AI.

### Self Assessment

State whether the following statements are true or false:

13. Arrays are extremely useful when you have several similar objects that you want to be able to process the same way or to be able to process as a group rather than individually.
14. Concat(secondarray) adds an element to the end of the array.
15. Pop() removes the first element from the array making the array smaller.

## 5.6 Prolog and Other AI Programming Languages

In the 1970s, an alternative paradigm for symbolic computation and AI programming arose from the success in the area of automatic theorem proving. Notably, the resolution proof procedure developed by Robinson (1965) showed that formal logic, in particular predicate calculus, could be used as a notation for defining algorithms and therefore, for performing symbolic computations. In the early 1970s, Prolog (an acronym for Programming in Logic), the first logical based programming language appeared. Basically, Prolog consists of a method for specifying predicate calculus propositions and a restricted form of resolution. Programming in Prolog consists of the specification of facts about objects and their relationships, and rules specifying their logical relationships. Prolog programs are declarative collections of statements about a problem because they do not specify how a result is to be computed but rather define what the logical structure of a result should be. This is quite different from imperative and even functional programming, in which the focus is on defining how a result is to be computed. Using Prolog, programming can be done at a very abstract level quite close to the formal specification of a problem. Prolog is still the most important logical programming language. There are a number of commercial programming systems on the market which include modern programming modules, i.e., compiler, debugger and visualization tools. Prolog has been used successfully in a number of AI areas such as expert systems and natural language processing, but also in such areas as relational database management systems or in education.

*Caution* A query is used to activate Prolog's proof procedure.

Logically, a query corresponds to an unknown theorem. It has the same form as a fact. In order to tell Prolog that a query has to be proven, the special query operator ? - is usually written in front of the query. In the simple Prolog program introduced above, we have already seen an informal description of how a query is used by Prolog. Prolog's inference process consists of two basic components: a search strategy and a unifier. The search strategy is used to search

Notes

through the fact and rule data base while unification is used for pattern matching and returns the bindings that make an expression true.

*Notes* The unifier is applied on two terms and tries to combine them both to form a new term. If unification is not possible, then unification is said to have failed. If the two terms contain no variables, then unification actually reduces to checking whether the terms are equal.

Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. Prolog was one of the first logic programming languages and remains the most popular among such languages today with many free and commercial implementations available. While initially aimed at natural language processing, the language has since then stretched far into other areas like theorem proving, expert systems, games, automated answering systems, ontologism and sophisticated control systems. Modern Prolog environments support creating graphical user interfaces, as well as administrative and networked applications.

### 5.6.1 Prolog Lists

Lists themselves have the following syntax. They always start and end with square brackets, and each of the items they contain is separated by a comma. Here is a simple list [a,freddie,A_Variable,apple]

Prolog also has a special facility to split the first part of the list (called the head) away from the rest of the list (known as the tail). We can place a special symbol | (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list. For example, consider the following.

[first,second,third] = [A | B]

where *A = first* and *B=[second,third]*

The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.

*Example 1:* Here are some example simple lists

[a,b,c,d,e,f,g]

[apple,pear,bananas,breadfruit]

[ ]/* this is a special list, it is called the empty list because it contains nothing */

Now lets consider some comparisons of lists:

[a,b,c] unifies with [Head | Tail] resulting in Head=a and Tail=[b,c]

[a] unifies with [H | T] resulting in H=a and T=[]

[a,b,c] unifies with [a | T] resulting in T=[b,c]

[a,b,c] doesn't unify with [b | T]

It doesn't unify with [H | T]

It unifies with []. Two empty lists always match

Notes

*Example 2:* Consider the following fact.

p([H|T], H, T).

Lets see what happens when we ask some simple queries.

?- p([a,b,c], X, Y).

X=a

Y=[b,c]

yes

?- p([a], X, Y).

X=a

Y=[ ]

yes

?- p([], X, Y).

no

### 5.6.2 Prolog Sequences

The kind of sequences most used in Prolog is "comma" sequences. There is no empty sequence (unlike for lists). The shortest sequence has one element. Longer sequences have elements separated by commas ",". An appropriate declaration for the comma operator would be

:- op(1000,xfy,',').

meaning that comma is right-associative with precedence 1000. (The comma operator is actually built-in.) Here is some Prolog behavior.

```
?- (H,T) = (1,2,3,4).
H = 1
T = 2,3,4

?- (a) = a.
Yes

?- (H,T) = (a).
No

?- (A,B,C) = (1,2,3,4,5).
A = 1
B = 2
C = 3,4,5
```

Prolog clauses use comma sequences.

```
?- assert((a(X):- b(X),c(X),d(X))). %% Note parens around clause
X = G1056

?- clause(a(X),Body), Body=(First,Next).
First = b(G1176)
Next = c(G1176), d(G1176)
Body = b(G1176), c(G1176), d(G1176)
X = G1176
```

Processing sequences is similar to processing lists, except that the base case for sequences is a unit sequence (one element), whereas for lists the base case is for the empty list.

Notes

*Example:* Here is a program to append comma sequences ...

```
sequence_append((X,R),S,(X,T)):-
!,
sequence_append(R,S,T).
sequence_append((X),S,(X,S)).
```

Note the use of cut (!) to make sure that the second clause is not available as a alternate choice for multi-element sequences.

```
?- sequence_append((1,2,3),(a,b,c,d),S).
S = 1, 2, 3, a, b, c, d
```

### 5.6.3 Prolog as a Relational Language

A Probabilistic Relational Programming Language (PRPL) is a programming language specially designed to describe and infer with Probabilistic Relational Models (PRMs). A PRM is usually developed with a set of algorithms for reducing, inference about and discovery of concerned distributions, which are embedded into the corresponding PRPL. PRPLs often extend from a basic language. The inventors' choices of underlying basic language depend on the similarity of their relational models to the basic language's relational model, as well as commercial considerations and personal preference. For instance, Infer.NET is based on .NET framework, while PRISM extends from Prolog. Currently there are several PRPLs in active development; some of them have advanced to the beta stage. However because PRMs are new, up to year 2010 there have been no well-known software projects utilizing those languages.

We have introduced Lisp as the main representative functional programming language (especially the widely used dialect Common Lisp), because it is still a widely used programming language for a number of Artificial Intelligence problems, like Natural Language Understanding, Information Extraction, Machine Learning, AI planning, or Genetic Programming. Beside Lisp a number of alternative functional programming languages have been developed. We will briefly mention two well-known members, viz. ML and Haskell. ML which stands for Meta-Language is a static-scoped functional programming language.

The main differences to Lisp is its syntax (which is more similar to that of Pascal), and a strict polymorphic type system (i.e., using strong types and type inference, which means that variables need not be declared). The type of each declared variable and expression can be determined at compile time. ML supports the definition of abstract data types, as demonstrated by the following

*Example:*

datatype tree = L of int

| int * tree * tree; which can be read as "every binary tree is either a leaf containing an integer or it is a node containing an integer and two trees (the subtrees)". An example of a recursive function definition applied on a tree data structure is shown in the next example:

fun depth(L) = 1

| depth(N(i,l,r)) =

1 + max(depth l, depth r);

The function depth maps trees to integers. The depth of a leaf is 1 and the depth of any other tree is 1 plus the maximum of the depths of the left and right sub trees.

Haskell is similar to ML: it uses a similar syntax, it is also static scoped, and makes use of the same type inference method. It differs from ML in that it is purely functional. This means that it

Notes

allows no side effects and includes no imperative features of any kind, basically because it has no variables and no assignment statements. Furthermore, it uses a lazy evaluation technique, in which no sub expression is evaluated until its value is known to be required. Lists are a commonly used data structure in Haskell.

*Example:* [1,2,3] is the list of three integers 1,2, and 3. The list [1,2,3] in Haskell is actually shorthand for the list 1:(2:(3:[])), where [] is the empty list and: is the infix operator that adds its first argument to the front of its second argument (a list). As an example of a user-defined function that operates on lists, consider the problem of counting the number of elements in a list by defining the function length:

length:: [a] -> Integer

length [ ] = 0

length (x:xs) = 1 + length xs; which can be read as "The length of the empty list is 0, and the length of a list whose first element is x and remainder is xs is 1 plus the length of xs". In Haskell, function invocation is guided by pattern matching. For example, the left-hand sides of the equations contain patterns such as [] and x:xs. In a function application, these patterns are matched against actual parameters ([]) only matches the empty list, and x:xs will successfully match any list with at least one element, binding x to the first element and xs to the rest of the list). If the match succeeds, the right hand side is evaluated and returned as the result of the application. If it fails, the next equation is tried, and if all equations fail, an error results.

*Caution* Use Prolog after creating Algorithm.

*Task* Use Prolog for any general programs.

## Self Assessment

State whether the following statements are true or false:

16. In the 1970s an alternative paradigm for symbolic computation and AI programming arose from the success in the area of automatic theorem proving.
17. A query is used to activate Prolog's proof procedure.
18. Prolog was not the first logic programming languages.

## 5.7 Summary

- Programming languages in artificial intelligence (AI) are the major tool for exploring and building computer programs that can be used to simulate intelligent processes such as learning, reasoning and understanding symbolic information in context.
- The AI programming benefits considerably if the programming language frees the AI programmer from the constraints of too many technical constructions.
- The input-output functions print, prinl, princ, read, terpri, and format were defined and example programs presented.

Notes

- The data and functions are represented as s – expressions which can be either atoms or lists.
- The prog function is similar to let in that the first arguments following it are a list of local variables where each element is either a variable name or a list containing a variable name and its initial value.
- PROLOG is a logical and a declarative programming language. The name itself, PROLOG, is short for Programming in Logic.

## 5.8 Keywords

***Artificial Intelligence:*** It is a major tool for exploring and building computer programs.

***Predicates:*** It is a function that tests their arguments for some specific condition.

***Processing Sequences:*** It is similar to processing lists.

***PROLOG:*** It refers to interpreted language.

***Setf:*** It is an assignment function.

## 5.9 Review Questions

1. Explain LISP.
2. What do you understand by Prolog?
3. Define predicate and conditional.
4. Discuss Prolog as a Relational Language.
5. Explain with the help of example how Prolog clauses use comma sequences.
6. Write brief note on Property Lists and Arrays.

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | False | 2. | True |
| 3. | False | 4. | True |
| 5. | False | 6. | True |
| 7. | True | 8. | False |
| 9. | False | 10. | True |
| 11. | True | 12. | False |
| 13. | True | 14. | False |
| 15. | False | 16. | True |
| 17. | True | 18. | False |

## 5.10 Further Readings

Notes

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://programmers.stackexchange.com/questions/84407/why-is-prolog-good-for-ai-programming

http://stackoverflow.com/questions/82036/what-is-a-good-programming-language-for-ai

http://www.bracil.net/Teaching/Intro_AI/downloads/Intro_AI-001.pdf

http://www.cs.millersville.edu/~chaudhary/340/AIThruProlog.pdf

http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/intro.html

http://www.engin.umd.umich.edu/CIS/course.des/cis400/prolog/prolog.html

http://www.inf.ed.ac.uk/teaching/courses/aipp/

http://www.programmerinterview.com/index.php/recursion/explanation-of-recursion/

https://news.ycombinator.com/item?id=3944506

Notes

# Unit 6: Formalized Symbolic Logics

**CONTENTS**



*Contd...*

Notes

## Objectives

After studying this unit, you will be able to:

- Explain the concept of Formalized Symbolic Logics
- Discuss Syntax and Semantics for First-order Logic (FOL)
- Describe the Well-Formed Formula (WFF)
- Explain Conversion to Clausal Form
- Identify the standard form of Rules of Inference
- Discuss the concept of Resolution Logic
- Identify various Deductive Inference Methods
- Describe Truth Maintenance System
- Discuss the Predicated Completion and Circumscription
- Define Modal Logic
- Define Temporal Logic
- Explain the concept of Fuzzy Logic
- Discuss the Natural Language Computation

## Introduction

Mathematical logic (also symbolic logic, formal logic, or, less frequently, modern logic) is a subfield of mathematics with close connections to the foundations of mathematics, theoretical computer science and philosophical logic The field includes both the mathematical study of logic and the applications of formal logic to other areas of mathematics. The unifying themes in mathematical logic include the study of the expressive power of formal systems and the deductive power of formal proof systems.

Mathematical logic is often divided into the fields of set theory, model theory, recursion theory, and proof theory. These areas share basic results on logic, particularly first-order logic, and definability. In computer science (particularly in the ACM Classification), mathematical logic encompasses additional topics not detailed in this article; see logic in computer science for those.

Since its inception, mathematical logic has both contributed to, and has been motivated by, the study of foundations of mathematics. This study began in the late 19th century with the development of axiomatic frameworks for geometry, arithmetic, and analysis. In the early 20th century, it was shaped by David Hilbert's program to prove the consistency of foundational

Notes

theories. Results of Kurt Gödel, Gerhard Gentzen, and others provided partial resolution to the program, and clarified the issues involved in proving consistency. Work in set theory showed that almost all ordinary mathematics can be formalized in terms of sets, although there are some theorems that cannot be proven in common axiom systems for set theory. Contemporary work in the foundations of mathematics often focuses on establishing which parts of mathematics can be formalized in particular formal systems (as in reverse mathematics) rather than trying to find theories in which all of mathematics can be developed.

## 6.1 Concept of Formalized Symbolic Logics

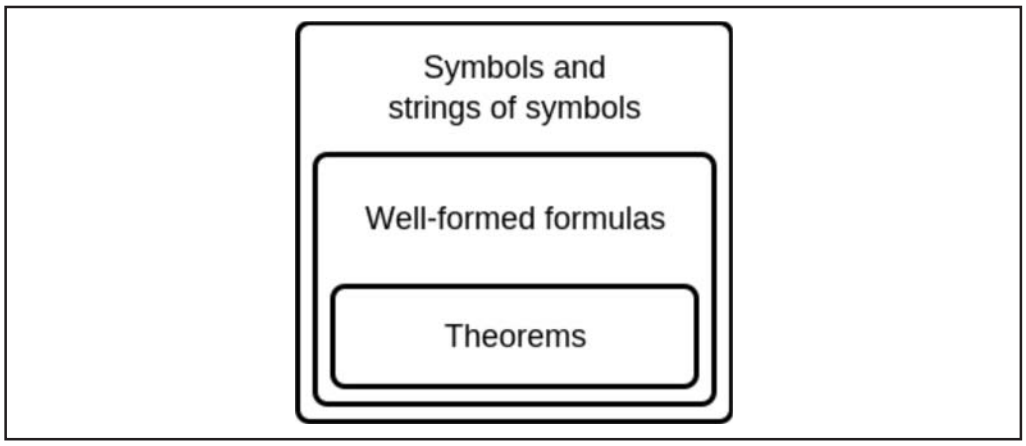
Mathematical logic emerged in the mid-19th century as a subfield of mathematics independent of the traditional study of logic. Before this emergence, logic was studied with rhetoric, through the syllogism, and with philosophy. The first half of the 20th century saw an explosion of fundamental results, accompanied by vigorous debate over the foundations of mathematics. Theories of logic were developed in many cultures in history, including China, India, Greece and the Islamic world. In 18th century Europe, attempts to treat the operations of formal logic in a symbolic or algebraic way had been made by philosophical mathematicians including Leibniz and Lambert, but their labors remained isolated and little known. In the middle of the 19th century, George Boole and then Augustus De Morgan presented systematic mathematical treatments of logic. Their work, building on work by algebraists such as George Peacock, extended the traditional Aristotelian doctrine of logic into a sufficient framework for the study of foundations of mathematics. Charles Sanders Peirce built upon the work of Boole to develop a logical system for relations and quantifiers, which he published in several papers from 1870 to 1885. Gottlob Frege presented an independent development of logic with quantifiers in his *Begri ffsschrift*, published in 1879, a work generally considered as marking a turning point in the history of logic. Frege's work remained obscure, however, until Bertrand Russell began to promote it near the turn of the century. The two-dimensional notation Frege developed was never widely adopted and is unused in contemporary texts. From 1890 to 1905, Ernst Schröder published *Vorlesungen über die Algebra der Logik* in three volumes. This work summarized and extended the work of Boole, De Morgan, and Peirce, and was a comprehensive reference to symbolic logic as it was understood at the end of the 19th century.

### 6.1.1 Syntax Logic

In logic, syntax is anything having to do with formal languages or formal systems without regard to any interpretation or meaning given to them. Syntax is concerned with the rules used for constructing, or transforming the symbols and words of a language, as contrasted with the semantics of a language which is concerned with its meaning. The symbols, formulas, systems, theorems, proofs, and interpretations expressed in formal languages are syntactic entities whose properties may be studied without regard to any meaning they may be given, and, in fact, need not be given any. Syntax is usually associated with the rules (or grammar) governing the composition of texts in a formal language that constitute the well-formed formulas of a formal system.

*Did u know?* In computer science, the term "syntax" refers to the rules governing the composition of meaningful texts in a formal language, such as a programming language, that is, those texts for which it makes sense to define the semantics or meaning, or otherwise provide an interpretation.

Notes




This diagram shows the syntactic entities which may be constructed from formal languages. The symbols and strings of symbols may be broadly divided into nonsense and well-formed formulas. A formal language is identical to the set of its well-formed formulas. The set of well-formed formulas may be broadly divided into theorems and non-theorems.

### 6.1.2 Propositional Logic

A proposition is a sentence expressing something true or false. A proposition is identified ontologically as an idea, concept or abstraction whose token instances are patterns of symbols, marks, sounds, or strings of words. Propositions are considered to be syntactic entities and also truth bearers. A formal theory is a set of sentences in a formal language. A symbol is an idea, abstraction or concept, tokens of which may be marks or a configuration of marks which form a particular pattern. Symbols of a formal language need not be symbols of anything. For instance, there are logical constants which do not refer to any idea, but rather serve as a form of punctuation in the language (e.g. parentheses). A symbol or string of symbols may comprise a well-formed formula if the formulation is consistent with the formation rules of the language. Symbols of a formal language must be capable of being specified without any reference to any interpretation of them.

#### Formal Language

A formal language is a syntactic entity which consists of a set of finite strings of symbols which are its words (usually called its well-formed formulas). Which strings of symbols are words is determined by fiat by the creator of the language, usually by specifying a set of formation rules. Such a language can be defined without reference to any meanings of any of its expressions; it can exist before any interpretation is assigned to it – that is, before it has any meaning. Formation rules are a precise description of which strings of symbols are the well-formed formulas of a formal language. It is synonymous with the set of strings over the alphabet of the formal language which constitute well formed formulas. However, it does not describe their semantics (i.e. what they mean).

#### Formal Systems

A formal system (also called a logical calculus, or a logical system) consists of a formal language together with a deductive apparatus (also called a deductive system). The deductive apparatus may consist of a set of transformation rules (also called inference rules) or a set of axioms, or have both. A formal system is used to derive one expression from one or more other expressions.

Notes

Formal systems, like other syntactic entities may be defined without any interpretation given to it (as being, for instance, a system of arithmetic). A formula A is a syntactic consequence within some formal system $\mathcal{FS}$ of a set Γ of formulas if there is a derivation in formal system $\mathcal{FS}$ of A from the set Γ.

$$\Gamma \vdash_{\mathcal{FS}} A$$

Syntactic consequence does not depend on any interpretation of the formal system.

### Syntactic Completeness of a Formal System

A formal system $\mathcal{S}$ is syntactically complete (also deductively complete, maximally complete, negation complete or simply complete) if for each formula A of the language of the system either A or ¬A is a theorem of $\mathcal{S}$. In another sense, a formal system is syntactically complete if no unprovable axiom can be added to it as an axiom without introducing an inconsistency. Truth-functional propositional logic and first-order predicate logic are semantically complete, but not syntactically complete (for example the propositional logic statement consisting of a single variable "a" is not a theorem, and neither is its negation, but these are not tautologies). Gödel's incompleteness theorem shows that no recursive system that is sufficiently powerful, such as the Peano axioms, can be both consistent and complete.

### Self Assessment

State whether the following statements are true or false:

1. A formal systems is a syntactic entity which consists of a set of finite strings of symbols which are its words.
2. Formation rules are a precise description of which strings of symbols are the well-formed formulas of a formal language.

## 6.2 Syntax and Semantics for First-order Logic (FOL)

First-order logic is a formal system used in mathematics, philosophy, linguistics, and computer science. It is also known as first-order predicate calculus, the lower predicate calculus, quantification theory, and predicate logic (a less precise term). First-order logic is distinguished from propositional logic by its use of quantified variables. A theory about some topic is usually first-order logic together with a specified domain of discourse over which the quantified variables range, finitely many functions which map from that domain into it, finitely many predicates defined on that domain, and a recursive set of axioms which are believed to hold for those things. Sometimes "theory" is understood in a more formal sense, which is just a set of sentences in first-order logic.

The adjective "first-order" distinguishes first-order logic from higher-order logic in which there are predicates having predicates or functions as arguments, or in which one or both of predicate quantifiers or function quantifiers are permitted. In first-order theories, predicates are often associated with sets. In interpreted higher-order theories, predicates may be interpreted as sets of sets. There are many deductive systems for first-order logic that are sound (all provable statements are true) and complete (all true statements are provable). Although the logical consequence relation is only semi decidable, much progress has been made in automated theorem proving in first-order logic. First-order logic also satisfies several met logical theorems that make it amenable to analysis in proof theory, such as the Löwenheim – Skolem theorem and the compactness theorem. First-order logic is of great importance to the foundations of mathematics, because it is the standard formal logic for axiomatic systems. Many common axiomatic systems,

Notes

such as first-order Peano arithmetic and axiomatic set theory, including the canonical Zermelo – Fraenkel set theory (ZF), can be formalized as first-order theories.

*Notes* No first-order theory, however, has the strength to describe fully and categorically structures with an infinite domain, such as the natural numbers or the real line. Categorical axiom systems for these structures can be obtained in stronger logics such as second-order logic.

Propositional Logic is concerned with propositions and their interrelationships. The notion of a proposition here cannot be defined precisely. Roughly speaking, a proposition is a possible condition of the world about which we want to say something. The condition need not be true in order for us to talk about it. In fact, we might want to say that it is false or that it is true if some other proposition is true. In this unit, we first look at the syntactic rules for the language of Propositional Logic. We then look at semantic interpretation for the expressions specified by these rules. Given this semantics, we define the concept of propositional entailment, which identifies for us, at least in principle, all of the logical conclusions one can draw from any set of propositional sentences.

## Syntax

In Propositional Logic, there are two types of sentences – simple sentences and compound sentences. Simple sentences express "atomic" propositions about the world. Compound sentences express logical relationships between the simpler sentences of which they are composed. Simple sentences in Propositional Logic are often called propositional constants or, sometimes, logical constants. In what follows, we refer to a logical constant using a sequence of alphanumeric characters beginning with a lower case character. For example, raining is a logical constant, as are rAiNiNg and r32aining. Raining is not a logical constant because it begins with an upper case character. 324567 fails because it begins with a number. Raining-or-snowing fails because it contains non-alphanumeric characters. Compound sentences are formed from simpler sentences and express relationships among the constituent sentences. There are six types of compound sentences, viz. negations, conjunctions, disjunctions, implications, reductions, and equivalences. A negation consists of the negation operator ¬ and a simple or compound sentence, called the target. For example, given the sentence $p$, we can form the negation of $p$ as shown below:

$$\neg p$$

A conjunction is a sequence of sentences separated by occurrences of the $\wedge$ operator and enclosed in parentheses, as shown below. The constituent sentences are called conjuncts. For example, we can form the conjunction of $p$ and $q$ as follows:

$$(p \wedge q)$$

A disjunction is a sequence of sentences separated by occurrences of the $\vee$ operator and enclosed in parentheses. The constituent sentences are called disjuncts. For example, we can form the disjunction of $p$ and $q$ as follows:

$$(p \vee q)$$

An implication consists of a pair of sentences separated by the $\Rightarrow$ operator and enclosed in parentheses. The sentence to the left of the operator is called the antecedent, and the sentence to the right is called the consequent. The implication of $p$ and $q$ is shown below:

$$(p \Rightarrow q)$$

A reduction is the reverse of an implication. It consists of a pair of sentences separated by the $\Leftarrow$ operator and enclosed in parentheses. In this case, the sentence to the left of the operator is called

**Notes**

the consequent, and the sentence to the right is called the antecedent. The reduction of $p$ to $q$ is shown below:

$$(p \Leftarrow q)$$

Equivalence is a combination of an implication and a reduction. For example, we can express the equivalence of $p$ and $q$ as shown below:

$$(p \Leftrightarrow q)$$

Note that the constituent sentences within any compound sentence can be either simple sentences or compound sentences or a mixture of the two. For example, the following is a legal compound sentence.

$$((p \vee q) \Rightarrow \neg r)$$

One disadvantage of our notation, as written, is that the parentheses tend to build up and need to be matched correctly. It would be nice if we could dispense with parentheses, e.g. simplifying the preceding sentence to the one shown below:

$$p \vee q \Rightarrow \neg r$$

Unfortunately, we cannot do without parentheses entirely, since then we would be unable to render certain sentences unambiguously. For example, the sentence shown above could have resulted from dropping parentheses from either of the following sentences:

$$((p \vee q) \Rightarrow \neg r)$$
$$(p \vee (q \Rightarrow \neg r))$$

The solution to this problem is the use of operator precedence. The following gives a hierarchy of precedence for our operators.

The $\neg$ operator has higher precedence than $\wedge$; $\wedge$ has higher precedence than $\vee$; and $\vee$ has higher precedence than $\Rightarrow$, $\Leftarrow$, and $\Leftrightarrow$.

$$\neg$$
$$\wedge$$
$$\vee$$
$$\Rightarrow \Leftarrow \Leftrightarrow$$

In unparenthesized sentences, it is often the case that an expression is flanked by operators, one on either side. In interpreting such sentences, the question is whether the operator associates with the operator on its left or the one on its right. We can use precedence to make this determination. In particular, we agree that an operand in such a situation always associates with the operator of higher precedence. When an operand is surrounded by operators of equal precedence, the operand will be associated to the left. The following examples show how these rules work in various cases. The expressions on the right are the fully parenthesized versions of the expressions on the left.

| | |
|---|---|
| $\neg p \wedge q$ | $(\neg p \wedge q)$ |
| $p \wedge \neg q$ | $(p \wedge \neg q)$ |
| $p \wedge q \vee r$ | $((p \wedge q) \vee r)$ |
| $p \vee q \wedge r$ | $(p \vee (q \wedge r)$ |
| $p \Rightarrow q \Rightarrow r$ | $((p \Rightarrow q) \Rightarrow r)$ |
| $p \Rightarrow q \Leftarrow r$ | $((p \Rightarrow q) \Leftarrow r)$ |

Note that just because precedence allows us to delete parentheses in some case does not mean that we can dispense with parentheses entirely. Consider the example shown above. Precedence eliminates the ambiguity by dictating that the unparenthesized sentence is an implication with

Notes

a disjunction as antecedent. However, this makes for a problem for those cases when we want to express a disjunction with an implication as a disjunctive. In such cases, we must retain at least one pair of parentheses.

## Semantics

The semantics of logic is similar to the semantics of algebra. Algebra is unconcerned with the real-world meaning of variables like $x$. What is interesting is the relationship between the variables expressed in the equations we write; and algebraic methods are designed to respect these relationships, no matter what meanings or values are assigned to the constituent variables. In a similar way, logic itself is unconcerned with what sentences say about the world being described. What is interesting is the relationship between the truth of simple sentences and the truth of compound sentences within which the simple sentences are contained. Also, logical reasoning methods are designed to work no matter what meanings or values are assigned to the logical "variables" used in sentences. Although the values assigned to variables are not crucial in the sense just described, in talking about logic itself, it is sometimes useful to make these assignments explicit and to consider various assignments or all assignments and so forth. Such an assignment is called an interpretation. Formally, an interpretation for propositional logic is a mapping assigning a truth value to each of the simple sentences of the language. In what follows, we refer to the meaning of a constant or expression under an interpretation $i$ by superscripting the constant or expression with $i$ as the superscript.

The assignment shown below is an example for the case of a logical language with just three propositional constants, viz. $p$, $q$, and $r$.

$p^i$ = true

$q^i$ = false

$r^i$ = true

The following assignment is another interpretation for the same language.

$p^i$ = false

$q^i$ = false

$r^i$ = true

Note that the expressions above are not themselves sentences in Propositional Logic. Propositional Logic does not allow superscripts and does not use the = symbol. Rather, these are informal, Meta level statements about particular interpretations. Although talking about propositional logic using a notation similar to that propositional logic can sometimes be confusing, it allows us to convey meta-information precisely and efficiently. To minimize problems, in this book we use such meta-notation infrequently and only when there is little chance of confusion. Looking at the preceding interpretations, it is important to bear in mind that, as far as logic is concerned, any interpretation is as good as any other. It does not directly fix the interpretation of individual logical constants. On the other hand, given an interpretation for the logical constants of a language, logic does fix the interpretation for all compound sentences in that language. In fact, it is possible to determine the truth value of a compound sentence by repeatedly applying the following rules.

1. If the truth value of a sentence is true in an interpretation, the truth value of its negation is false. If the truth value of a sentence is false, the truth value of its negation is true.
2. The truth value of a conjunction is true under an interpretation if and only if the truth value of its conjuncts are both true; otherwise, the truth value is false.
3. The truth value of a disjunction is true if and only if the truth value of at least one its conjuncts is true; otherwise, the truth value is false. Note that this is the inclusive or

Notes

interpretation of the $\vee$ operator and is differentiated from exclusive or in which a disjunction is true if and only if an odd number of its disjuncts are false.

4. The truth value of an implication is false if and only if its antecedent is true and is consequent is false; otherwise, the truth value is true. This is called material implication.

5. As with an implication, the truth value of a reduction is false if and only if its antecedent is true and its consequent is false; otherwise, the truth value is true. Of course, it is important to remember that in a reduction the antecedent and consequent are reversed.

6. An equivalence is true if and only if the truth values of its constituents agree, i.e. they are either both true or both false.

We say that an interpretation *i* satisfies a sentence if and only if it is true under that interpretation.

### Evaluation

Given the semantic definitions in the last section, we can easily determine for any given interpretation whether or not any sentence is true or false under that interpretation. The technique is simple. We substitute true and false values for the propositional constants and replace complex expressions with the corresponding values, working from the inside out.

As an example, consider the interpretation *i* show below:

$p^i$ = true

$q^i$ = false

$r^i$ = true

We can see that *i* satisfies $(p \vee q) \wedge (\neg q \vee r)$.

$(p \vee q)$ '" $(\neg q \vee r)$

(true $\vee$ false) $\wedge$ ($\neg$false $\vee$ true)

true $\wedge$ ($\neg$false $\vee$ true)

true $\wedge$ (true $\vee$ true)

true $\wedge$ true

true

Now, consider interpretation *j* defined as follows:

$p^i$ = true

$q^i$ = false

$r^i$ = true

In this case, *j* does not satisfy $(p \vee q) \wedge (\neg q \vee r)$.

$(p \vee q) \wedge (\neg q \vee r)$

(true $\vee$ true) $\wedge$ ($\neg$true $\vee$ false)

true $\wedge$ ($\neg$true $\vee$ false)

true $\wedge$ (false $\vee$ false)

true $\wedge$ false

false

Using this technique, we can evaluate the truth of arbitrary sentences in our language. The cost is proportional to the size of the sentence.

Notes

## Reverse Evaluation

Reverse evaluation is the opposite of evaluation. We begin with one or more compound sentences and try to figure out which interpretations satisfy those sentences. One way to do this is using a truth table for the language. A truth table for a propositional language is a table showing all of the possible interpretations for the propositional constants in the language. The following figure shows a truth table for a propositional language with just three propositional constants. Each row corresponds to a single interpretation. The interpretations *i* and *j* correspond to the third and seventh rows of this table, respectively.

| *p* | *q* | *r* |
|---|---|---|
| true | true | true |
| true | true | false |
| true | false | true |
| true | false | false |
| false | true | true |
| false | true | false |
| false | false | true |
| false | false | false |

Note that, for a propositional language with *n* logical constants, there are *n* columns in the truth tables and $2^n$ rows. In doing reverse evaluation, we process input sentences in turn, for each sentence crossing out interpretations in the truth table that do not satisfy the sentence. The interpretations remaining at the end of this process are all possible interpretations of the input sentences.

## Validity, Satisfiability and Unsatisfiability

Evaluation and reverse evaluation are processes that involve specific sentences and specific interpretations. In computational logic, we are rarely concerned with specific interpretations; we are more interested in the properties of sentences that hold across interpretations. In particular, the notion of satisfaction imposes a classification of sentences in a language based on whether there are interpretations that satisfy that sentence. A sentence is valid if and only if it is satisfied by every interpretation. The following sentence is valid.

$$p \vee \neg p$$

A sentence is satisfiable if and only if it is satisfied by at least one interpretation. A sentence is falsifiable if and only if there is at least one interpretation that makes it false. We have already seen several examples of satisfiable and falsifiable sentences. A sentence is unsatisfiable if and only if it is not satisfied by any interpretation. The following sentence is unsatisfiable. No matter what interpretation we take, the sentence is always false.

$$p \Leftrightarrow \neg p$$

A sentence is contingent if and only if it is both satisfiable and falsifiable, i.e. it is neither valid nor unsatisfiable. In one sense, valid sentences and unsatisfiable sentences are useless. Valid sentences do not rule out any possible interpretations; unsatisfiable sentences rule out all interpretations; thus they say nothing about the world. On the other hand, from a logical perspective, they are extremely useful in that, as we shall see, they serve as the basis for legal transformations that we can perform on other logical sentences.

Notes

Note that we can easily check the validity, contingency, or unsatisfiability of a sentence can easily by looking at the truth table for the propositional constants in the sentence.

### Propositional Entailment

Validity, satisfiability, and unsatisfiability are properties of individual sentences. In logical reasoning, we are not so much concerned with individual sentences as we are with the relationships between sentences. In particular, we would like to know, given some sentences, whether other sentences are or are not logical conclusions. This relative property is known as logical entailment. When we are speaking about Propositional Logic, we use the phrase propositional entailment.

A set of sentences $\Delta$ logically entails a sentence $\varphi$ (written $\Delta \models \varphi$) if and only if every interpretation that satisfies $\Delta$ also satisfies $\varphi$. For example, the sentence $p$ logically entails the sentence $(p \vee q)$. Since a disjunction is true whenever one of its disjuncts is true, then $(p \vee q)$ must be true whenever $p$ is true. On the other hand, the sentence $p$ does not logically entail $(p \wedge q)$. A conjunction is true if and only if both of its conjuncts are true, and $q$ may be false. Of course, any set of sentences containing both $p$ and $q$ does logically entail $(p \wedge q)$. Note that the relationship of logical entailment is a logical one. Even if the premises of a problem do not logically entail the conclusion, this does not mean that the conclusion is necessarily false, even if the premises are true. It just means that it is possible that the conclusion is false. Once again, consider the case of $(p \wedge q)$. Although $p$ does not logically entail this sentence, it is possible that both $p$ and $q$ are true and, therefore, $(p \wedge q)$ is true. However, the logical entailment does not hold because it is also possible that $q$ is false and, therefore, $(p \wedge q)$ is false.

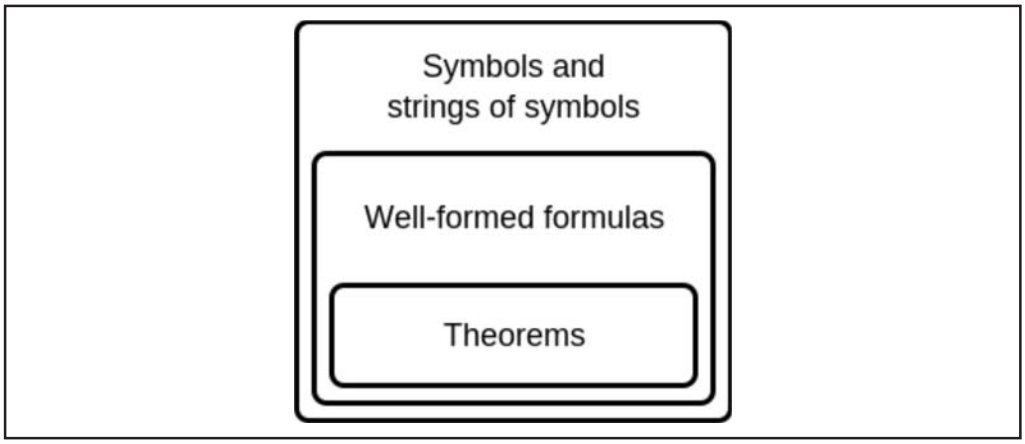
### Self Assessment

State whether the following statements are true or false:

3. Validity, satisfiability, and unsatisfiability are properties of individual sentences.
4. A sentence is falsifiable if and only if it is satisfied by at least one interpretation.

## 6.3 Well-formed Formula (WFF)

In mathematical logic, a well-formed formula, shortly WFF, often simply formula, is a word (i.e. a finite sequence of symbols from a given alphabet) which is part of a formal language. A formal language can be considered to be identical to the set containing all and only its formulas. A formula is a syntactic formal object that can be informally given a semantic meaning.



Notes

This diagram shows the syntactic entities which may be constructed from formal languages. The symbols and strings of symbols may be broadly divided into nonsense and well-formed formulas. A formal language can be thought of as identical to the set of its well-formed formulas. The set of well-formed formulas may be broadly divided into theorems and non-theorems. A key use of formulas is in propositional logic and predicate logics such as first-order logic. In those contexts, a formula is a string of symbols φ for which it makes sense to ask "is φ true?", once any free variables in φ have been instantiated. In formal logic, proofs can be represented by sequences of formulas with certain properties, and the final formula in the sequence is what is proven.

Although the term "formula" may be used for written marks (for instance, on a piece of paper or chalkboard), it is more precisely understood as the sequence being expressed, with the marks being a token instance of formula. It is not necessary for the existence of a formula that there be any actual tokens of it. A formal language may thus have an infinite number of formulas regardless whether each formula has a token instance. Moreover, a single formula may have more than one token instance, if it is written more than once. Formulas are quite often interpreted as propositions (as, for instance, in propositional logic). However formulas are syntactic entities, and as such must be specified in a formal language without regard to any interpretation of them. An interpreted formula may be the name of something, an adjective, an adverb, a preposition, a phrase, a clause, an imperative sentence, a string of sentences, a string of names, etc.

*Notes* A formula may even turn out to be nonsense, if the symbols of the language are specified so that it does. Furthermore, a formula need not be given *any* interpretation.

An expression $P(t_1,\ldots,t_n)$ where P is a predicate symbol of arity n and $t_1,\ldots t_n$ are terms is a WFF

- If S is a WFF then so is ¬S
- If $S_1$ and $S_2$ are WFFs then so is $S_1 \wedge S_2$
- If $S_1$ and $S_2$ are WFFs then so is $S_1 \wedge S_2$
- If $S_1$ and $S_2$ are WFFs then so is $S_1 \wedge S_2$
- If x is a variable and S is a WFF then so is $\forall$ x. S
  - We say that any occurrence of x in S is bound.
- If x is a variable and S is a WFF then so is $\exists$ x. S
  - We say that any occurrence of x in S is bound.

WFF may still contain unbound variables, for example:

- $\forall$ x. teaches(x, y) $\wedge$ Lecturer(x)
- This can cause us problems when we try and interpret them as it's not clear what we should do with the variable.
- To cope with this we define a further class of formulae known as sentences.
- A sentence is a WFF with no unbound (or free) variables.
  - Any variable occurring in the formula is bound by a quantifier

Notes

## 6.3.1 Properties of WFF

### Transforming to Clause Normal Form

There are several algorithms for this conversion, but they all have some parts in common, and all have similarities. The best algorithm is the one which produces the CNF which is most easily shown to be unsatisfiable. In general, such a determination is impossible to make, but a common measure is to aim to produce a CNF with the fewest symbols (where a symbol is a variable, predicate, or constant). The key feature of all the algorithms is that they are satisfiability preserving.

The starting point is a set of closed formulae. As a preprocessing step, all variables must be renamed apart.

### Rename Variables Apart

The first operation is to simplify the formulae so that they contain only the ∀, ∃, &, | and ~ connectives. This is done by using known logical identities. The identities are easily verified using truth tables, to show that the original formula has the same value as the simplified formula for all possible interpretations of the component formulae, denoted by uppercase variable letters *P* and *Q*.

**Simplify**

| Formula | Rewrites to |
|---|---|
| *P* <~> *Q* | (*P* \| *Q*) & (~*P* \| ~*Q*) |
| *P* <=> *Q* | (*P* => *Q*) & (*Q* => *P*) |
| *P* => *Q* | ~*P* \| *Q* |

The second operation is to move negations in so that they apply to only atoms. This is done by using known logical identities (including De Morgan's laws), which are easily verified as above. After moving negations in the formulae are in literal normal form.

**Move negations in**

| Formula | Rewrites to |
|---|---|
| ~∀X *P* | ∃X ~*P* |
| ~∃X *P* | ∀X ~*P* |
| ~(*P* & *Q*) | (~*P* \| ~*Q*) |
| ~(*P* \| *Q*) | (~*P* & ~*Q*) |
| ~~*P* | *P* |

The third operation is to move quantifiers out so that each formula is in the scope of all the quantifiers in that formula. This is done by using known logical identities, which are easily verified as above. In these identities, if the variable X already exists in *Q*, it is renamed to a new variable name that does not exist in the formula. After moving quantifiers out the formulae are in prenex normal form.

**Move quantifiers out**

| Formula | Rewrites to |
|---|---|
| ∀X *P* & *Q* | ∀X (*P* & *Q*) |
| ∃X *P* & *Q* | ∃X (*P* & *Q*) |

*Contd...*

Notes

| | |
|---|---|
| $Q \,\&\, \forall X\, P$ | $\forall X\, (Q \,\&\, P)$ |
| $Q \,\&\, \exists X\, P$ | $\exists X\, (Q \,\&\, P)$ |
| $\forall X\, P \mid Q$ | $\forall X\, (P \mid Q)$ |
| $\exists X\, P \mid Q$ | $\exists X\, (P \mid Q)$ |
| $Q \mid \forall X\, P$ | $\forall X\, (Q \mid P)$ |
| $Q \mid \exists XP$ | $\exists X\, (Q \mid P)$ |

The fourth operation is to skolemize to remove existential quantifiers. This step replaces existentially quantified variables by skolem functions and removes all quantifiers. The variables remaining after skolemization are all implicitly universally quantified. Whereas the previous three operations maintain equivalence between the original and transformed formulae, skolemization does not. However, Skolemization does maintain the satisfiability of the formulae, which is what is required for the overall goal of establishing logical consequence. After Skolemization the formulae are in Skolem normal form.

**Skolemize**

| Formula | Rewrites to |
|---|---|
| Initially | Set $\gamma$ = {} |
| $\forall X\, P$ | $P$<br>and set $\gamma = \gamma \cup \{X\}$ |
| $\exists X\, P$ | $P[X/x(\gamma)]$<br>where x is a new Skolem functor. |

The fifth operation is to distribute disjunctions so that the formulae are conjunctions of disjunctions. This is done by using known logical identities, which are easily verified as above. After distributing disjunctions the formulae are in conjunctive normal form.

**Distribute Disjunctions**

| Formula | Rewrites to |
|---|---|
| $P \mid (Q \,\&\, R)$ | $(P \mid Q) \,\&\, (P \mid R)$ |
| $(Q \,\&\, R) \mid P$ | $(Q \mid P) \,\&\, (R \mid P)$ |

The last operation is to convert to Clause Normal Form. This is done by removing the conjunctions, to form a set of clauses.

**Convert to CNF**

| Formula | Rewrites to |
|---|---|
| $P_1 \,\&\, \ldots \,\&\, P_n$ | $\{P_1, \ldots, P_n\}$ |

*Example:*

1st order formula

```
∀Y (∀X (taller(Y,X) | wise(X)) => wise(Y))
```

Simplify

```
∀Y (~∀X (taller(Y,X) | wise(X)) | wise(Y))
```

Move negations in

```
∀Y (∃X (~taller(Y,X) & ~wise(X)) | wise(Y))
```

Move quantifiers out

```
∀Y (∃X ((~taller(Y,X) & ~wise(X)) | wise(Y)))
```

Notes

Skolemize

```
∃X ((~taller(Y,X) & ~wise(X)) | wise(Y)) ã = {Y}
(~taller(Y,x(Y)) & ~wise(x(Y))) | wise(Y)
```

Distribute disjunctions

```
(~taller(Y,x(Y)) | wise(Y)) & (~wise(x(Y)) | wise(Y))
```

Convert to CNF

```
{ ~taller(Y,x(Y)) | wise(Y),
~wise(x(Y)) | wise(Y) }
```

*Caution* Use of truth table should be handled carefully.

*Task* Find the effect of propositional logic in real life.

## Self Assessment

State whether the following statements are true or false:

5. A formula is a syntactic formal object that can be informally given a semantic meaning.
6. A formal language may thus have an infinite number of formulas regardless whether each formula has a token instance.

## 6.4 Conversion to Clausal Form

### Clause Normal Form (CNF)

Clause Normal Form is a sub-language of 1st order logic. A clause is an expression of the form $L_1 | ... | L_m$ where each $L_i$ is a literal. Clauses are denoted by uppercase letters with a superscript |, e.g., $C^{|}$. There are satisfiability preserving transformations from 1st order logic to CNF, i.e., if a set of (1st order) formulae are satisfiable, then their CNF is satisfiable. Conversely, if the CNF of a set of formulae is unsatisfiable, then the formulae are unsatisfiable. This is then useful for showing logical consequence. The benefit of converting to CNF is that more is possible using just the Herbrand interpretations. Computationally, CNF is easier to work with, and is the form used by the resolution inference rule.

### 6.4.1 Conversion to Clausal Normal Form

Clauses may be expressed in Kowalski form. Kowalski form forms the antecedent of an implication by conjoining the atoms of the negative literals in a clause, and forms the consequent from the disjunction of the positive literals.

*Example 1:*

```
~taller(Y,x(Y)) | wise(Y) | ~wise(x(Y)) | taller(x(Y),Y)
```

is written as

```
taller(Y,x(Y)) & wise(x(Y)) => wise(Y) | taller(x(Y),Y)
```

If the antecedent is empty it is set to `TRUE`, and if the consequent is empty it is set to `FALSE`.

Notes

Horn clauses are clauses that have one or zero positive literals. Sets of Horn clauses are also called Horn.

*Example 2:*

```
~taller(Y,x(Y)) | wise(Y) | ~wise(x(Y))
~wise(geoff) | ~taller(Person,brother_of(jim))
```

are Horn clauses. Written in Kowalski form, these Horn clauses are:

```
taller(Y,x(Y)) & wise(x(Y)) => wise(Y)
wise(geoff) & taller(Person,brother_of(jim)) => FALSE
```

Non-Horn clauses are clauses that have two or more positive literals. Sets of clauses that contain one or more non-Horn clauses are also called non-Horn.

*Example 3:*

```
~taller(Y,x(Y)) | ~wise(Y) | wise(x(Y)) | taller(x(Y),x(x(Y)))
wise(X) | wise(brother_of(X))
```

are non-Horn clauses. Written in Kowalski form, these non-Horn clauses are:

```
taller(Y,x(Y)) & wise(Y) => wise(x(Y)) | taller(x(Y),x(x(Y)))
TRUE => wise(X) | wise(brother_of(X))
```

### Self Assessment

State whether the following statements are true or false:

7. Clause Normal Form (CNF) is a sub-language of 1st order logic.
8. Clauses are denoted by lowercase letters with a superscript.

## 6.5 Inference Rules

In terms of ATP, the disjunction of positive literals in non-Horn clauses corresponds to a choice point in any goal directed search. This is a source of search. Thus, in general, Horn problems are easier than non-Horn problems. In logic, a rule of inference, inference rule, or transformation rule is the act of drawing a conclusion based on the form of premises interpreted as a function which takes premises, analyzes their syntax, and returns a conclusion (or conclusions).

*Example:* The rule of inference modus ponens takes two premises, one in the form of "If *p* then *q*" and another in the form of "*p*" and returns the conclusion "*q*".

The rule is valid with respect to the semantics of classical logic (as well as the semantics of many other non-classical logics), in the sense that if the premises are true (under an interpretation) then so is the conclusion. Typically, a rule of inference preserves truth, a semantic property. In many-valued logic, it preserves a general designation. But a rule of inference's action is purely syntactic, and does not need to preserve any semantic property: any function from sets of formulae to formulae counts as a rule of inference. Usually only rules that are recursive are important; i.e. rules such that there is an effective procedure for determining whether any given formula is the conclusion of a given set of formulae according to the rule. An example of a rule that is not effective in this sense is the infinitary ω-rule.

Notes

### 6.5.1 The Standard Form of Rules of Inference

#### Rules for Conditionals

*Deduction Theorem (or Conditional Introduction)*

$$\frac{\varphi \vdash \psi}{\varphi \rightarrow \psi}$$

*Modus Ponens (or Conditional Elimination)*

$$\frac{\varphi \rightarrow \psi \quad \varphi}{\psi}$$

*Modus Tollens*

$$\frac{\varphi \rightarrow \psi \quad \neg \psi}{\neg \varphi}$$

#### Rules for Conjunctions

*Adjunction (or Conjunction Introduction)*

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi}$$

*Simplification (or Conjunction Elimination)*

$$\frac{\varphi \wedge \psi}{\varphi} \qquad \frac{\varphi \wedge \psi}{\psi}$$

#### Rules for Disjunctions

*Addition (or Disjunction Introduction)*

$$\frac{\varphi}{\varphi \vee \psi} \qquad \frac{\psi}{\varphi \vee \psi}$$

*Case Analysis*

$$\frac{\varphi \vee \psi \quad \varphi \rightarrow \chi \quad \psi \rightarrow \chi}{\chi}$$

Notes

*Disjunctive Syllogism*

$$\frac{\varphi \vee \psi \quad \neg\varphi}{\psi}$$

$$\frac{\varphi \vee \psi \quad \neg\psi}{\varphi}$$

## Rules for Biconditionals

*Biconditional Introduction*

$$\frac{\varphi \rightarrow \psi \quad \psi \rightarrow \varphi}{\varphi \leftrightarrow \psi}$$

*Biconditional Elimination*

$$\frac{\varphi \leftrightarrow \psi \quad \varphi}{\psi}$$

$$\frac{\varphi \leftrightarrow \psi \quad \psi}{\varphi}$$

$$\frac{\varphi \leftrightarrow \psi \quad \neg\varphi}{\neg\psi}$$

$$\frac{\varphi \leftrightarrow \psi \quad \neg\psi}{\neg\varphi}$$

$$\frac{\varphi \leftrightarrow \psi \quad \psi \vee \varphi}{\psi \wedge \varphi}$$

$$\frac{\varphi \leftrightarrow \psi \quad \neg\psi \vee \neg\varphi}{\neg\psi \wedge \neg\varphi}$$

## Rules of Classical Predicate Calculus

In the following rules, $\varphi(\beta/\alpha)$ is exactly like $\varphi$ except for having the term $\beta$ everywhere $\varphi$ has the free variable $\alpha$.

*Universal Introduction (or Universal Generalization)*

$$\frac{\varphi\,(\beta/\alpha)}{\forall\,\alpha\,\varphi}$$

***Restriction 1:*** $\beta$ does not occur in $\varphi$.

***Restriction 2:*** $\beta$ is not mentioned in any hypothesis or undischarged assumptions.

Notes

***Universal Elimination (or Universal Instantiation)***

$$\frac{\forall \alpha \varphi}{\varphi(\beta/\alpha)}$$

***Restriction:*** No free occurrence of $\alpha$ in $\varphi$ falls within the scope of a quantifier quantifying a variable occurring in $\beta$.

***Existential Introduction (or Existential Generalization)***

$$\frac{\varphi(\beta/\alpha)}{\exists \alpha \varphi}$$

***Restriction:*** No free occurrence of $\alpha$ in $\varphi$ falls within the scope of a quantifier quantifying a variable occurring in $\beta$.

***Existential Elimination (or Existential Instantiation)***

$$\frac{\exists \alpha \varphi \quad \varphi(\beta/\alpha) \vdash \psi}{\psi}$$

***Restriction 1:*** No free occurrence of $\alpha$ in $\varphi$ falls within the scope of a quantifier quantifying a variable occurring in $\beta$.

***Restriction 2:*** There is no occurrence, free or bound, of $\beta$ in $\psi$.

*Example 1:* Let us consider the following assumptions: "If it rains today, then we will not go on a canoe today. If we do not go on a canoe trip today, then we will go on a canoe trip tomorrow. Therefore (Mathematical symbol for "therefore" is $\therefore$), if it rains today, we will go on a canoe trip tomorrow. To make use of the rules of inference in the above table $p$ we let be the proposition "If it rains today", $q$ be " We will not go on a canoe today" and let $r$ be "We will go on a canoe trip tomorrow". Then this argument is of the form:

$$\therefore \quad \frac{p \rightarrow q \quad q \rightarrow r}{p \rightarrow r}$$

*Example 2:* Let us consider a more complex set of assumptions: "It is not sunny today and it is colder than yesterday". "We will go swimming only if it is sunny", "If we do not go swimming, then we will have a barbecue", and "If we will have a barbecue, then we will be home by sunset" lead to the conclusion "We will be home before sunset." Proof by rules of inference: Let $p$ be the proposition "It is sunny this today", $q$ the proposition "It is colder than yesterday", $r$ the proposition "We will go swimming", $s$ the proposition "We will have a barbecue", and $t$ the proposition "We will be home by sunset". Then the hypotheses become $\neg p \wedge q$, $r \rightarrow p$, $\neg r \rightarrow s$ and $s \rightarrow t$. Using our intuition we conjecture that the conclusion might be $t$. Using the Rules of Inference table, we can proof the conjecture easily:

| | Step | Reason |
|---|---|---|
| 1. | $\neg p \wedge q$ | Hypothesis |
| 2. | $\neg p$ | Simplification using Step 1 |
| 3. | $r \rightarrow p$ | Hypothesis |

Notes

| | | |
|---|---|---|
| 4. | $\neg r$ | Modus tollens using Step 2 and 3 |
| 5. | $\neg r \rightarrow s$ | Hypothesis |
| 6. | $s$ | Modus ponens using Step 4 and 5 |
| 7. | $s \rightarrow t$ | Hypothesis |
| 8. | $s$ | Modus ponens using Step 6 and 7 |

### Self Assessment

State whether the following statements are true or false:

9. Horn problems are difficult than non-Horn problems.

10. Transformation rule is the act of drawing a conclusion based on the form of premises interpreted as a function.

## 6.6 Resolution Logic

In mathematical logic and automated theorem proving, resolution is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic. In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable; this method may prove the satisfiability of a first-order satisfiable formula, but not always, as it is the case for all methods for first-order logic. Resolution was introduced by John Alan Robinson in 1965.

The clause produced by a resolution rule is sometimes called a resolvent.

### 6.6.1 Resolution Rule

The resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other (in the following, is taken to be the complement to $b_j$). The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{a_1 \vee \ldots \vee a_i \vee \ldots \vee a_n, b_1 \vee \ldots \vee b_j \vee \ldots \vee b_m}{a_1 \vee \ldots \vee a_{i-1} \vee a_{i+1} \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_{j-1} \vee b_{j+1} \vee \ldots \vee b_m}$$

where

all $a$s and $b$s are literals,

$a_i$ is the complement to $b_j$, and

the dividing line stands for entails

The clause produced by the resolution rule is called the resolvent of the two input clauses.

When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.

Modus ponens can be seen as a special case of resolution of a one-literal clause and a two-literal clause.

Notes

## A Resolution Technique

When coupled with a complete search algorithm, the resolution rule yields a sound and complete algorithm for deciding the satisfiability of a propositional formula, and, by extension, the validity of a sentence under a set of axioms.

This resolution technique uses proof by contradiction and is based on the fact that any sentence in propositional logic can be transformed into an equivalent sentence in conjunctive normal form. The steps are as follows:

- All sentences in the knowledge base and the negation of the sentence to be proved (the conjecture) are conjunctively connected.
- The resulting sentence is transformed into a conjunctive normal form with the conjuncts viewed as elements in a set, $S$, of clauses.

  For example $(A_1 \vee A_2) \wedge (B_1 \vee B_2 \vee B_3) \wedge (C_1)$ gives rise to the set $S = \{A_1 \vee A_2, B_1 \vee B_2 \vee B_3, C_1\}$.
- The resolution rule is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is discarded (as a tautology). If not, and if it is not yet present in the clause set $S$, it is added to $S$, and is considered for further resolution inferences.
- If after applying a resolution rule the empty clause is derived, the original formula is unsatisfiable (or contradictory), and hence it can be concluded that the initial conjecture follows from the axioms.
- If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, the conjecture is not a theorem of the original knowledge base.

One instance of this algorithm is the original Davis – Putnam algorithm that was later refined into the DPLL algorithm that removed the need for explicit representation of the resolvents.

This description of the resolution technique uses a set $S$ as the underlying data-structure to represent resolution derivations. Lists, Trees and Directed Acyclic Graphs are other possible and common alternatives. Tree representations are more faithful to the fact that the resolution rule is binary. Together with a sequent notation for clauses, a tree representation also makes it clear to see how the resolution rule is related to a special case of the cut-rule, restricted to atomic cut-formulas. However, tree representations are not as compact as set or list representations, because they explicitly show redundant subderivations of clauses that are used more than once in the derivation of the empty clause. Graph representations can be as compact in the number of clauses as list representations and they also store structural information regarding which clauses were resolved to derive each resolvent.

*Example:* $\dfrac{a \vee b, \quad \neg a \vee c}{b \vee c}$

***In plain language:*** Suppose $a$ is false. In order for the premise $a \vee b$ to be true, $b$ must be true. Alternatively, suppose $a$ is true. In order for the premise $\neg a \vee c$ to be true, $c$ must be true. Therefore regardless of falsehood or veracity of $a$, if both premises hold, then the conclusion $b \vee c$ is true.

## Resolution in First-order Logic

Notes

In first-order logic, resolution condenses the traditional syllogisms of logical inference down to a single rule. To understand how resolution works, consider the following example syllogism of term logic:

All Greeks are Europeans.

Homer is a Greek.

Therefore, Homer is a European.

Or, more generally:

$$\forall x.P(x) \Rightarrow Q(x)$$

$$P(a)$$

Therefore, $Q(a)$

To recast the reasoning using the resolution technique, first the clauses must be converted to conjunctive normal form. In this form, all quantification becomes implicit: universal quantifiers on variables ($X$, $Y$, …) are simply omitted as understood, while existentially-quantified variables are replaced by Skolem functions.

$$\neg P(x) \vee Q(x)$$

$$P(a)$$

Therefore, $Q(a)$

So the question is, how does the resolution technique derive the last clause from the first two? The rule is simple:

- Find two clauses containing the same predicate, where it is negated in one clause but not in the other.
- Perform a unification on the two predicates. (If the unification fails, you made a bad choice of predicates. Go back to the previous step and try again.)
- If any unbound variables which were bound in the unified predicates also occur in other predicates in the two clauses, replace them with their bound values (terms) there as well.
- Discard the unified predicates, and combine the remaining ones from the two clauses into a new clause, also joined by the "$\vee$" operator.

To apply this rule to the above example, we find the predicate $P$ occurs in negated form

$$\neg P(X)$$

in the first clause, and in non-negated form

$$P(a)$$

in the second clause. $X$ is an unbound variable, while a is a bound value (term). Unifying the two produces the substitution

$$X \mapsto a$$

Discarding the unified predicates, and applying this substitution to the remaining predicates (just $Q(X)$, in this case), produces the conclusion:

$$Q(a)$$

Notes

For another example, consider the syllogistic form:

All Cretans are islanders.

All islanders are liars.

Therefore all Cretans are liars.

Or more generally,

$$\forall X\ P(X) \rightarrow Q(X)$$

$$\forall X\ Q(X) \rightarrow R(X)$$

Therefore, $\forall X\ P(X) \rightarrow R(X)$

In CNF, the antecedents become:

$$\neg P(X) \vee Q(X)$$

$$\neg Q(Y) \vee R(Y)$$

Note that the variable in the second clause was renamed to make it clear that variables in different clauses are distinct.

Now, unifying $Q(X)$ in the first clause with $\neg Q(Y)$ in the second clause means that $X$ and $Y$ become the same variable anyway. Substituting this into the remaining clauses and combining them gives the conclusion:

$$\neg P(X) \vee R(X)$$

The resolution rule, as defined by Robinson, also incorporated factoring, which unifies two literals in the same clause, before or during the application of resolution as defined above. The resulting inference rule is refutation complete, in that a set of clauses is unsatisfiable if and only if there exists a derivation of the empty clause using resolution alone.

### 6.6.2 The Resolution Principle

The resolution principle, due to Robinson (1965), is a method of theorem proving that proceeds by constructing refutation proofs, i.e., proofs by contradiction. This method has been exploited in many automatic theorem provers. The resolution principle applies to first-order logic formulas in Solemnized form. These formulas are basically sets of clauses each of which is a disjunction of literals. Unification is a key technique in proofs by resolution.

If two or more positive literals (or two or more negative literals) in clause are unifiable and $\eta$ is their most general unifier, then $S_\eta$ is called a factor of $s$. For example, $P(x) \vee \neg\, Q(f(x), b) \vee P(g(y))$ is factored to $P(g(y)) \vee \neg\, Q(f(g(y)), b)$. In such a factorization, duplicates are removed. Let $S_1$ and $S_2$ be two clauses with no variables in common, let $S_1$ contain a positive literal $L_1$, $S_2$ contain a negative literal $L_2$, and let $\eta$ be the most general unifier of $L_1$ and $L_2$. Then

$$(S_1\ \eta - L_1\ \eta) \cup (S_2\ \eta - L_2\ \eta)$$

is called a binary resolvent of $S_1$ and $S_2$. For example, if $S_1 = P(x) \vee Q(x)$ and $S_2 = \neg P(a) \vee R(y)$, then $Q(a) \vee R(y)$ is their binary resolvent.

A resolvent of two clauses $S_1$ and $S_2$ is one of the four following binary resolvents.

1. A binary resolvent of $S_1$ and $S_2$.
2. A binary resolvent of $S_1$ and a factor of $S_2$.
3. A binary resolvent of a factor of $S_1$ and $S_2$.
4. A binary resolvent of a factor of $S_1$ and a factor of $S_2$.

Notes

Generation of a resolvent from two clauses, called resolution, is the sole rule of inference of the resolution principle. The resolution principle is complete, so a set (conjunction) of clauses is unclassifiable if the empty clause can be derived from it by resolution. Proof of the completeness of the resolution principle is based on Herbrand's theorem. Since unsatisfiability is dual to validity, the resolution principle is exercised on theorem negations. Multiple strategies have been developed to make the resolution principle more efficient. These strategies help select clause pairs for application of the resolution rule. For instance, linear resolution is the following deduction strategy. Let be the initial set of clauses. If , $C_o \in S$, a linear deduction of $C_n$ from $S$ with top clause $C_o$ is a deduction in which each $C_{i+1}$ is a resolvent of $C_1$ and $B_1 (0 \le i \le n-1)$, and each $B_i$ is either in or a resolvent $C_j$(with $j < i$).

Linear resolution is complete, so if $C$ is a clause in an unsatisfiable set $S$ of clauses and $S - C$ is satisfiable, then the empty clause can be obtained by linear resolution with as the top clause. If the additional restriction that $B_i$ be in $S$ imposed, then this restricted strategy is incomplete. However, it is complete for sets of Horn clauses containing exactly one goal, and the goal is selected as the top clause. All other clauses in such sets are definite Horn clauses. Implementations of programming language Prolog conduct search within the framework of this strategy.

*Caution* Resolution principle can be applied in all situations.

*Task* Apply inference rule to declare today is hot or not.

### Self Assessment

State whether the following statements are true or false:

11. A literal is a propositional variable or the negation of a propositional variable.
12. Unification is a key technique in proofs by resolution.

## 6.7 Deductive Inference Methods

Inference is the act or process of deriving logical conclusions from premises known or assumed to be true. The conclusion drawn is also called an idiomatic. The laws of valid inference are studied in the field of logic. Or inference can be defined in another way. Inference is the non-logical, but rational, means, through observation of patterns of facts, to indirectly see new meanings and contexts for understanding. Of particular use to this application of inference are anomalies and symbols. Inference, in this sense, does not draw conclusions but opens new paths for inquiry. In this definition of inference, there are two types of inference: inductive inference and deductive inference. Unlike the definition of inference in the first paragraph above, meaning of word meanings are not tested but meaningful relationships are articulated. Human inference (i.e. how humans draw conclusions) is traditionally studied within the field of cognitive psychology; artificial intelligence researchers develop automated inference systems to emulate human inference. Statistical inference allows for inference from quantitative data.

### 6.7.1 Law of Detachment

In propositional logic is a valid, simple argument form and rule of inference. It can be summarized as "*P* implies *Q*; *P* is asserted to be true, so therefore *Q* must be true." The history of modus

Notes

ponens goes back to antiquity. While modus ponens is one of the most commonly used concepts in logic it must not be mistaken for a logical law; rather, it is one of the accepted mechanisms for the construction of deductive proofs that includes the "rule of definition" and the "rule of substitution". Modus ponens allows one to eliminate a conditional statement from a logical proof or argument (the antecedents) and thereby not carry these antecedents forward in an ever-lengthening string of symbols; for this reason modus ponens is sometimes called the rule of detachment. For example, modus ponens can produce shorter formulas from longer ones and Russell observes that "the process of the inference cannot be reduced to symbols. Its sole record is the occurrence of $\vdash q$ [the consequent] . . . an inference is the dropping of a true premise; it is the dissolution of an implication".

A justification for the "trust in inference is the belief that if the two former assertions [the antecedents] are not in error, the final assertion [the consequent] is not in error". In other words, if one statement or proposition implies a second one, and the first statement or proposition is true, then the second one is also true. If $P$ implies $Q$ and $P$ is true, then $Q$ is true. An example is:

If it's raining, I'll meet you at the movie theater.

It's raining.

Therefore, I'll meet you at the movie theater.

*Modus ponens* can be stated formally as: $\dfrac{P \to Q, P}{\therefore Q}$

where the rule is that whenever an instance of "$P \to Q$" and "$P$" appear by themselves on lines of a logical proof, $Q$ can validly be placed on a subsequent line; furthermore, the premise $P$ and the implication "dissolves", their only trace being the symbol $Q$ that is retained for use later e.g. in a more complex deduction.

### 6.7.2 Law of Syllogism

A relation of inference is monotonic if the addition of premises does not undermine previously reached conclusions; otherwise the relation is no monotonic. Deductive inference, is monotonic: if a conclusion is reached on the basis of a certain set of premises, then that conclusion still holds if more premises are added. By contrast, everyday reasoning is mostly non-monotonic because it involves risk: we jump to conclusions from deductively insufficient premises. We know when it is worth or even necessary (e.g. in medical diagnosis) to take the risk. Yet we are also aware that such inference is defensible—that new information may undermine old conclusions. Various kinds of defensible but remarkably successful inference have traditionally captured the attention of philosophers (theories of induction, Peirce's theory of abduction, inference to the best explanation, etc.). More recently logicians have begun to approach the phenomenon from a formal point of view. The result is a large body of theories at the interface of philosophy, logic and artificial intelligence.

### 6.7.3 Deductive Logic: Validity and Soundness

In mathematical logic, a logical system has the soundness property if and only if its inference rules prove only formulas that are valid with respect to its semantics. In most cases, this comes down to its rules having the property of preserving *truth*, but this is not the case in general. Soundness is among the most fundamental properties of mathematical logic. A soundness property provides the initial reason for counting a logical system as desirable. The completeness property means that every validity (truth) is provable. Together they imply that all and only validities are provable. Most proofs of soundness are trivial.

Notes

*Example:* In an axiomatic system proof of soundness amounts to verifying the validity of the axioms and that the rules of inference preserve validity (or the weaker property, truth). Most axiomatic systems have only the rule of modus ponens (and sometimes substitution), so it requires only verifying the validity of the axioms and one rule of inference.

Soundness properties come in two main varieties: weak and strong soundness, of which the former is a restricted form of the latter.

### Soundness

Soundness of a deductive system is the property that any sentence that is provable in that deductive system is also true on all interpretations or structures of the semantic theory for the language upon which that theory is based. In symbols, where $S$ is the deductive system, $L$ the language together with its semantic theory, and $P$ a sentence of $L$: if $\vdash_S P$, then also $\models_L P$.

### Strong Soundness

Strong soundness of a deductive system is the property that any sentence $P$ of the language upon which the deductive system is based that is derivable from a set $\Gamma$ of sentences of that language is also a logical consequence of that set, in the sense that any model that makes all members of $\Gamma$ true will also make $P$ true. In symbols, where $\Gamma$ is a set of sentences of $L$: if $\Gamma \vdash_S P$, then also $\Gamma \models_L P$. Notice that in the statement of strong soundness, when $\Gamma$ is empty, we have the statement of weak soundness.

### Arithmetic Soundness

If $T$ is a theory whose objects of discourse can be interpreted as natural numbers, we say $T$ is arithmetically sound if all theorems of $T$ are actually true about the standard mathematical integers.

## 6.7.4 Representations Using Rules Dealing with Inconsistencies and Uncertainties

AI research is highly technical and specialized, deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. There are subfields which are focused on the solution of specific problems, on one of several possible approaches, on the use of widely differing tools and towards the accomplishment of particular applications. The central problems (or goals) of AI research include reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects. General intelligence (or "strong AI") is still among the field's long term goals. Currently popular approaches include statistical methods, computational intelligence and traditional symbolic AI. There are an enormous number of tools used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics, and many others. The field was founded on the claim that a central property of humans, intelligence—the sapience of *Homo sapiens*—can be so precisely described that it can be simulated by a machine. This raises philosophical issues about the nature of the mind and the ethics of creating artificial beings, issues which have been addressed by myth, fiction and philosophy since antiquity. Artificial intelligence has been the subject of tremendous optimism but has also suffered stunning setbacks. Today it has become an essential part of the technology industry, providing the heavy lifting for many of the most difficult problems in computer science.

Notes

### Self Assessment

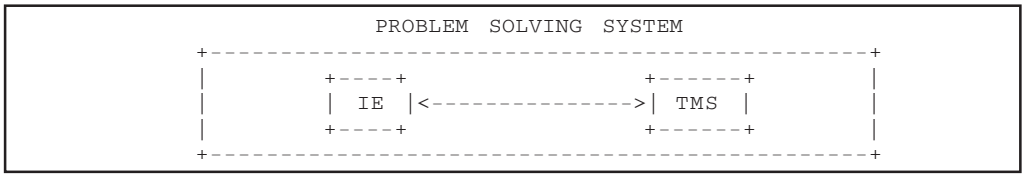
State whether the following statements are true or false:

13. Deductive inference is monotonic.

14. AI research is not technical and specialized, deeply divided into subfields that often fail to communicate with each other.

## 6.8 Truth Maintenance System (TMS)

Truth Maintenance Systems, also called Reason Maintenance Systems, are used within Problem Solving Systems, in conjunction with Inference Engines (IE) such as rule-based inference systems, to manage as Dependency Network the inference engine's beliefs in given sentences.




A TMS is intended to satisfy a number of goals:

#### Provide Justifications for Conclusions

When a problem solving system gives an answer to a user's query, an explanation of the answer is usually required. If the advice to a stockbroker is to invest millions of dollars, an explanation of the reasons for that advice can help the broker reach a reasonable decision.

*Notes* An explanation can be constructed by the IE by tracing the justification of the assertion.

#### Recognise Inconsistencies

The IE may tell the TMS that some sentences are contradictory. Then, if on the basis of other IE commands and of inferences we find that all those sentences are believed true, then the TMS reports to the IE that a contradiction has arisen. The IE can eliminate an inconsistency by determining the assumptions used and changing them appropriately, or by presenting the contradictory set of sentences to the users and asking them to choose which sentence(s) to retract.

#### Support Default Reasoning

In many situations we want, in the absence of firmer knowledge, to reason from default assumptions. If Tweety is a bird, until told otherwise, we will assume that Tweety flies and use as justification the fact that Tweety is a bird and the assumption that birds fly.

#### Remember Derivations Computed Previously

In the process of determining what is responsible for a network problem, we may have derived, while examining the performance of a name server, that Ms. Doe is an expert on e-mail systems.

Notes

That conclusion will not need to be derived again when the name server ceases to be the potential culprit for the problem and we examine instead the routers.

### Support Dependency Driven Backtracking

The justification of a sentence, as maintained by the TMS, provides the natural indication of what assumptions need to be changed if we want to invalidate that sentence.

Our belief [by "our belief" we mean the "inference-engine's belief"] about a sentence can be:

- false, the sentence is believed to be unconditionally false; this is also called a contradiction
- true, the sentence is believed unconditionally true; this is also called a premise
- assumed-true, the sentence is assumed true [we may change our belief later]; this is also called an enabled assumption
- assumed-false, the sentence is assumed false [we may change our belief later]; this is also called a retracted assumption
- assumed, the sentence is believed by inference from other sentences
- don't-care.

We say if a sentence is true or assumed to be true; it is out of false, assumed-false, or don't-care. A TMS maintains a Dependency Network. The network is bipartite, with nodes representing sentences and justifications. A sentence may correspond to a fact, such as "Socrates is a man", or a rule, such as "if ?x is a man then ?x is mortal". We will say that a sentence node is a premise if its sentence is true, is a contradiction if its sentence is false, is an assumption if its sentence is assumed-true or assumed-false or assumed. A sentence node receives arcs from justification nodes. Each such justification node provides an argument for believing the given sentence node. In turn, a sentence node has arcs to the justification nodes that use it as a premise. A justification node has inputs from sentence nodes, its premises or justifiers, and has output to a sentence node, its conclusion or justification. A justification node represents the inference of the conclusion from the conjunction of the stated premises. The conventions used in representing graphically dependency networks are summarized here.

The TMS maintains the following information with each sentence node:

- a sentence
- a label expressing the current belief in the sentence; it is IN for sentences that are believed, and OUT for sentences that are not believed.
- a list of the justification nodes that support it
- a list of the justification nodes supported by it
- an indication if the node is an assumption, contradiction, or premise.

The TMS maintains very little information with justification nodes. Only a label with value IN or OUT depending if we believe the justification valid or not.

The IE can give at least the following orders to the TMS:

- create a sentence node with specific properties
- create a justification with specific premises and conclusions
- attach rules to sentence nodes and execute them when specific beliefs hold at those nodes [these are like callbacks or triggers; a standard callback informs the IE when a contradiction becomes true]
- retrieve the properties of a sentence node or its justifications and consequences.

**Notes**

Truth Maintenance Systems can have different characteristics:

### Justification-based Truth Maintenance System (JTMS)

It is a simple TMS where one can examine the consequences of the current set of assumptions. The meaning of sentences is not known.

### Assumption-based Truth Maintenance System (ATMS)

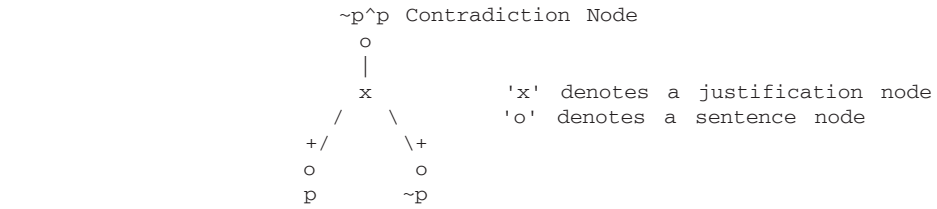
It allows to maintain and reason with a number of simultaneous, possibly incompatible, current sets of assumption. Otherwise it is similar to JTMS, i.e. it does not recognise the meaning of sentences.

### Logical-based Truth Maintenance System (LTMS)

Like JTMS in that it reasons with only one set of current assumptions at a time. More powerful than JTMS in that it recognises the propositional semantics of sentences, i.e. understands the relations between p and ~p, p and q and p&q, and so on.

A Justification-based Truth Maintenance System is a simple TMS where one can examine the consequences of the current set of assumptions. In JTMS, labels are attached to arcs from sentence nodes to justification nodes. This label is either "+" or "-". Then, for a justification node we can talk of its in-list, the list of its inputs with "+" label, and of its out-list, the list of its inputs with "-" label.

The meaning of sentences is not known. We can have a node representing a sentence p and one representing ~p and the two will be totally unrelated, unless relations are established between them by justifications. For example, we can write:




which says that if both p and ~p are IN we have a contradiction.

The association of IN or OUT labels with the nodes in a dependency network defines an in-out-labeling function. This function is consistent if:

- The label of a justification node is IN if the labels of all the sentence nodes in its in-list are all IN and the labels of all the sentence nodes in its out-list are OUT.
- The label of a sentence node is IN if it is a premise, or an enabled assumption node, or it has an input from a justification node with label IN.

Here are examples of JTMS operations:

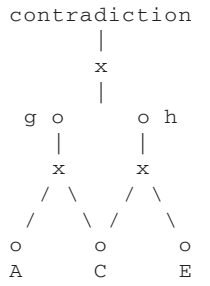
- ***create-jtms (node-string contradiction-handler enqueue-procedure)*** creates a dependency network. The parameters are:
  - node-string, a function to be used by the JTMS, which, given a node, returns its description
  - contradiction-handler, a function to be invoked by the JTMS when it recognises a contradiction
  - enqueue-procedure, a function to be called by the JTMS when it changes the label of a node to IN.

Notes

- ***tms-create-node (jtms datum assumptionp contradictoryp)*** creates a sentence node. The parameters are:
  - jtms, the jtms within which the sentence node is to be created
  - datum, the sentence
  - assumptionp, true if this is an assumption node; initially the assumption is retracted
  - contradictoryp, true if this is a contradictory node

  Note that premise nodes are recognised because they are given a justification without any premises.
- ***enable-assumption (node)***, retract-assumption (node). These functions respectively enable and retract an existing assumption node.
- ***justify-node (informant conclusion-node list-of-premise-nodes)***, creates a justification node. The parameters are:
  - The informant, that is who/what is entering this justification
  - The conclusion-node and list-of-premise-nodes that specify respectively the conclusion and the premises of this justification.

Here is a simple dependency network:




```
This network is created:
     (setq *jtms* (create-jtms "Forbus-deKleer example on page 181"))
The nodes A, C, E are created and enabled:
     (setq assumption-a (tms-create-node *jtms* "A" :assumptionp t)
           assumption-c (tms-create-node *jtms* "C" :assumptionp t)
           assumption-e (tms-create-node *jtms* "E" :assumptionp t))
     (enable-assumption assumption-a)
     (enable-assumption assumption-c)
     (enable-assumption assumption-e)
The node h is created and justified:
     (setq node-h (tms-create-node *jtms* "h"))
     (justify-node "R1" node-h (list assumption-c assumption-a))
                ;; R1 is the informant for this
                ;; justification
The node g is created, justified, and established as a contradiction:
     (setq node-g (tms-create-node *jtms* "g"))
     (justify-node "R2" node-g (list assumption-a assumption-c))
     (setq contradiction (tms-create-node *jtms* 'contra
:contradictoryp t))
     (justify-node "R3" contradiction (list node-g))

This last command will recognise that a contradiction exists and will
invoke a contradiction handler that should state that the node
"contradiction" is a contradiction and that some of the assumptions
in {A, C} should be retracted.
```

Here should be a larger dependency network as used in a JTMS.

Notes

### Self Assessment

State whether the following statements are true or false:

15. The IE can eliminate an inconsistency by determining the assumptions used and changing them.

16. A TMS maintains a Dependency Network.

## 6.9 Predicated Completion and Circumscription

Circumscription is a non-monotonic logic created by John McCarthy to formalize the common sense assumption that things are as expected unless otherwise specified. Circumscription was later used by McCarthy in an attempt to solve the frame problem. In its original first-order logic formulation, circumscription minimizes the extension of some predicates, where the extension of a predicate is the set of tuples of values the predicate is true on. This minimization is similar to the closed world assumption that what is not known to be true is false.

The original problem considered by McCarthy was that of missionaries and cannibals: there are three missionaries and three cannibals on one bank of a river; they have to cross the river using a boat that can only take two, with the additional constraint that cannibals must never outnumber the missionaries on either bank (as otherwise the missionaries would be killed and, presumably, eaten). The problem considered by McCarthy was not that of finding a sequence of steps to reach the goal (the article on them and problem contains one such solution), but rather that of excluding conditions that are not explicitly stated. For example, the solution "go half a mile south and cross the river on the bridge" is intuitively not valid because the statement of the problem does not mention such a bridge. On the other hand, the existence of this bridge is not excluded by the statement of the problem either. That the bridge does not exist is a consequence of the implicit assumption that the statement of the problem contains everything that is relevant to its solution. Explicitly stating that a bridge does not exist is not a solution to this problem, as there are many other exceptional conditions that should be excluded (such as the presence of a rope for fastening the cannibals, the presence of a larger boat nearby, etc.) Circumscription was later used by McCarthy to formalize the implicit assumption of inertia: things do not change unless otherwise specified. Circumscription seemed to be useful to avoid specifying that conditions are not changed by all actions except those explicitly known to change them; this is known as the frame problem. However, the solution proposed by McCarthy was later shown leading to wrong results in some cases, like in the Yale shooting problem scenario. Other solutions to the frame problem that correctly formalize the Yale shooting problem exist; some use circumscription but in a different way.

### 6.9.1 The Propositional Case

Circumscription is a non-monotonic logic created by John McCarthy to formalize the common sense assumption that things are as expected unless otherwise specified. Circumscription was later used by McCarthy in an attempt to solve the frame problem. In its original first-order logic formulation, circumscription minimizes the extension of some predicates, where the extension of a predicate is the set of tuples of values the predicate is true on. This minimization is similar to the closed world assumption that what is not known to be true is false.

The original problem considered by McCarthy was that of missionaries and cannibals: there are three missionaries and three cannibals on one bank of a river; they have to cross the river using a boat that can only take two, with the additional constraint that cannibals must never outnumber the missionaries on either bank (as otherwise the missionaries would be killed and, presumably, eaten). The problem considered by McCarthy was not that of finding a sequence of steps to reach

Notes

the goal (the article on the missionaries and cannibals problem contains one such solution), but rather that of excluding conditions that are not explicitly stated. For example, the solution "go half a mile south and cross the river on the bridge" is intuitively not valid because the statement of the problem does not mention such a bridge. On the other hand, the existence of this bridge is not excluded by the statement of the problem either. That the bridge does not exist is a consequence of the implicit assumption that the statement of the problem contains everything that is relevant to its solution. Explicitly stating that a bridge does not exist is not a solution to this problem, as there are many other exceptional conditions that should be excluded (such as the presence of a rope for fastening the cannibals, the presence of a larger boat nearby, etc.)

While circumscription was initially defined in the first-order logic case, the particularization to the propositional case is easier to define. Given a propositional formula $T$, its circumscription is the formula having only the models of $T$ that do not assign a variable to true unless necessary. Formally, propositional models can be represented by sets of propositional variables; namely, each model is represented by the set of propositional variables it assigns to true. For example, the model assigning true to $a$, false to $b$, and true to $c$ is represented by the set $\{a, c\}$, because $a$ and $c$ are exactly the variables that are assigned to true by this model. Given two models $M$ and $N$ represented this way, the condition $N \subseteq M$ is equivalent to $M$ setting to true every variable that $N$ sets to true. In other words, $\subseteq$ models the relation of "setting to true less variables". $N \subset M$ means that $N \subseteq M$ but these two models do not coincide.

This lets us define models that do not assign variables to true unless necessary. A model $M$ of a theory $T$ is called minimal, if and only if there is no model $N$ of $T$ for which $N \subset M$.

Circumscription is expressed by selecting only the minimal models. It is defined as follows:

$$CIRC(T) = \{M \mid M \text{ is a minimal model of } T\}$$

Alternatively, one can define $CIRC(T)$ as a formula having exactly the above set of models; furthermore, one can also avoid giving a definition of $CIRC$ and only define minimal inference as $T \models_M Q$ if and only if every minimal model of $T$ is also a model of $Q$.

As an example, the formula $T = a \wedge (b \vee c)$ has three models:

1. $a$, $b$, are $c$ true, i.e. $\{a, b, c\}$;
2. $a$ and $b$ are true, $c$ is false, i.e. $\{a, b\}$;
3. $a$ and $c$ are true, $b$ is false, i.e. $\{a, c\}$.

The first model is not minimal in the set of variables it assigns to true. Indeed, the second model makes the same assignments except for $c$, which is assigned to false and not to true. Therefore, the first model is not minimal. The second and third models are incomparable: while the second assigns true to $b$, the third assigns true to $c$ instead. Therefore, the models circumscribing $T$ are the second and third models of the list. A propositional formula having exactly these two models is the following one:

$$a \wedge \neg (b \leftrightarrow c)$$

Intuitively, in circumscription a variable is assigned to true only if this is necessary. Dually, if a variable can be false, it must be false. For example, at least one of $b$ and $c$ must be assigned to true according to $T$; in the circumscription exactly one of the two variables must be true. The variable cannot be false in any model of $T$ and neither of the circumscription.

### 6.9.2 Fixed and Varying Predicates

The extension of circumscription with fixed and varying predicates is due to Vladimir Lifschitz. The idea is that some conditions are not to be minimized. In propositional logic terms, some variables are not to be falsified if possible. In particular, two kind of variables can be considered:

Notes

***Varying:*** These are variables that are not to be taken into account at all in the course of minimization.

***Fixed:*** These are variables considered fixed while doing a minimization; in other words, minimization can be done only by comparing models with the same values of these variables.

The difference is the value of the varying conditions is simply assumed not to matter. The fixed conditions instead characterize a possible situation, so that comparing two situations where these conditions have different value makes no sense.

Formally, the extension of circumscription that incorporate varying and fixed variables is as follows, where $P$ is the set of variables to minimize, $Z$ the fixed variables, and the varying variables are those not in $P \bigcup Z$:

$$CIRC(T; P, Z) = \{M \mid M \models T \text{ and } \exists N \text{ such that } N \models T, N \bigcap P \subset M \bigcap P \text{ and } N \bigcap Z = M \bigcap Z\}$$

In words, minimization of the variables assigned to true is only done for the variables in $P$; moreover, models are only compared if the assign the same values on the variables of $Z$. All other variables are not taken into account while comparing models. The solution to the frame problem proposed by McCarthy is based on circumscription with no fixed conditions. In the propositional case, this solution can be described as follows: in addition to the formulae directly encoding what is known, one also define new variables representing changes in the values of the conditions; these new variables are then minimized.

For example, of the domain in which there is a door that is closed at time 0 and in which the action of opening the door is executed at time 2, what is explicitly known is represented by the two formulae:

$$\neg \text{open}_0$$
$$\text{true} \rightarrow \text{open}_2$$

The frame problem shows in this example as the problem that $\neg \text{open}_1$ is not a consequence of the above formulae, while the door is supposed to stay closed until the action of opening it is performed. Circumscription can be used to this aim by defining new variables change_open$_t$ to model changes and then minimizing them:

$$\text{change open}_0 \equiv (\text{open}_0 \not\equiv \text{open}_1)$$
$$\text{change open}_1 \equiv (\text{open}_1 \not\equiv \text{open}_2)$$
$$\ldots$$

As shown by the Yale shooting problem, this kind of solution does not work. For example, $\neg \text{open}_1$ is not yet entailed by the circumscription of the formulae above: the model in which change open$_0$ is true and change open$_1$ is false is incomparable with the model with the opposite values. Therefore, the situation in which the door becomes open at time 1 and then remains open as a consequence of the action is not excluded by circumscription.

Several other formalizations of dynamical domains not suffering from such problems have been developed. Many use circumscription but in a different way.

### 6.9.3 Predicate Circumscription

The original definition of circumscription proposed by McCarthy is about first-order logic. The role of variables in propositional logic (something that can be true or false) is played in first-order logic by predicates. Namely, a propositional formula can be expressed in first-order logic by replacing each propositional variable with a predicate of zero arity (i.e., a predicate with no

Notes

arguments). Therefore, minimization is done on predicates in the first-order logic version of circumscription: the circumscription of a formula is obtained forcing predicates to be false whenever possible.

Given a first-order logic formula $T$ containing a predicate $P$, circumscribing this predicate amounts to selecting only the models of $T$ in which $P$ is assigned to true on a minimal set of tuples of values.

Formally, the extension of a predicate in a first-order model is the set of tuples of values this predicate assign to true in the model. First-order models indeed includes the evaluation of each predicate symbol; such an evaluation tells whether the predicate is true or false for any possible value of its arguments. Since, each argument of a predicate must be a term, and each term evaluates to a value, the models tells whether $P(v_1,\ldots v_n)$ is true for any possible tuple of values $\langle v_1,\ldots v_n\rangle$. The extension of $P$ in a model is the set of tuples of terms such that $P(v_1,\ldots v_n)$ is true in the model.

The circumscription of a predicate $P$ in a formula $T$ is obtained by selecting only the models of $T$ with a minimal extension of $P$. For example, if a formula has only two models, differing only because $P(v_1,\ldots v_n)$ is true in one and false in the second, then only the second model is selected. This is because $\langle v_1,\ldots v_n\rangle$ is in the extension of $P$ in the first model but not in the second.

The original definition by McCarthy was syntactical rather than semantical. Given a formula $T$ and a predicate $P$, circumscribing $P$ in $T$ is the following second-order formula:

$$T(P) \wedge \forall p \neg (T(P) \wedge p < P)$$

In this formula, $p$ is a predicate of the same arity as $P$. This is a second-order formula because it contains a quantification over a predicate. The subformula $p < P$ is a shorthand for:

$$\forall x(p(x) \rightarrow P(x)) \wedge \neg \forall x(P(x) \rightarrow p(x))$$

In this formula, $x$ is a n-tuple of terms, where n is the arity of $P$. This formula states that extension minimization has to be done: in order for a truth evaluation on $P$ of a model being considered, it must be the case that no other predicate $p$ can assign to false every tuple that $P$ assigns to false and yet being different from $P$.

This definition only allows circumscribing a single predicate. While the extension to more than one predicate is trivial, minimizing the extension of a single predicate has an important application: capturing the idea that things are usually as expected. This idea can be formalized by minimized a single predicate expressing the abnormality of situations. In particular, every known fact is expressed in logic with the addition of a literal $\neg Abnormal(\ldots)$ stating that the fact holds only in normal situations. Minimizing the extension of this predicate allows for reasoning under the implicit assumption that things are as expected (that is, they are not abnormal), and that this assumption is made only if possible (abnormality can be assumed false only if this is consistent with the facts.)

### 6.9.4 Pointwise Circumscription

Pointwise circumscription is a variant of first-order circumscription that has been introduced by Vladimir Lifschitz. In the propositional case, pointwise and predicate circumscription coincide. The rationale of pointwise circumscription it minimize the value of a predicate for each tuple of values separately, rather than minimizing the extension of the predicate. For example, there are two models of $P(a) \equiv P(b)$ with domain $\{a, b\}$, one setting $P(a) = P(b) = false$ and the other setting $P(a) = P(b) = true$. Since, the extension of $P$ in the first model is 0 while the extension for the second is $\{a, b\}$, circumscription only selects the first model.

Notes

In pointwise circumscription, each tuple of values is considered separately. For example, in the formula $P(a) \equiv P(b)$ one would consider the value of $P(a)$ separately from $P(b)$. A model is minimal only it is not possible to turn any such value from true to false while still satisfying the formula. As a result, the model in which $P(a) = P(b) = true$ is selected by pointwise circumscription because turning only $P(a)$ into false does not satisfy the formula, and the same happens for $P(b)$.

### 6.9.5 Domain and Formula Circumscription

An earlier formulation of circumscription by McCarthy is based on minimizing the domain of first-order models, rather than the extension of predicates. Namely, a model is considered less than another if it has a smaller domain, and the two models coincide on the evaluation of the common tuples of values. This version of circumscription can be reduced to predicate circumscription. Formula circumscription was a later formalism introduced by McCarthy. This is a generalization of circumscription in which the extension of a formula is minimized, rather than the extension of a predicate. In other words, a formula can be specified so that the set of tuples of values of the domain that satisfy the formula is made as small as possible.

### Self Assessment

State whether the following statements are true or false:

17. Circumscription was later used by Vladimir Lifschitz in an attempt to solve the frame problem.
18. Pointwise circumscription is a variant of first-order circumscription that has been introduced by McCarthy.

## 6.10 Modal Logic

**Modal logic** is a type of formal logic primarily developed in the 1960s that extends classical propositional and predicate logic to include operators expressing modality. Modals—words that express modalities—qualify a statement.

*Example:* The statement "John is happy" might be qualified by saying that John is usually happy, in which case the term "usually" is functioning as a modal.

The traditional alethic modalities, or modalities of truth, include possibility ("Possibly, p", "It is possible that p"), necessity ("Necessarily, p", "It is necessary that p"), and impossibility ("Impossibly, p", "It is impossible that p") Other modalities that have been formalized in modal logic include temporal modalities, or modalities of time (notably, "It was the case that p", "It has always been that p", "It will be that p", "It will always be that p"), deontic modalities (notably, "It is obligatory that p", and "It is permissible that p"), epistemic modalities, or modalities of knowledge ("It is known that p") and doxastic modalities, or modalities of belief ("It is believed that p"). A formal modal logic represents modalities using modal operators.

*Example:* "It might rain today" and "It is possible that rain will fall today" both contain the notion of possibility.

In a modal logic, this is represented as an operator, possibly, attached to the sentence "It will rain today". The basic unary (1-place) modal operators are usually written $\Box$ for necessarily and $\Diamond$ for possibly. In a classical modal logic, each can be expressed by the other with negation:

Notes

$\Diamond P \leftrightarrow \neg\Box\neg P$;
$\Box P \leftrightarrow \neg\Diamond\neg P$.

Thus it is possible that it will rain today if and only if it is not necessary that it will not rain today; and it is necessary that it will rain today if and only if it is not possible that it will not rain today. Alternative symbols used for the modal operators are "L" for necessarily and "M" for Possibly.

### Self Assessment

State whether the following statements are true or false:

19. A informal modal logic represents modalities using modal operators.

20. Modal logic is a type of formal logic primarily developed in the 1980s.

## 6.11 Temporal Logic

In logic, the term "temporal logic" is used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. In a temporal logic, we can then express statements like "I am always hungry", "I will eventually be hungry", or "I will be hungry until I eat something". Temporal logic is sometimes also used to refer to tense logic, a particular modal logic-based system of temporal logic introduced by Arthur Prior in the late 1950s, and important results obtained were by Hans Kamp. Subsequently it has been developed further by computer scientists, notably Amir Pnueli, and logicians. Temporal logic has found an important application in formal verification, where it is used to state requirements of hardware or software systems. For instance, one may wish to say that whenever a request is made, access to a resource is eventually granted, but it is never granted to two requestors simultaneously. Such a statement can conveniently be expressed in a temporal logic.

Consider the statement: "I am hungry." Though its meaning is constant in time, the truth value of the statement can vary in time. Sometimes the statement is true, and sometimes the statement is false, but the statement is never true and false simultaneously. In a temporal logic, statements can have a truth value which can vary in time. Contrast this with a temporal logic, which can only discuss statements whose truth value is constant in time. This treatment of truth values over time differentiates temporal logic from computational verb logic. Temporal logic always has the ability to reason about a time line. So-called linear time logics are restricted to this type of reasoning. Branching logics, however, can reason about multiple time lines. This presupposes an environment that may act unpredictably. To continue the example, in a branching logic we may state that "there is a possibility that I will stay hungry forever." We may also state that "there is a possibility that eventually I am no longer hungry." If we do not know whether or not I will ever get fed, these statements are both true some times.

### Self Assessment

State whether the following statements are true or false:

21. Temporal logic has found an important application in formal verification.

22. Temporal logic is used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time.

Notes

## 6.12 Fuzzy Logic

Fuzzy logic is a form of many-valued logic or probabilistic logic; it deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets (where variables may take on true or false values) fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions. Irrationality can be described in terms of what is known as the fuzzjective. The term "fuzzy logic" was introduced with the 1965 proposal of fuzzy set theory by Lotfi A. Zadeh. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence. Fuzzy logics however had been studied since the 1920s as infinite-valued logics notably by Lukasiewicz and Tarski.

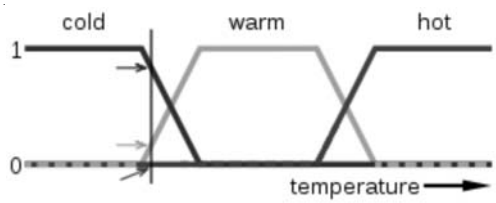
Classical logic only permits propositions having a value of truth or falsity. The notion of whether 1+1=2 is absolute, immutable, mathematical truth. However, there exist certain propositions with variable answers, such as asking various people to identify a color. The notion of truth doesn't fall by the wayside, but rather a means of representing and reasoning over partial knowledge is afforded, by aggregating all possible outcomes into a dimensional spectrum. Both degrees of truth and probabilities range between 0 and 1 and hence may seem similar at first. For example, let a 100 ml glass contain 30 ml of water. Then we may consider two concepts: Empty and Full. The meaning of each of them can be represented by a certain fuzzy set. Then one might define the glass as being 0.7 empty and 0.3 full. Note that the concept of emptiness would be subjective and thus would depend on the observer or designer. Another designer might equally well design a set membership function where the glass would be considered full for all values down to 50 ml. It is essential to realize that fuzzy logic uses truth degrees as a mathematical model of the vagueness phenomenon while probability is a mathematical model of ignorance.

### 6.12.1 Applying Truth Values

A basic application might characterize subranges of a continuous variable.

*Example:* A temperature measurement for anti-lock brakes might have several separate membership functions defining particular temperature ranges needed to control the brakes properly. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the brakes should be controlled.

**Figure 6.1: Fuzzy Logic Temperature**




In this image, the meanings of the expressions cold, warm, and hot are represented by functions mapping a temperature scale. A point on that scale has three "truth values"—one for each of the three functions. The vertical line in the image represents a particular temperature that the three arrows (truth values) gauge. Since the red arrow points to zero, this temperature may be interpreted as "not hot". The orange arrow (pointing at 0.2) may describe it as "slightly warm" and the blue arrow (pointing at 0.8) "fairly cold".

Notes

### Linguistic Variables

While variables in mathematics usually take numerical values, in fuzzy logic applications, the non-numeric linguistic variables are often used to facilitate the expression of rules and facts. A linguistic variable such as age may have a value such as young or its antonym old. However, the great utility of linguistic variables is that they can be modified via linguistic hedges applied to primary terms. The linguistic hedges can be associated with certain functions.

### Early Applications

The Japanese were the first to utilize fuzzy logic for practical applications. The first notable application was on the high-speed train in Sendai, in which fuzzy logic was able to improve the economy, comfort, and precision of the ride. It has also been used in recognition of hand written symbols in Sony pocket computers, Canon auto-focus technology, Omron auto-aiming cameras, earthquake prediction and modeling at the Institute of Seismology Bureau of Metrology in Japan, etc.

*Example:* Fuzzy set theory defines fuzzy operators on fuzzy sets. The problem in applying this is that the appropriate fuzzy operator may not be known. For this reason, fuzzy logic usually uses IF-THEN rules, or constructs that are equivalent, such as fuzzy associative matrices. Rules are usually expressed in the form:

**IF** ***variable*** **IS** ***property*** **THEN** ***action***

For example, a simple temperature regulator that uses a fan might look like this:

```
IF temperature IS very cold THEN stop fan
IF temperature IS cold THEN turn down fan
IF temperature IS normal THEN maintain level
IF temperature IS hot THEN speed up fan
```

There is no "ELSE" – all of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to different degrees.

The AND, OR, and NOT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the Zadeh operators. So for the fuzzy variables x and y:

```
NOT x = (1 - truth(x))
x AND y = minimum(truth(x), truth(y))
x OR y = maximum(truth(x), truth(y))
```

## Self Assessment

State whether the following statements are true or false:

23. Fuzzy logic is a form of many-valued logic or probabilistic logic.

24. Fuzzy logics however had been studied since the 1940s as infinite-valued logics notably by Lukasiewicz and Tarski.

## 6.13 Natural Language Computation

Natural language processing (NLP) is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human – computer interaction. Many challenges in NLP involve natural language understanding – that is, enabling computers to derive meaning from

Notes

human or natural language input. What distinguishes these computation from other potential and actual NLP tasks is not only the volume of research devoted to them but the fact that for each one there is typically a well-defined problem setting, a standard metric for evaluating the task, standard corpora on which the task can be evaluated, and competitions devoted to the specific task.

### 6.13.1 Defining Natural Language

The following is a list of some of the most commonly researched tasks in NLP. Note that some of these tasks have direct real-world applications, while others more commonly serve as subtasks that are used to aid in solving larger tasks.

- ***Automatic Summarization:*** Produce a readable summary of a chunk of text. Often used to provide summaries of text of a known type, such as articles in the financial section of a newspaper.
- ***Co-reference Resolution:*** Given a sentence or larger chunk of text, determine which words ("mentions") refer to the same objects ("entities"). Anaphora resolution is a specific example of this task, and is specifically concerned with matching up pronouns with the nouns or names that they refer to. The more general task of co reference resolution also includes identifying so-called "bridging relationships" involving referring expressions. For example, in a sentence such as "He entered John's house through the front door", "the front door" is a referring expression and the bridging relationship to be identified is the fact that the door being referred to is the front door of John's house (rather than of some other structure that might also be referred to).
- ***Discourse Analysis:*** This rubric includes a number of related tasks. One task is identifying the discourse structure of connected text, i.e. the nature of the discourse relationships between sentences (e.g. elaboration, explanation, contrast). Another possible task is recognizing and classifying the speech acts in a chunk of text (e.g. yes-no question, content question, statement, assertion, etc.).
- ***Machine Translation:*** Automatically translate text from one human language to another. This is one of the most difficult problems, and is a member of a class of problems colloquially termed "AI-complete", i.e. requiring all of the different types of knowledge that humans possess (grammar, semantics, facts about the real world, etc.) in order to solve properly.
- ***Morphological Segmentation:*** Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology (i.e. the structure of words) of the language being considered. English has fairly simple morphology, especially inflectional morphology, and thus it is often possible to ignore this task entirely and simply model all possible forms of a word (e.g. "open, opens, opened, and opening") as separate words. In languages such as Turkish, however, such an approach is not possible, as each dictionary entry has thousands of possible word forms.
- ***Named Entity Recognition (NER):*** Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is (e.g. person, location, organization). Note that, although capitalization can aid in recognizing named entities in languages such as English, this information cannot aid in determining the type of named entity, and in any case is often inaccurate or insufficient. For example, the first word of a sentence is also capitalized, and named entities often span several words, only some of which are capitalized. Furthermore, many other languages in non-Western scripts (e.g. Chinese or Arabic) do not have any capitalization at all, and even languages with capitalization may not consistently use it to distinguish names.

Notes

For example, German capitalizes all nouns, regardless of whether they refer to names, and French and Spanish do not capitalize names that serve as adjectives.

- ***Natural Language Generation:*** Convert information from computer databases into readable human language.
- ***Natural Language Understanding:*** Convert chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate. Natural language understanding involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organized notations of natural languages concepts. Introduction and creation of language meta model and ontology are efficient however empirical solutions. An explicit formalization of natural languages semantics without confusions with implicit assumptions such as Closed World Assumption (CWA) vs. open world assumption, or subjective Yes/No vs. objective True/False is expected for the construction of a basis of semantics formalization.

*Caution* Natural Language processing can be done if better algorithm is designed.

*Task* Apply NLP to find the trait of human.

### Self Assessment

State whether the following statements are true or false:

25. Co reference resolution produce a readable summary of a chunk of text.
26. Morphological segmentation separate words into individual morphemes and identify the class of the morphemes.

## 6.14 Summary

- A formal language can be thought of as identical to the set of its well-formed formulas. The set of well-formed formulas may be broadly divided into theorems and non-theorems.
- A variation, closely related to Temporal or Chronological or Tense logics, are Modal logics based upon "topology", "place", or "spatial position".
- A well-formed formula, shortly WFFS, often simply formula, is a word (i.e. a finite sequence of symbols from a given alphabet) which is part of a formal language.
- Fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic on which the modern computer is based. Natural language (like most other activities in life and indeed the universe) is not easily translated into the absolute terms of 0 and 1.
- In logic, a rule of inference, inference rule, or transformation rule is the act of drawing a conclusion based on the form of premises interpreted as a function which takes premises, analyses their syntax, and returns a conclusion (or conclusions).
- In mathematical logic and automated theorem proving, resolution is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic.

Notes

- Propositional logic is a formal system in mathematics and logic. Other names for the system are propositional calculus and sentential calculus.
- Reason maintenance is a knowledge representation approach to efficient handling of inferred information that is explicitly stored.
- Symbolic Logic is designed to cultivate research on the borders of logic, philosophy, and the sciences, and to support substantive interactions between these disciplines.
- The clausal normal form (or clause normal form, conjunctive normal form, CNF) of a logical formula is used in logic programming and many theorem proving systems.

## 6.15 Keywords

***Chomsky-type Grammars:*** Within the field of computer science, specifically in the area of formal languages, the Chomsky hierarchy (occasionally referred to as Chomsky – Schützenberger hierarchy) is a containment hierarchy of classes of formal grammars.

***Disjunctions:*** Logical disjunction is an operation on two logical values, typically the values of two propositions, that produces a value of false if and only if both of its operands are false. More generally a disjunction is a logical formula that can have one or more literals separated only by ORs. A single literal is often considered to be a degenerate disjunction.

***Domain:*** The term domain can refer either to a local sub-network or to descriptors for sites on the Internet.

***Inference Engine:*** In computer science, and specifically the branches of knowledge engineering and artificial intelligence, an inference engine is a computer program that tries to derive answers from a knowledge base. It is the "brain" that expert systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions.

***Non-classical Logics:*** A non-classical logic (and sometimes alternative logics) is the name given to formal systems which differ in a significant way from standard logical systems such as propositional and predicate logic

***Topology:*** The physical topology of a network refers to the configuration of cables, computers, and other peripherals.

## 6.16 Review Questions

1. Explain syntax logic with an example.
2. Define FOL.
3. What is the purpose of well-formed formula (WFF)?
4. Describe the clausal normal form and its conversion.
5. Write the inference rule. Explain each rule.
6. Explain the resolution logic with an example.
7. Write few lines on truth maintenance system.
8. Describe the purpose of circumscription logic.
9. Why does the point wise circumscription use?
10. Explain Fuzzy Logic with an example.
11. Enumerate NLP with an example.
12. Draw a Fuzzy Logic Transition Diagram.

## Answers: Self Assessment

Notes

| | | | |
|---|---|---|---|
| 1. | False | 2. | True |
| 3. | True | 4. | False |
| 5. | True | 6. | True |
| 7. | True | 8. | False |
| 9. | False | 10. | True |
| 11. | True | 12. | True |
| 13. | True | 14. | False |
| 15. | True | 16. | True |
| 17. | False | 18. | False |
| 19. | False | 20. | False |
| 21. | True | 22. | True |
| 23. | True | 24. | False |
| 25. | False | 26. | True |

## 6.17 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://people.hofstra.edu/stefan_waner/realworld/logic/logic1.html

http://vedyadhara.ignou.ac.in/wiki/images/1/11/B2U2mcse-003.pdf

http://www.academia.edu/2157642/Rational_inference_deductive_inductive_and_probabilistic_thinking

http://www.gelbukh.com/clbook/

http://www.webpages.uidaho.edu/~morourke/404-phil/Summer-99/Handouts/Philosophical/Non-Deductive-Inference.htm

http://www4.uwsp.edu/philosophy/dwarren/WhatIsFormalLogic/WhatIsFormalLogicNarration.pdf

https://wiki.umiacs.umd.edu/clip/index.php/Main_Page

Notes

# Unit 7: Probabilistic Reasoning

**CONTENTS**



## Objectives

After studying this unit, you will be able to:

- Discuss the Bayesian Probabilistic Inference
- Describe the Possible World Representations
- Explain the Dempster – Shafer Theory
- Identify various Heuristic Reasoning Methods

Notes

## Introduction

In statistics, Bayesian inference is a method of inference in which Bays'' rule is used to update the probability estimate for a hypothesis as additional evidence is learned. Bayesian updating is an important technique throughout statistics, and especially in mathematical statistics. For some cases, exhibiting a Bayesian derivation for a statistical method automatically ensures that the method works as well as any competing method. Bayesian updating is especially important in the dynamic analysis of a sequence of data. Bayesian inference has found application in a range of fields including science, engineering, philosophy, medicine, and law.

In the philosophy of decision theory, Bayesian inference is closely related to discussions of subjective probability, often called "Bayesian probability". Bayesian probability provides a rational method for updating beliefs; however, non-Bayesian updating rules are compatible with rationality, according to philosophers Ian Hacking and Bas van Fraassen.

## 7.1 Bayesian Probabilistic Inference

Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a "likelihood function" derived from a probability model for the data to be observed. Bayesian inference computes the posterior probability according to Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

where

- $\mid$ means given.
- $H$ stands for any hypothesis whose probability may be affected by data (called evidence below). Often there are competing hypotheses, from which one chooses the most probable.
- the evidence $E$ corresponds to data that were not used in computing the prior probability.
- $P(H)$, the prior probability, is the probability of $H$ before $E$ is observed. This indicates one's preconceived beliefs about how likely different hypotheses are, absent evidence regarding the instance under study.
- $P(H \mid E)$, the posterior probability, is the probability of $H$ given $E$, i.e., after $E$ is observed. This tells us what we want to know: the probability of a hypothesis given the observed evidence.
- $P(E \mid H)$, the probability of observing $E$ given $H$, is also known as the likelihood. It indicates the compatibility of the evidence with the given hypothesis.
- $P(E)$ is sometimes termed the marginal likelihood or "model evidence". This factor is the same for all possible hypotheses being considered. (This can be seen by the fact that the hypothesis $H$ does not appear anywhere in the symbol, unlike for all the other factors.) This means that this factor does not enter into determining the relative probabilities of different hypotheses.

Note that what affects the value of $P(H \mid E)$ for different values of $H$ is only the factors $P(H)$ and $P(E \mid H)$, which both appear in the numerator, and hence the posterior probability is proportional to both. In words:

- (more exactly) The posterior probability of a hypothesis is determined by a combination of the inherent likeliness of a hypothesis (the prior) and the compatibility of the observed evidence with the hypothesis (the likelihood).

Notes

- (more concisely) Posterior is proportional to prior times likelihood.

Note that Bayes' rule can also be written as follows:

$$P(H \mid E) = \frac{P(E \mid H)}{P(E)} \cdot P(H)$$

where the factor $\frac{P(E \mid H)}{P(E)}$ represents the impact of $E$ on the probability of $H$.

Informal, Rationally, Bayes' rule makes a great deal of sense. If the evidence does not match up with a hypothesis, one should reject the hypothesis. But if a hypothesis is extremely unlikely a priori, one should also reject it, even if the evidence does appear to match up.

*Example:* Imagine that I have various hypotheses about the nature of a newborn baby of a friend, including:

- $H_1$ : the baby is a brown-haired boy.
- $H_2$ : the baby is a blond-haired girl.
- $H_3$ : the baby is a dog.

Then consider two scenarios:

I'm presented with evidence in the form of a picture of a blond-haired baby girl. I find this evidence supports $H_2$ and opposes $H_1$ and $H_3$.

I'm presented with evidence in the form of a picture of a baby dog. Although this evidence, treated in isolation, supports $H_3$, my prior belief in this hypothesis (that a human can give birth to a dog) is extremely small, so the posterior probability is nevertheless small.

The critical point about Bayesian inference, then, is that it provides a principled way of combining new evidence with prior beliefs, through the application of Bayes' rule. (Contrast this with frequentist inference, which relies only on the evidence as a whole, with no reference to prior beliefs.) Furthermore, Bayes' rule can be applied iteratively: after observing some evidence, the resulting posterior probability can then be treated as a prior probability, and a new posterior probability computed from new evidence. This allows for Bayesian principles to be applied to various kinds of evidence, whether viewed all at once or over time. This procedure is termed "Bayesian updating".

### Bayesian Updating

Bayesian updating is widely used and computationally convenient. However, it is not the only updating rule that might be considered "rational".

Ian Hacking noted that traditional "Dutch book" arguments did not specify Bayesian updating: they left open the possibility that non-Bayesian updating rules could avoid Dutch books. Hacking wrote "And neither the Dutch book argument, nor any other in the personalist arsenal of proofs of the probability axioms, entails the dynamic assumption. Not one entails Bayesianism. So the personalist requires the dynamic assumption to be Bayesian. It is true that in consistency a personalist could abandon the Bayesian model of learning from experience. Salt could lose its savour."

*Did u know?* There are non-Bayesian updating rules that also avoid Dutch books. The additional hypotheses needed to uniquely require Bayesian updating have been deemed to be substantial, complicated, and unsatisfactory.

Notes

### 7.1.1 Inference

Bayesian probability is one of the different interpretations of the concept of probability and belongs to the category of evidential probabilities. The Bayesian interpretation of probability can be seen as an extension of the branch of mathematical logic known as propositional logic that enables reasoning with propositions whose truth or falsity is uncertain. To evaluate the probability of a hypothesis, the Bayesian probabilist specifies some prior probability, which is then updated in the light of new, relevant data.

The Bayesian interpretation provides a standard set of procedures and formulae to perform this calculation. Bayesian probability interprets the concept of probability as "an abstract concept, a quantity that we assign theoretically, for the purpose of representing a state of knowledge, or that we calculate from previously assigned probabilities," in contrast to interpreting it as a frequency or "propensity" of some phenomenon.

The term "Bayesian" refers to the 18th century mathematician and theologian Thomas Bayes, who provided the first mathematical treatment of a non-trivial problem of Bayesian inference. Nevertheless, it was the French mathematician Pierre-Simon Laplace who pioneered and popularised what is now called Bayesian probability.

Broadly speaking, there are two views on Bayesian probability that interpret the probability concept in different ways. According to the objectivist view, the rules of Bayesian statistics can be justified by requirements of rationality and consistency and interpreted as an extension of logic. According to the subjectivist view, probability quantifies a "personal belief". Many modern machine learning methods are based on objectivist Bayesian principles. In the Bayesian view, a probability is assigned to a hypothesis, whereas under the frequentist view, a hypothesis is typically being assigned a probability.

### 7.1.2 Brief Introduction to the Bayesian Approach

Broadly speaking, there are two views on Bayesian probability that interpret the 'probability' concept in different ways. For objectivists, probability objectively measures the plausibility of propositions, i.e. the probability of a proposition corresponds to a reasonable belief everyone (even a "robot") sharing the same knowledge should share in accordance with the rules of Bayesian statistics, which can be justified by requirements of rationality and consistency. For subjectivists, probability corresponds to a 'personal belief'. For subjectivists, rationality and coherence constrain the probabilities a subject may have, but allow for substantial variation within those constraints. The objective and subjective variants of Bayesian probability differ mainly in their interpretation and construction of the prior probability.

*Notes* The use of Bayesian probabilities as the basis of Bayesian inference has been supported by several arguments, such as the Cox axioms, the Dutch book argument, arguments based on decision theory and de Finetti's theorem.

#### Axiomatic Approach

Richard T. Cox showed that Bayesian updating follows from several axioms, including two functional equations and a controversial hypothesis of differentiability. It is known that Cox's 1961 development (mainly copied by Jay Ness) is non-rigorous, and in fact a counter example has been found by Halpern. The assumption of differentiability or even continuity is questionable since the Boolean algebra of statements may only be finite. Other axiomatizations have been suggested by various authors to make the theory more rigorous.

Notes

### Dutch Book Approach

The Dutch book argument was proposed by de Finetti, and is based on betting. A Dutch book is made when a clever gambler places a set of bets that guarantee a profit, no matter what the outcome of the bets. If a bookmaker follows the rules of the Bayesian calculus in the construction of his odds, a Dutch book cannot be made.

However, Ian Hacking noted that traditional Dutch book arguments did not specify Bayesian updating: they left open the possibility that non-Bayesian updating rules could avoid Dutch books. For example, Hacking writes" And neither the Dutch book argument, nor any other in the personalist arsenal of proofs of the probability axioms, entails the dynamic assumption. Not one entails Bayesianism. So the personalist requires the dynamic assumption to be Bayesian. It is true that in consistency a personalist could abandon the Bayesian model of learning from experience. Salt could lose its savour."

*Caution* There are non-Bayesian updating rules that also avoid Dutch books. The additional hypotheses sufficient to (uniquely) specify Bayesian updating are substantial, complicated, and unsatisfactory.

### Decision Theory Approach

A decision-theoretic justification of the use of Bayesian inference (and hence of Bayesian probabilities) was given by Abraham Wald, who proved that every admissible statistical procedure is either a Bayesian procedure or a limit of Bayesian procedures. Conversely, every Bayesian procedure is admissible.

## 7.1.3 An Alternative to Bayes: The Stanford Certainty Factor Algebra

It is a more general approach to representing uncertainty than the Bayesian approach. Particularly useful when decision to be made is based on the amount of evidence that has been collected. It is appropriate for combining expert opinions, since experts do differ in their opinions with a certain degree of ignorance. This Approach distinguishes between uncertainty and ignorance by creating belief functions. This also assumes that the sources of information to be combined are statistically independent. 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.

MB and MD can be tied together

CF(H | E) = MB(H | E) - MD(H | E)

As CF approaches 1 the confidence for H is stronger

As CF approaches -1 the confidence against H is stronger

As CF approaches 0 there is little support for belief or disbelief in H

A basic probability m(S), a belief BELIEF(S) and a plausible belief PLAUSIBILITY(S) all have value in the interval [0,1].

### Combining Beliefs

To combine multiple sources of evidence to a single (or multiple) hypothesis do the following: Suppose $M_1$ and $M_2$ are two belief functions. Let X be the set of subsets of W to which $M_1$ assigns

**Notes**

a nonzero value and let Y be a similar set for $M_2$. Then to get a new belief function $M_3$ from the combination of $M_1$ and $M_2$ beliefs in and we do:

$$M_3 = \Sigma_{X\cap Y=Z} M_1(X)\ M_2(y)/1\text{-}\Sigma_{X\cap Y=\phi} M_1(X)\ M_2(y)$$

### Stanford Certainty Factor Algebra

This approach was used in MYCIN for first time. It is based on Several Observations. Here Facts and the outcome of rules are given confidence factor between -1 and +1

+1 means fact is known to be true.

-1 means fact is known to be false.

Here, the knowledge content of the rules is more important than the algebra of confidences holding the system together. In this approach, confidence measures correspond to informal assessments by humans such as "it is probably true" or "it is often the case. It also separates "confidence for" from "confidence against".

MB(H|E) – The measure of belief in hypothesis H given evidence E.

MD(H|E) – The measure of disbelief of H given E.

Where the belief brings together all the evidence that would lead us to believe in P with some certainty and the plausibility brings together the evidence that is compatible with P and is not inconsistent with it.

Rules of the algebra:

CF(P1 & P2) = Min(CF(P1),CF(P2))

CF(P1 OR P2) = Max(CF(P1),CF(P2))

Given: IF P THEN Q with CF(R)

CF(Q)=CF(P) * CF(R)

If two rules R1 and R2 support Q with CF(QR1) and CF(QR2) then to find the actual CF(Q):

If both +ve: CF(Q)=CF(QR1) + CF(QR2) - CF(QR1) * CF(QR2)

If both -ve: CF(Q)=CF(QR1) + CF(QR2) + CF(QR1) * CF(QR2)

If of opposite signs: CF(Q)= (CF(QR1) + CF(QR2))/(1 - Min(|CF(QR1)|, |CF(QR2)|))

### Casual Networks

This approach accedes to the kind of descriptions provided by experts (e.g. CASNET, ABEL). Causal models depict relationships as links between nodes in a graph or a network of nodes. It is critical to explanatory power. Mostly developed for medical application, these networks provide multiple levels of detail to link clinical findings to pathophysiological state. They provide focus mechanisms for diagnostic investigation. It exhibits further information seeking behavior and provides mechanism for accounting for disease interaction.

### Limitations

1. A lot of detail required.
2. Not immediately clear how to choose which level to reason at and how to combine reasoning at different levels. Not a purely mechanistic approach, still unclear when the problem space is adequately covered.

Notes

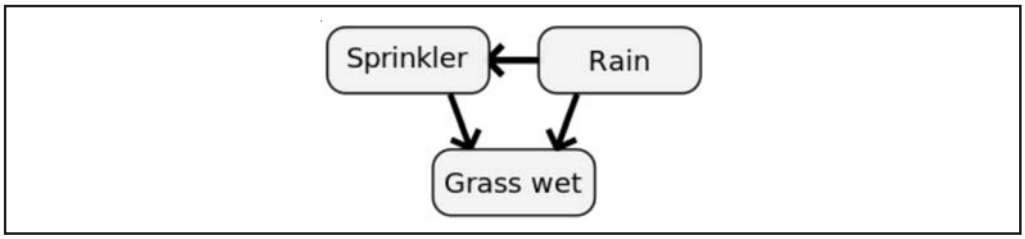
3. At a high level, our goal is to efficiently represent and operate upon a joint distribution P r over a set of random variables X = fX1;:::; Xng. Even if these variables are binary-valued, a naive representation of the joint distribution requires the specification of 2 n numbers (the probabilities of the 2 n different assignments to the variables), which would be infeasible except for very small n. Fortunately, most real-world application domains exhibit a high degree of structure in this joint distribution that allows us to factor the representation of the distribution into modular components.

### 7.1.4 Graphical Models for Uncertain Reasoning

A Bayesian network, Bayes network, belief network, Bayes(ian) model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are directed acyclic graphs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example, if the parents are *m* Boolean variables then the probability function could be represented by a table of $2^m$ entries, one entry for each of the $2^m$ possible combinations of its parents being true or false. Similar ideas may be applied to undirected, and possibly cyclic, graphs; such are called Markov networks.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables (e.g. speech signals or protein sequences) are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.




Inference, or model evaluation, is the process of updating probabilities of outcomes based upon the relationships in the model and the evidence known about the situation at hand. When a Bayesian model is actually used, the end user applies evidence about recent events or observations. This information is applied to the model by "instantiating" or "clamping" a variable to a state that is consistent with the observation. Then the mathematical mechanics are performed to update the probabilities of all the other variables that are connected to the variable representing the new evidence.

*Notes* After inference, the updated probabilities reflect the new levels of belief in (or probabilities of) all possible outcomes coded in the model. These beliefs are mediated by the original assessment of belief performed by the author of the model.

Notes

The beliefs originally encoded in the model are known as prior probabilities, because they are entered before any evidence is known about the situation. The beliefs computed after evidence is entered are known as posterior probabilities, because they reflect the levels of belief computed in light of the new evidence.

### Parents and Children: Introduction to Diagrams

To recap, every variable in the real world situation is represented by a Bayesian variable. Each such variable describes a set of states that represent all possible distinct situations for the variable.

Once the set of variables and their states are known, the next step is to define the causal relationships among them. For any variable, this means asking the questions:

- What other variables (if any) directly influence this variable?
- What other variables (if any) are directly influenced by this variable?

In a standard Bayesian belief network, each variable is represented by a colored ellipse; this graphical representation is called a node.

Each causal influence relationship is described by a line (or arc) connecting the influencing variable to the influenced variable. The influence arc has a terminating arrowhead pointing to the influenced variable.

The common terminology, then is as follows:

- A node is a Bayesian variable.
- An arc connects a parent (influencing) node to a child (influenced) node.

#### *Belief Network Creation*

To create a belief network, then, the following steps are necessary.

1. Create a set of variables representing the distinct elements of the situation being modeled.
2. For each such variable, define the set of outcomes or states that each can have. This set is referred to in the mathematical literature as "mutually exclusive and exhaustive," meaning that it must cover all possibilities for the variable, and that no important distinctions are shared between states.
3. Establish the causal dependency relationships between the variables. This involves creating arcs (lines with arrowheads) leading from the parent variable to the child variable.
4. Assess the prior probabilities. This means supplying the model with numeric probabilities for each variable in light of the number of parents the variable was given in Step 3.

After the model has been developed, MSBNX allows you to model or mimic real-world situations by entering or removing evidence on the model. As you do so, there are various methods for displaying the resulting probabilities.

## 7.1.5 The Bayesian Belief Network

To keep the formalism simple and general, we develop the notion of features without considering time explicitly. Constraint satisfaction problems will be described in terms of possible worlds. When we are not modeling change, there is a direct one-to-one correspondence between features and variables, and between states and possible worlds. A possible world is a possible way the world (the real world or some imaginary world) could be. For example, when representing a crossword puzzle, the possible worlds correspond to the ways the crossword could be filled out.

Notes

In the electrical environment, a possible world specifies the position of every switch and the status of every component.

Possible worlds are described by algebraic variables. An algebraic variable is a symbol used to denote features of possible worlds. Algebraic variables will be written starting with an uppercase letter. Each algebraic variable V has an associated domain, dom(V), which is the set of values the variable can take on.

### 7.1.6 Inference with a Bayesian Belief Network

Internal to a computer, a symbol is just a sequence of bits that can be distinguished from other symbols. Some symbols have a fixed interpretation, for example, symbols that represent numbers and symbols that represent characters. Symbols that do not have fixed meaning appear in many programming languages. In Java, starting from Java 1.5, they are called enumeration types. Lisp refers to them as atoms. Usually, they are implemented as indexes into a symbol table that gives the name to print out. The only operation performed on these symbols is equality to determine if two symbols are the same or not. To a user of a computer, symbols have meanings. A person who inputs constraints or interprets the output associates meanings with the symbols that make up the constraints or the outputs. He or she associates a symbol with some concept or object in the world. For example, the variable Harry's Height to the computer, is just a sequence of bits. It has no relationship to Harry's Weight or Sue's Height. To a person, this variable may mean the height, in particular units, of a particular person at a particular time.

The meaning associated with a variable-value pair must satisfy the clarity principle: an omniscient agent - a fictitious agent who knows the truth and the meanings associated with all of the symbols - should be able to determine the value of each variable.

*Example:* The height of Harry only satisfies the clarity principle if the particular person being referred to and the particular time are specified as well as the units. For example, we may want to reason about the height, in centimeters, of Harry Potter at the start of the second movie of J. K. Rowling's book. This is different from the height, in inches, of Harry Potter at the end of the same movie (although they are, of course, related). If you want to refer to Harry's height at two different times, you must have two different variables. You should have a consistent meaning. When stating constraints, you must have the same meaning for the same variable and the same values, and you can use this meaning to interpret the output.

The bottom line is that symbols can have meanings because we give them meanings. For this unit, assume that the computer does not know what the symbols mean. A computer can only know what a symbol means if it can perceive and manipulate the environment.

### Self Assessment

State whether the following statements are true or false:

1. Bayesian updating is widely used and computationally convenient.
2. The term "Bayesian" refers to the 20th century mathematician and theologian Thomas Bayes.
3. There are four views on Bayesian probability that interpret the probability concept in different ways.
4. A decision-theoretic justification of the use of Bayesian inference (and hence of Bayesian probabilities) was given by Richard T. Cox.

## 7.2 Possible World Representations

Notes

A discrete variable is one whose domain is finite or countably infinite. One particular case of a discrete variable is a Boolean variable, which is a variable with domain {true, false}. If X is a Boolean variable, we write X=true as its lowercase equivalent, x, and write X=false as ¬x. We can also have variables that are not discrete; for example, a variable whose domain corresponds to a subset of the real line is a continuous variable.

*Example:* The variable Class_time may denote the starting time for a particular class. The domain of Class_time may be the following set of possible times:

dom(Class_time)={8, 9, 10, 11, 12, 1, 2, 3, 4, 5}.

The variable Height_joe may refer to the height of a particular person at a particular time and have as its domain the set of real numbers, in some range, that represent the height in centimeters. Raining may be a Boolean random variable with value true if it is raining at a particular time.

### 7.2.1 Pointing to the Subject

Knowledge Representation is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inference from those knowledge elements, creating new elements of knowledge. The KR can be made to be independent of the underlying knowledge model or knowledge base system (KBS) such as a semantic network.

Knowledge representation (KR) research involves analysis of how to reason accurately and effectively and how best to use a set of symbols to represent a set of facts within a knowledge domain. A symbol vocabulary and a system of logic are combined to enable inferences about elements in the KR to create new KR sentences. Logic is used to supply formal semantics of how reasoning functions should be applied to the symbols in the KR system. Logic is also used to define how operators can process and reshape the knowledge. Examples of operators and operations include negation, conjunction, adverbs, adjectives, quantifiers and modal operators. The logic is interpretation theory. These elements – symbols, operators, and interpretation theory – are what give sequences of symbols meaning within a KR.

A key parameter in choosing or creating a KR is its expressivity. The more expressive a KR, the easier and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR. However, more expressive languages are likely to require more complex logic and algorithms to construct equivalent inferences. A highly expressive KR is also less likely to be complete and consistent. Less expressive KRs may be both complete and consistent. Auto epistemic temporal modal logic is a highly expressive KR system, encompassing meaningful chunks of knowledge with brief, simple symbol sequences (sentences). Propositional logic is much less expressive but highly consistent and complete and can efficiently produce inferences with minimal algorithm complexity. Nonetheless, only the limitations of an underlying knowledge base affect the ease with which inferences may ultimately be made (once the appropriate KR has been found). This is because a knowledge set may be exported from a knowledge model or knowledge base system (KBS) into different KRs, with different degrees of expressiveness, completeness, and consistency. If a particular KR is inadequate in some way, that set of problematic KR elements may be transformed by importing them into a KBS, modified and operated on to eliminate the problematic elements or augmented with additional knowledge imported from other sources, and then exported into a different, more appropriate KR.

In applying KR systems to practical problems, the complexity of the problem may exceed the resource constraints or the capabilities of the KR system.

Notes

*Did u know?* Recent developments in KR include the concept of the Semantic Web, and development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL) and Web Ontology Language (OWL).

There are several KR techniques such as frames, rules, tagging, and semantic networks which originated in cognitive science. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to facilitate reasoning, inferencing, or drawing conclusions. A good KR must be both declarative and procedural knowledge. What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand:

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world."

### 7.2.2 Common Realms of Discourse

The Dempster – Shafer theory (DST) is a mathematical theory of evidence. It allows one to combine evidence from different sources and arrive at a degree of belief (represented by a belief function) that takes into account all the available evidence. The theory was first developed by Arthur P. Dempster and Glenn Shafer.

In a narrow sense, the term "Dempster – Shafer theory" refers to the original conception of the theory by Dempster and Shafer. However, it is more common to use the term in the wider sense of the same general approach, as adapted to specific kinds of situations. In particular, many authors have proposed different rules for combining evidence, often with a view to handling conflicts in evidence better.

Dempster – Shafer theory is a generalization of the Bayesian theory of subjective probability; whereas the latter requires probabilities for each question of interest, belief functions base degrees of belief (or confidence, or trust) for one question on the probabilities for a related question. These degrees of belief may or may not have the mathematical properties of probabilities; how much they differ depends on how closely the two questions are related. Put another way, it is a way of representing epistemic plausibilities but it can yield answers that contradict those arrived at using probability theory.

Often used as a method of sensor fusion, Dempster – Shafer theory is based on two ideas: obtaining degrees of belief for one question from subjective probabilities for a related question, and Dempster's rule for combining such degrees of belief when they are based on independent

Notes

items of evidence. In essence, the degree of belief in a proposition depends primarily upon the number of answers (to the related questions) containing the proposition, and the subjective probability of each answer. Also contributing are the rules of combination that reflect general assumptions about the data.

In this formalism, a degree of belief (also referred to as a mass) is represented as a belief function rather than a Bayesian probability distribution. Probability values are assigned to sets of possibilities rather than single events: their appeal rests on the fact they naturally encode evidence in favor of propositions.

*Notes* Dempster – Shafer theory assigns its masses to all of the non-empty subsets of the entities that compose a system

### Belief and Plausibility

Shafer's framework allows for belief about propositions to be represented as intervals, bounded by two values, belief (or support) and plausibility:

belief ≤ plausibility

Belief in a hypothesis is constituted by the sum of the masses of all sets enclosed by it (i.e. the sum of the masses of all subsets of the hypothesis). It is the amount of belief that directly supports a given hypothesis at least in part, forming a lower bound. Belief (usually denoted Bel) measures the strength of the evidence in favor of a set of propositions. It ranges from 0 (indicating no evidence) to 1 (denoting certainty). Plausibility is 1 minus the sum of the masses of all sets whose intersection with the hypothesis is empty. It is an upper bound on the possibility that the hypothesis could be true, i.e. it "could possibly be the true state of the system" up to that value, because there is only so much evidence that contradicts that hypothesis. Plausibility (denoted by Pl) is defined to be Pl(s)=1-Bel(~s). It also ranges from 0 to 1 and measures the extent to which evidence in favor of ~s leaves room for belief in s. For example, suppose we have a belief of 0.5 and a plausibility of 0.8 for a proposition, say "the cat in the box is dead." This means that we have evidence that allows us to state strongly that the proposition is true with a confidence of 0.5. However, the evidence contrary to that hypothesis (i.e. "the cat is alive") only has a confidence of 0.2. The remaining mass of 0.3 (the gap between the 0.5 supporting evidence on the one hand, and the 0.2 contrary evidence on the other) is "indeterminate," meaning that the cat could either be dead or alive. This interval represents the level of uncertainty based on the evidence in your system.

| Hypothesis | Mass | Belief | Plausibility |
|---|---|---|---|
| Null (neither alive nor dead) | 0 | 0 | 0 |
| Alive | 0.2 | 0.2 | 0.5 |
| Dead | 0.5 | 0.5 | 0.8 |
| Either (alive or dead) | 0.3 | 1.0 | 1.0 |

The null hypothesis is set to zero by definition (it corresponds to "no solution"). The orthogonal hypotheses "Alive" and "Dead" have probabilities of 0.2 and 0.5, respectively. This could correspond to "Live/Dead Cat Detector" signals, which have respective reliabilities of 0.2 and 0.5. Finally, the all-encompassing "Either" hypothesis (which simply acknowledges there is a cat in the box) picks up the slack so that the sum of the masses is 1. The belief for the "Alive" and "Dead" hypotheses matches their corresponding masses because they have no subsets; belief for "Either" consists of the sum of all three masses (Either, Alive, and Dead) because "Alive" and

Notes

"Dead" are each subsets of "Either". The "Alive" plausibility is 1 – m (Dead) and the "Dead" plausibility is 1 – m (Alive). Finally, the "Either" plausibility sums m(Alive) + m(Dead) + m(Either). The universal hypothesis ("Either") will always have 100% belief and plausibility—it acts as a checksum of sorts.

Here is a somewhat more elaborate example where the behavior of belief and plausibility begins to emerge. We're looking through a variety of detector systems at a single faraway signal light, which can only be coloured in one of three colours (red, yellow, or green):

| Hypothesis | Mass | Belief | Plausibility |
|---|---|---|---|
| Null | 0 | 0 | 0 |
| Red | 0.35 | 0.35 | 0.56 |
| Yellow | 0.25 | 0.25 | 0.45 |
| Green | 0.15 | 0.15 | 0.34 |
| Red or Yellow | 0.06 | 0.66 | 0.85 |
| Red or Green | 0.05 | 0.55 | 0.75 |
| Yellow or Green | 0.04 | 0.44 | 0.65 |
| Any | 0.1 | 1.0 | 1.0 |

Events of this kind would not be modeled as disjoint sets in probability space as they are here in mass assignment space. Rather the event "Red or Yellow" would be considered as the union of the events "Red" and "Yellow", and P(Red or Yellow)· P(Yellow), and P(Any)=1, where Any refers to Red or Yellow or Green. In DST, the mass assigned to Any refers to the proportion of evidence that can't be assigned to any of the other states, which here means evidence that says there is a light but doesn't say anything about what color it is. In this example, the proportion of evidence that shows the light is either Red or Green is given a mass of 0.05. Such evidence might, for example, be obtained from a R/G color blind person. DST lets us extract the value of this sensor's evidence. Also, in DST the Null set is considered to have zero mass, meaning here that the signal light system exists and we are examining its possible states, not speculating as to whether it exists at all.

*Caution* Deduct uncertain information for knowledge.

*Task* List five cases of uncertainty of information.

### 7.2.3 Knowledge Representation

The problem we now face is how to combine two independent sets of probability mass assignments in specific situations. In case, different sources express their beliefs over the frame in terms of belief constraints such as in case of giving hints or in case of expressing preferences, then Dempster's rule of combination is the appropriate fusion operator. This rule derives common shared belief between multiple sources and ignores all the conflicting (non-shared) belief through a normalization factor. Use of that rule in other situations than that of combining belief constraints has come under serious criticism, such as in case of fusing separate beliefs estimates from multiple sources that are to be integrated in a cumulative manner, and not as constraints. Cumulative fusion means that all probability masses from the different sources are reflected in the derived belief, so no probability mass is ignored.

Notes

Specifically, the combination (called the joint mass) is calculated from the two sets of masses $m_1$ and $m_2$ in the following manner:

$$m_{1,2}(\varnothing) = 0$$

$$m_{1,2}(A) = (m_1 \oplus m_2)(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \varnothing} m_1(B) m_2(C)$$

where

$$K = \sum_{B \cap C = \varnothing} m_1(B) m_2(C).$$

$K$ is a measure of the amount of conflict between the two mass sets.

### Effects of Conflict

The normalization factor above, 1 – K, has the effect of completely ignoring conflict and attributing any mass associated with conflict to the null set. This combination rule for evidence can therefore produce counterintuitive results, as we show next.

*Example:* **Producing Correct Results in Case of High Conflict**

The following example shows how Dempster's rule produces intuitive results when applied in a preference fusion situation, even when there is high conflict.

Suppose that two friends, Alice and Bob, want to see a film at the cinema one evening, and that there are only three films showing: X, Y and Z. Alice expresses her preference for film X with probability 0.99, and her preference for film Y with a probability of only 0.01. Bob expresses his preference for film Z with probability 0.99, and his preference for film Y with a probability of only 0.01. When combining the preferences with Dempster's rule of combination it turns out that their combined preference results in probability 1.0 for film Y, because it is the only film that they both agree to see. Dempster's rule of combination produces intuitive results even in case of totally conflicting beliefs when interpreted in this way. Assume that Alice prefers film X with probability 1.0, and that Bob prefers film Z with probability 1.0. When trying to combine their preferences with Dempster's rule it turns out that it is undefined in this case, which means that there is no solution. This would mean that they can not agree on seeing any film together, so they don't go to the cinema together that evening.

*Example:* **Producing Counter-intuitive Results in Case of High Conflict**

An example with exactly the same numerical values was introduced by Zadeh in 1979, to point out counter-intuitive results generated by Dempster's rule when there is a high degree of conflict. The example goes as follows:

Suppose that one has two equi-reliable doctors and one doctor believes a patient has either a brain tumor—with a probability (i.e. a basic belief assignment - bba's, or mass of belief) of 0.99—or meningitis—with a probability of only 0.01. A second doctor believes the patient has a concussion—with a probability of 0.99—and believes the patient suffers from meningitis—with a probability of only 0.01. Applying Dempster's rule to combine these two sets of masses of belief, one gets finally m(meningitis)=1 (the meningitis is diagnosed with 100 percent of confidence).

Such result goes against the common sense since both doctors agree that there is a little chance that the patient has a meningitis. This example has been the starting point of many research works for trying to find a solid justification for Dempster's rule and for foundations of Dempster – Shafer Theory or to show the inconsistencies of this theory.

Notes

*Example:* Producing Counter-intuitive Results in Case of Low Conflict

The following example shows where Dempster's rule produces a counter-intuitive result, even when there is low conflict.

Suppose that one doctor believes a patient has either a brain tumor, with a probability of 0.99, or meningitis, with a probability of only 0.01. A second doctor also believes the patient has a brain tumor, with a probability of 0.99, and believes the patient suffers from concussion, with a probability of only 0.01. If we calculate m (brain tumor) with Dempster's rule, we obtain

$m$(brain tumor) = Bel(brain tumor) = 1.

This result implies complete support for the diagnosis of a brain tumour, which both doctors believed very likely. The agreement arises from the low degree of conflict between the two sets of evidence comprised by the two doctors' opinions.

In either case, it would be reasonable to expect that:

$m$(brain tumor) < 1 and Bel(brain tumor) < 1.

Since, the existence of non-zero belief probabilities for other diagnoses implies less than complete support for the brain tumour diagnosis.

### Self Assessment

State whether the following statements are true or false:

5. The KR can be made to be dependent of the underlying knowledge model or knowledge base system (KBS) such as a semantic network.
6. The more expressive a KR, the complex and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR.
7. There is only one KR techniques.
8. Belief in a hypothesis is constituted by the sum of the masses of all sets enclosed by it.

## 7.3 The Dempster – Shafer Theory

Potential computational complexity problems

(i) It lacks a well-established decision theory whereas Bayesian decision theory maximizing expected utility is almost universally accepted.

(ii) Experimental comparisons between D – S theory and probability theory seldom done and rather difficult to do; no clear advantage of D – S theory shown.

### 7.3.1 Characteristics of D – S

The public also relies on specific heuristics to form opinions about science and science news. Scientists and communicators often assume that the public objectively accumulates and evaluates scientific information to develop opinions, but research has shown heuristics have a larger effect than specific science knowledge.

*Example:* A 2007 study examined how people in the United States developed opinions about agricultural biotechnology. Their results showed that the public used key heuristics to

Notes

arrive at their opinions rather than specific knowledge of biotechnology. Specifically, the heuristics they used were deference to scientific authority, trust in scientific institutions, and whether they had seen media coverage of biotechnology. Different heuristics were used for different demographic groups, and actual knowledge of biotechnology played a small role in opinion formation.

This also brings up the idea of using current media news as a heuristic. Whatever information has been most recently presented by the media is likely to be more accessible in an individual's mind. This information can then be used as a shortcut in evaluating an issue, and is used heuristically in place of lengthier cognitive processing using past information.

Our use of heuristics may come directly from individuals. Opinions of trusted or elite individuals may themselves become a heuristic. When evaluating a decision or problem, individuals can turn to these trusted or elite individuals for their opinions. Rather than evaluating the information surrounding the decision, the individual uses these trusted opinions as informational shortcuts to make their decisions.

### 7.3.2 Example of D – S Application

Heuristics used when forming opinions can also be ideologically based. A 2008 study looked at the relationship between religion and opinions about nanotechnology. This research found that the more religious the citizens of a country, the less likely they were to support nanotechnology. This suggests that people used religion as a shortcut or heuristic; they were not informed about nanotechnology, but because their religious beliefs cautioned them against some forms of technology, they used an ideological heuristic to form their opinions about an unknown technology.

Different individuals use different heuristics to process the information before them based on their available schema and the framing of the information. Issues may resonate with different schemata depending on the individual and the way the issue is framed.

*Example:* "Drilling for oil" may activate schemata relating to corporate profits, environmental disasters, and exploitation of workers, while "exploring for energy" may activate schemata related to protecting the environment, American pride, and American innovation. These two terms refer to the same activity, but when they are framed differently, different schemata are activated, which results in the use of different heuristics.

### 7.3.3 Combining Evidences

In computer science, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. By trading optimality, completeness, accuracy, and/or precision for speed, a heuristic can quickly produce a solution that is good enough for solving the problem at hand, as opposed to finding all exact solutions in a prohibitively long time.

*Example:* Many real-time anti-virus scanners use heuristic signatures for detecting viruses and other forms of malware. One way of achieving this computational performance gain consists in solving a simpler problem whose solution is also a solution to the more complex problem. Heuristics is used in the A* algorithm whose intent is to find a short path from one node to another. A high-value heuristic computes a path quickly, but the path might not be the shortest. A low-value heuristic computes a path more slowly, but the path becomes shorter.

Notes

### 7.3.4 Advantages and Disadvantages of D – S Theory

Five general advantages are as follows:

- Goal-based evaluation
- Goal-free evaluation
- Responsive evaluation
- Systems evaluation
- Professional review

*Advantage*

- The difficult problem of specifying priors can be avoided
- In addition to uncertainty, also ignorance can be expressed
- It is straightforward to express pieces of evidence with different levels of abstraction
- Dempster's combination rule can be used to combine pieces of evidence

### Self Assessment

State whether the following statements are true or false:

9. Experimental comparisons between D – S theory and probability theory seldom done and rather difficult to do.
10. Opinions of trusted or elite individuals may themselves become a heuristic.
11. Heuristics used when forming opinions can also be ideologically based.
12. The public do not rely on specific heuristics to form opinions about science and science news.

## 7.4 Heuristic Reasoning Methods

Heuristic refers to experience-based techniques for problem solving, learning, and discovery that give a solution which is not guaranteed to be optimal. Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, or common sense.

*Notes* In more precise terms, heuristics are strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines.

Heuristics were also found to be used in the manipulation and creation of cognitive maps. Cognitive maps are internal representations of our physical environment, particularly associated with spacial relationships. These internal representations of our environment are used as memory as a guide in our external environment. It was found that when questioned about maps imaging, distancing, and etc., people commonly made distortions to images. These distortions took shape

Notes

in the regularization of images (i.e., images are represented as more like pure abstract geometric images, though they are irregular in shape). There are several ways that humans form and use cognitive maps. Visual intake is a key part of mapping. The first is by using landmarks. This is where a person uses a mental image to estimate a relationship, usually distance, between two objects. Second, is route-road knowledge, and this is generally developed after a person has performed a task and is relaying the information of that task to another person. Third, is survey.

A person estimates a distance based on a mental image that, to them, might appear like an actual map. This image is generally created when a person's brain begins making image corrections. These are presented in five ways:

1. Right-angle bias is when a person straightens out an image, like mapping an intersection, and begins to give everything 90-degree angles, when in reality it may not be that way.
2. Symmetry heuristic is when people tend to think of shapes, or buildings, as being more symmetrical than they really are.
3. Rotation heuristic is when a person takes a naturally (realistically) distorted image and straightens it out for their mental image.
4. Alignment heuristic is similar to the pervious, where people align objects mentally to make them straighter than they really are.
5. Relative-position heuristic people do not accurately distance landmarks in their mental image based on how well they remember that particular item.

*Did u know?* Another method of creating cognitive maps is by means of auditory intake based on verbal descriptions. Using the mapping based from a person's visual intake, another person can create a mental image, such as directions to a certain location.

### 7.4.1 Problems with Current Approaches to Uncertainty

One of the main reasons for the problems with current methods of environmental management is scientific uncertainty not just its existence, but the radically different expectations and modes of operation that scientists and policymakers have developed to deal with it. To solve this problem, these differences must be exposed and understood, and better methods to incorporate uncertainty into policymaking and environmental management must be designed. To understand the scope of the problem, it is necessary to differentiate between risk, which is an event with a known probability (sometimes referred to as statistical uncertainty), and true uncertainty, which is an event with an unknown probability (sometimes referred to as indeterminacy). For instance, every time you drive your car, you run the risk of having an accident because the probability of car accidents is known with very high certainty. The risk involved in driving is well known because there have been many car accidents with which to calculate the probabilities. These probabilities are known with enough certainty that they are used by insurance companies, for instance, to set rates that will assure those companies of a certain profit.

There is little uncertainty about the possibility of car accidents. If you live near the disposal sight of some newly synthesized toxic chemical, however, your health may be in jeopardy, but no one knows to what extent. Because no one knows the probability of your getting cancer, for instance, or some other disease from this exposure, there is true uncertainty. Most important environmental problems suffer from true uncertainty, not merely risk. Uncertainty may be thought of as a continuum ranging from zero for certain information to intermediate levels for information with statistical uncertainty and known probabilities (risk) to high levels for information with

Notes

true uncertainty or indeterminacy. Risk assessment has become a central guiding principle at the U.S. Environmental Protection Agency (EPA) and other environmental management agencies, but true uncertainty has yet to be adequately incorporated into environmental protection strategies. Scientists treat uncertainty as a given, a characteristic of all information that must be honestly acknowledged and communicated.

Over the years, scientists have developed increasingly sophisticated methods to measure and communicate uncertainty arising from various causes. In general, however, scientists have uncovered more uncertainty rather than the absolute precision that the lay public often mistakenly associates with scientific results. Scientific inquiry can only set boundaries on the limits of knowledge. It can define the edges of the envelope of known possibilities, but often the envelope is very large, and the probabilities of what's inside (the known possibilities) actually occurring can be a complete mystery. For instance, scientists can describe the range of uncertainty about global warming and toxic chemicals and maybe say something about the relative probabilities of different outcomes, but, in most important cases, they cannot say which of the possible outcomes will occur with any degree of accuracy.

*Caution* Deduct uncertain information for knowledge.

*Task* List five cases of uncertainty of information.

### Self Assessment

State whether the following statements are true or false:

13. Cognitive maps are external representations of our physical environment, particularly associated with spacial relationships.
14. Heuristics were also found to be used in the manipulation and creation of cognitive maps.
15. Visual intake is a key part of mapping.
16. Heuristic refers to experience-based techniques for problem solving.

## 7.5 Summary

- The reasoning supporting these networks, based on two simplifying assumptions (that reasoning could not be cyclic and that the causality supporting a child state would be expressed in the links between it and its parent states) made BBN reasoning quite manageable computationally.
- Bayesian belief networks (BBNs) can dramatically reduce the number of parameters of the full Bayesian model and show how the data of a domain (or even the absence of data) can partition and focus reasoning.
- Bayes's theorem requires masses of statistical data in addition to the degrees of belief in preconditions.
- D – S assigns the part of the probability to the entire universe to make the aggregate of all events to 1. This part is called ignorance level.
- Dumpster – Shafer theory allows updating of beliefs based on evidence gathered.

Notes

## 7.6 Keywords

***Frames:*** Frame is a data structure that typically consists of: Frame name, Slot-filler (relations target), Pointers (links) to other Frames, Instantiation Procedure (inheritance, default, consistency).

***Parts:*** A precondition (or IF) and an action (or THEN); if a production's precondition matches the current state of the world, then the production is said to be triggered.

***Predicate Logic:*** Predicate is a function may be TRUE for some arguments, and FALSE for others.

***Production Rules:*** Consists of a set of rules about behavior; a production consists two.

***Scripts:*** The Scripts are linked sentences using frame-like structures; e.g., a record of sequence of events for a given type of occurrence.

***Semantic Networks:*** Semantic net is just a graph, where the nodes represent concepts, and the arcs represent binary relationships between concepts.

## 7.7 Review Questions

1. Define logical reasoning.
2. What is probabilistic reasoning?
3. Write few lines on probabilistic argumentation.
4. Differentiate uncertainty with ignorance.
5. Define derivation of degrees of belief.
6. What are the problems with current approaches to uncertainty?
7. Explain uncertainty in probabilistic reasoning.
8. Explain with an example, intelligent reasoning about uncertainty.

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | False | 4. | False |
| 5. | False | 6. | False |
| 7. | False | 8. | True |
| 9. | True | 10. | True |
| 11. | True | 12. | False |
| 13. | False | 14. | True |
| 15. | True | 16. | True |

Notes

## 7.8 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://arantxa.ii.uam.es/~modonnel/IC/04_3_DempsterShaferI.pdf

http://clear.colorado.edu/~bethard/teaching/csci3202_2008/slides/bayesian-networks.pdf

http://research.microsoft.com/enus/um/redmond/groups/adapt/msbnx/msbnx/basics_of_bayesian_inference.htm

http://www.cs.usask.ca/faculty/horsch/publications/dynBN-UAI90.pdf

http://www.glennshafer.com/assets/downloads/articles/article48.pdf

http://www.pitt.edu/~druzdzel/psfiles/thesis.pdf

# Unit 8: Structured Representation of Knowledge

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Explain the Associative Networks
- Discuss the need and scripts of Frame Structures and Scripts
- Describe Conceptual Dependency (CD) and its applications
- Identify the Object-oriented Representation
- Explain an overview of Object-oriented Systems
- Discuss the Object and Classes
- Explain the Messages and Methods

## Introduction

Representing knowledge using logical formalism, like predicate logic, has several advantages. They can be combined with powerful inference mechanisms like resolution, which makes

Notes

reasoning with facts easy. But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. cannot be described easily. A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

(i) ***Representational Adequacy:*** The ability to represent all kinds of knowledge that are needed in that domain.

(ii) ***Inferential Adequacy:*** The ability to manipulate the represented structure and infer new structures.

(iii) ***Inferential Efficiency:*** The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.

(iv) ***Acquisitional Efficiency:*** The ability to acquire new information easily, either by direct insertion or by program control.

The techniques that have been developed in AI systems to accomplish these objectives fall under two categories:

1. ***Declarative Methods:*** In these, knowledge is represented as static collection of facts which are manipulated by general procedures. Here, the facts need to be stored only one and they can be used in any number of ways. Facts can be easily added to declarative systems without changing the general procedures.

2. ***Procedural Method:*** In these, knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of "How to do things efficiently" can be easily represented.

In practice, most of the knowledge representation employ a combination of both. Most of the knowledge representation structures have been developed to handle programs that handle natural language input. One of the reasons that knowledge structures are so important is that they provide a way to represent information about commonly occurring patterns of things such descriptions are some times called schema. One definition of schema is:

*"Schema refers to an active organization of the past reactions, or of past experience, which must always be supposed to be operating in any well adapted organic response".*

By using schemas, people as well as programs can exploit the fact that the real world is not random. There are several types of schemas that have proved useful in AI programs. They include:

(i) ***Frames:*** Used to describe a collection of attributes that a given object possesses (e.g.: description of a chair).

(ii) ***Scripts:*** Used to describe common sequence of events (e.g.:- a restaurant scene).

(iii) ***Stereotypes:*** Used to described characteristics of people.

(iv) ***Rule Models:*** Used to describe common features shared among a set of rules in a production system.

Frames and scripts are used very extensively in a variety of AI programs. Before selecting any specific knowledge representation structure, the following issues have to be considered.

(i) The basis properties of objects, if any, which are common to every problem domain must be identified and handled appropriately.

(ii) The entire knowledge should be represented as a good set of primitives.

## 8.1 Structured Knowledge Representation

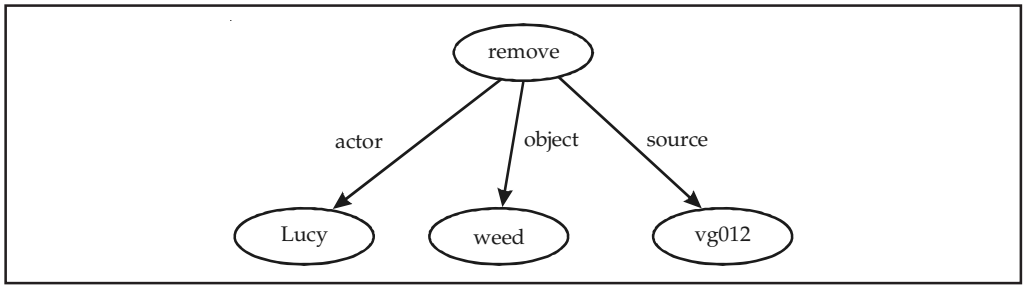
Notes

A means of representing relational knowledge as a labeled directed graph. Each vertex of the graph represents a concept and each label represents a relation between concepts. Access and updating procedures traverse and manipulate the graph. A semantic network is sometimes regarded as a graphical notation for logical formulas.

### Semantic or Associative Networks

Frame-based formalisms grew out of semantic (or associative) networks that were introduced in the sixties as models for human semantic memory. The data organization of a semantic network consists of labeled nodes representing concepts and labeled links representing relations between concepts. Nodes can be used to represent both types and tokens. The is a link hierarchically relates types to types or types to tokens; parts-of links relate concepts and their parts, and in general any relation can be the label of a link. It is a token whereas garden, for instance, is a type.

In a semantic network, each link between two nodes represents a separate proposition, e.g. the fact that Lucy is a person is a proposition separate from the fact that Lucy weeds the garden and the fact that the garden contains vegetables. But suppose that we want to represent relations between more than two nodes. A solution is to allow nodes to represent situations or actions. Each such node has outgoing links representing thematic roles (or cases) for participants in the situation or action.




### Frames

In order to incorporate more structure into semantic networks, frames were introduced by Minsky (1975) as a representation formalism. A frame is basically a encapsulated fragment of a semantic network with conceptually related nodes and links that can be addressed as a unity. For example, one can define a frame for the remove action and the associated roles actor, object and source.

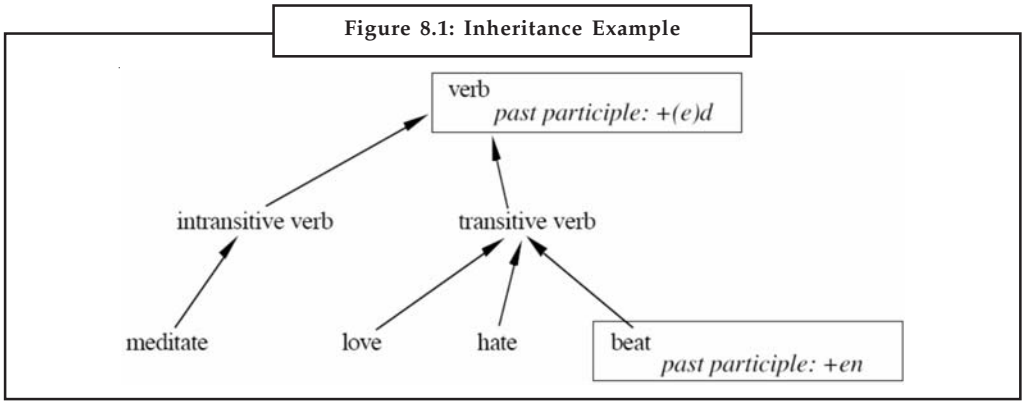
The term frame is not only used for structures representing domain knowledge, but also for structures representing linguistic knowledge. In Dells' (1986) model for phonological encoding, for example, a syllable frame may consist of an onset, a nucleus, and a coda. In the syllable best, these roles are filled by b, e, and st, respectively. Several modifications have been proposed to make frames so powerful that they can be used as high-level programming languages: (1) constraints on the items allowed to fill each slot in the list; (2) recursive frames, the slots of which may contain items that are themselves frames, (3) procedural attachment, in which procedures are attached to roles of the frame to compute the fillers of these roles, and (4) inheritance, which makes inferences along is-a links; this is dealt with in more detail in the next topic.

Notes

### Inheritance

Given the enormous number of linguistic facts that are brought to bear in language processing, it is clearly important to represent this knowledge in an efficient and general way. Inheritance is a powerful technique to represent generalizations over descriptions in a way similar to is-a relations in semantic networks, and to use these generalizations to make inferences. Daelemans et al. (1992) motivate the use of inheritance for linguistic knowledge as follows. Imagine that we are setting out on the job of building a lexicon for English. We begin by encoding everything we know about the verb love, including its syntactic category and how it is conjugated. Next, we turn our attention to the verb hate. Although these words are antonyms, they nevertheless have a lot in common: for example, they are both transitive verbs and have past participle ending on -ed. To save time and space, both can be categorized as transitive verbs and the common information about these kinds of words can be encoded in just one place called, say, transitive verb. The verb love then inherits information from transitive verb. Similarly, when we hit upon intransitive verbs, we collect all information about this kind of verbs in a place intransitive verbs. The next step is to extract from both classes of verbs the common information which is then stored in an even more abstract category verb.

**Figure 8.1: Inheritance Example**




Suppose, we discover the verb *beat,* which is transitive but not regular: it has a past participle ending on *-en.* If we let *beat* inherit from *transitive verb,* we still need to specify the exceptional information, but then we get an inconsistency with the inherited information. The obvious solution is to let the exceptional information override inherited information. This mechanism is called *default* (or *non-monotonic*) inheritance. Default inheritance is incorporated in semantic networks and frame-based representation languages used in AI.

*Did u know?* The sentence production model IPF, is implemented in a frame-based language using default inheritance. Default inheritance is also used in specific linguistic formalisms, for example DATR, a formalism for lexical representation.

### 8.1.1 Market Passing and Spreading Activation

One of the inference mechanisms available in semantic networks is *marker passing,* a process with which *intersection search* can be implemented. This type of search starts from two nodes which pass a marker to those nodes which are linked to them, a process which is repeated for each of those nodes. Whenever a marker originating from one of the original nodes encounters a marker originating from the other node, a path is found representing a association between the two concepts represented by the nodes. This way, semantic associations can be modeled, but marker passing can also be used in networks representing other types of linguistic knowledge.

Notes

A similar inference mechanism in networks is *spreading activation* where instead of discrete symbolic markers, a continuous (numerical) activation level is propagated along the links of a network. A model of lexical retrieval where a semantic network with labeled links is combined with spreading activation.

### 8.1.2 Conceptual Dependency Structures and Conceptual Graphs

Many of the early symbolic AI research on natural language understanding used semantic network or frame-based formalisms to represent its theoretical insights. Schank and his students developed Conceptual Dependency Theory for the description of the meaning of sentences and texts (Schank, 1975; 1980). This theory was based on semantic networks, but defined only a limited number of node types and link types (conceptual primitives) that were deemed necessary and sufficient as a language of thought to represent meaning unambiguously. Any implicit information in the text (information that can be inferred by the reader) was to be made explicit in the conceptual dependency representation.

This goal gave rise to the development of a large number of data structures and inference mechanisms (often without a well-defined semantics). Data structures included *causal* chains (a chain of states enabling or motivating actions which in turn result in, or initiate, other states), *scripts* and *scenarios* (prepackaged sequences of causal chains), and MOPS (Memory Organization Packages) (Schank & Abelson, 1977; Schank, 1982). These data structures enabled directed and efficient inference mechanisms, based on following up causal connections and associations between representations at the same and at different levels of abstraction.

One problem is that these models tend to focus on the data structure, and are vague on the inference part.

Two sources of knowledge are indispensable for developing useful symbolic natural language understanding systems: (1) knowledge about the intentions, plans and goals of different agents in narratives or dialogue, and (2) knowledge about preceding discourse (discourse representation). In work by Allen and Perrault (1980) and others, AI planning formalisms are combined with speech act theory to model the recognition of intention, an approach which gave rise to research on speech act planning, topic structure modeling, and user modeling. This AI work has influenced psycholinguistic models of discourse comprehension and discourse production

### Self Assessment

State whether the following statements are true or false:

1. A semantic network is regarded as a graphical notation for logical formulas.
2. In a semantic network, each link between four nodes represents a separate proposition.

## 8.2 Production Systems

Production systems are rule-based systems developed during the seventies as models for human problem solving. They are common in models for many areas of knowledge. In this kind of formalism, knowledge is expressed as rules taking the form of condition-action pairs: if X then do Y.

*Example:* In a model for language production, one of the rules for producing questions might be the following: if the intention is to query the truth of P, then produce a sentence about P where the finite verb of the main clause is moved up front.

Notes

Rules of this type, often called production rules, can only produce actual behavior with the help of an interpreter, a mechanism which applies the rules to reach a given goal. In addition to the rule-base (which acts as a kind of long-term memory), a production rule system also has a short-term memory (working memory) which registers the current state of the computation, as well as current input and output states. The control structure of a production system interpreter consists of a cyclical process, where each cycle consists of three phases:

1. ***Identification:*** This phase determines for which rules the condition sides are currently satisfied in working memory.
2. ***Selection:*** It will often happen that more than one rule's condition side will be satisfied. Since in general it is not desirable for all applicable rules to fire, one or more rules are selected on the basis of a particular conflict resolution strategy.
3. ***Execution:*** The action part of the chosen rule is executed. Although actions can take many forms, the most typical ones involve the addition to or removal from working memory of certain facts.

This interpreter's mode of operation is called forward chaining or data-driven: rules are identified when states in working memory match their conditions; the execution of the rules may in their turn activate other rules, until a goal is achieved. But it is also possible to run an interpreter in a backward chaining or goal-driven mode: in that case, rules are identified when their actions match the current goals; their execution may add elements of their conditions as new goals when they are not present in working memory, and so on, until rules are found whose conditions match the current states in working memory. It is evident that both modes represent different kinds of search. Rule-based architectures have been further developed toward more sophisticated cognitive architectures, for example, ACT* (Anderson, 1983) and SOAR.

*Notes* The ACT* system has a semantic network (see above) as part of its long term memory. Production systems have been used in a few psycholinguistic models, but no models based.

Anderson, Kline and Lewis (1977) describe a production system model of language processing. In PROZIN (Kolk, 1987), agrammatism effects are simulated by manipulating the processing speed of the production system interpreter and the decay rate of facts in working memory. Lewis (1993) describes a computer model of human sentence comprehension implemented in SOAR.

### Self Assessment

State whether the following statements are true or false:

3. Production systems are rule-based systems developed during the seventies as models for human problem solving.
4. This interpreter's mode of operation is called upward chaining.

## 8.3 Logic

Logic has often been used as a formal foundation for knowledge representation in AI. The formal properties of logic formalisms are relatively well understood and make them ideally suited as a language to which other formalisms can be translated in order to evaluate and

Notes

compare them. Data organization in predicate logic consists of a set of unambiguous constants (representing entities in the domain), a set of unambiguous predicates (representing relations between entities in the domain), a set of functions (mapping between sets), variables, quantifiers, and logical connectives. Inference in predicate logic is achieved by applying deductive inference rules, e.g. by means of resolution. For an introduction to the logical approach to knowledge representation.

A practical computer language based on a limited version of predicate logic is PROLOG. Below is a small program that expresses the fact that Socrates and Plato are human, and the rule that if *x* is human, then *x* is mortal:

```
human(socrates).
human(plato).
mortal(X):- human(X).
```

The *interpreter* of PROLOG uses these facts and rules to derive other facts. For example, the following dialog is possible, where we ask whether Socrates and Descartes are mortal, and who are all mortal beings the system knows. The system infers, for instance, that Socrates and Plato are mortal. Notice that PROLOG gives a negative answer for everything that does not occur in the knowledge base.

```
|?-mortal(socrates).
yes
|?-mortal(descartes).
no
|?-mortal(X).
X=socrates;
X=plato;
No
```

Predicate logic has some severe limitations as a tool for representing linguistic knowledge which is incomplete, inconsistent, dynamically changing, or relating to time, action and beliefs.

*Caution* For all these problems, special purpose logics are being designed. An example is default logic, which handles exceptional information without having to modify existing general knowledge.

### Self Assessment

State whether the following statements are true or false:

5. The interpreter of PROLOG uses these facts and rules to derive other facts.

6. Logic has been used as a formal foundation for knowledge representation in AI.

## 8.4 Grammar Formalisms

Grammar formalisms constitute a special type of formalism for natural language processing, even though they are not unrelated to the knowledge representation paradigms and formalisms discussed earlier. They often use a different terminology, due to the different background of the developers, which is linguistics, logic, and theoretical computer science rather than AI, and use special notations for linguistic strings and structures. Most grammar formalisms were developed as part of the efforts to build systems for natural language understanding, which up to now received more attention in AI than natural language generation.

Notes

### 8.4.1 Phrase Structure Grammars and Automata

The representation of grammatical knowledge as phrase structure rules is common for syntactic parsing in sentence comprehension and to some extent, the recognition of complex words. The use of these rules is somewhat similar to production rules, but they operate on strings of linguistic items. Phrase structure rules basically specify how an initial symbol can be recursively expanded into a sequence of other symbols.

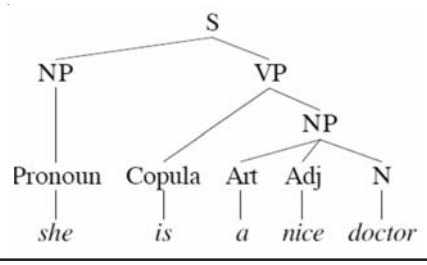
*Example:* The first rule in the rule set below specifies that a sentence (S) can be expanded into a noun phrase (NP) followed by a verb phrase (VP), or, inversely, that a noun phrase and a verb phrase can be reduced to a sentence.

The selection mechanism chooses among various applicable rules. Often, a symbol can be expanded into different ways, for example in the following rule set describing how an NP can be rewritten as either an article followed by a noun, or an article followed by an adjective, followed by a noun.

```
S -> NP VP
NP -> PRONOUN
NP -> ART N
NP -> ART ADJ N
VP -> COPULA NP
PRONOUN -> she
COPULA -> is
ART -> the
ART -> a
ADJ -> nice
ADJ -> smart
N -> doctor
```

A deterministic system will choose only one rule, whereas a non-deterministic system may search through a space of possibilities, using e.g. a parallel or backtracking search. When a rule is chosen, the left hand side of the rule is replaced with the right hand side. Successive expansions develop the structure until a solution is reached in the form of a sequence of words. The expansion history of a particular case can be represented as a syntactic tree structure. Clearly, different grammars give rise to different tree structures.

**Figure 8.2: Syntactic Tree Structure**




Phrase structure rules may operate in both directions, top-down, where the left hand sides of rules are rewritten as their right hand sides, or bottom-up, where the right hand sides are rewritten as the left hand sides. Depending on the form the left-hand side and the right-hand side of the phrase structure rules can take, different types of grammars can be formally defined: regular, context-free, context-sensitive, or unrestricted. Much research in CL is based on context-free grammars.

## Self Assessment

Notes

State whether the following statements are true or false:

7. Phrase structure rules specify how an initial symbol can be recursively expanded into a sequence of other symbols.

8. A symbol can be expanded in a one way.

## 8.5 Frames and Frame Systems

Many of the ideas about frame systems were first introduced in a unit by Marvin A. Minsky, entitled "Framework for Representing Knowledge," which appeared in P. H. Winston (Ed.), The Psychology of Computer Vision. NY: McGraw-Hill, 1975. Pp. 211-277. In this unit, Minsky introduces and argues for the idea of representing common sense knowledge in a data-structure that he calls a frame. His motivation for introducing these ideas is indicated immediately at the start of this unit where he states:

*"It seems to me that the ingredients of most theories both in artificial intelligence and in psychology have been on the whole too minute, local and unstructured to account either practically or phenomenologically for the effectiveness of common sense thought. The "chunks" of reasoning, language, memory and "perception" ought to be larger and more structured, and their factual and procedural contents must be more intimately connected in order to explain the apparent power and speed of mental activities."*

A frame is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information. Some to this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expressions are not confirmed.

We can think of a frame as a network of nodes and relations. The "top levels" of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals- "slots" that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (The assignments themselves are usually smaller "subframes.") Simple conditions are specified by markers that might require a terminal assignment to be a person an object of sufficient value, or a pointer to a sub-frame of a certain type. More complex conditions can specify relations among things assigned to several terminals.

Collections of related frames are linked together into frame systems. The effects of important actions are mirrored by transformations between the frames of a system. These are used to make certain kinds of calculation economical, to represent changes of emphasis and attention, and to account for the effectiveness of imagery."

As these ideas were realized in various "frame systems," a rough agreement emerged concerning exactly what constituted a frame. Below is listed these components.

A frame is a data structure that typically consists of:

- Frame Name
- Slot-filler (relations target)
  - default values
  - constraints on values within the slots of a frame

Notes

- pointers (links) to other Frames
  - ako (a-kind-of)
  - instance (is-a)
- Instantiation Procedure
  - Inheritance Procedure
  - Default Inference Procedure
  - Consistency Checking Procedure
  - Inconsistency Resolution Procedure

*Notes* That knowledge is organized around the objects. The knowledge is accessed when an instance of a particular object is created or **instantiated**. Instantiation can be an extremely rich and complicated procedure as can be noted by reviewing the various procedures that are listed above as part of the instantiation procedure. Additionally, instantiation is typically specialized with respect to classes of objects.

### An Example

A brief example of an instantiation process is shown below. This example is a trace recorded long ago (in the late 70s or early 80s) taken from a frame system, AIMDS, developed for use in research on plan recognition. For those unfamiliar with programming, most programming languages give one some facility create a record of chosen aspects of a procedure. In this example, certain component functions of the process were chosen and the result returned by these component function is printed as the function is exited.

In this particular example, the instantiation process is specialized for frames that represent actions which modify a world. The example below lists the definition provided the system for the two frames used in this example. The first is the frame referred to as ACT. The second is the frame referred to as WALK. WALK is a kind of ACT. When a WALK is instantiated an ACT representation is also created for that instance of ACT. In this example, we will focus on the frame, WALK. The strings TDN: and QSCC: that occur in these tables are simply the name of functions that are invoked to create these data structures when this knowledge is entered.

Beneath the name of the frame is listed the set of slots that are associated with the frame. For WALK, the first slot is (AGENT PERSON) the next (LOC LOCATION) and so on. The terms shown in bold (PERSON, LOCATION, and so one) are other frames which themselves would typically be further defined. The term proceeding these is the name of the relation or slot AGENT and LOC in these first two elements. Thus, this is a way of specifying that WALK is a kind of ACTION which involves an agent who is a PERSON, and so on.

**An Example Pair of Frames: ACT and WALK**

```
                (* STRUCTURAL DESCRIPTION OF THE ACT FRAME *)
(TDN:  [(ACT D)
       ((PRESUPPOSITIONS L) PROPOSITION)
       ((RESULTS L) PROPOSITION)
       ((OUTCOMES L) PROPOSITION)
       ((PRECONDITIONS L) PROPOSITION)
       ((ENABLE L) ACT ENABLEDBY)
       ((MOTIVATES L) ACT MOTIVATESOF) ])
```

Notes

```
            (* STRUCTURAL DESCRIPTION OF THE WALK FRAME *)
   (TDN:  [(WALK P A)
          ((AGENT I FN) PERSON)
          (LOC LOCATION)
          ((TOLOC I FN) LOCATION)
          (FROMLOC LOCATION)
          ((OPPORTUNITY L) (PROPOSITION ((X AGENT) LOC (X FROMLOC))))
          (PGOAL (PROPOSITION ((X AGENT) LOC (X TOLOC))))
          ((PRESUPPOSITIONS L) PROPOSITION)
          ((RESULTS L) PROPOSITION) ])

            (* CONSISTENCY CONDITIONS FOR THE WALK FRAME *)
   (QSCC: [((LOCATION Z) | (NOT [X TOLOC Z]) (X (AGENT LOC) Z)) WALK
FROMLOC])
   (QSCC: [((LOCATION L) | (X TOLOC L) (NOT [X FROMLOC L])) WALK
TOLOC])
   (QSCC: [((LOCATION L) | (X FROMLOC L)) WALK LOC])
   (QSCC: [((PERSON P) | (X (LOC LOCOF) P)) WALK AGENT])
```

The final segment of the example above lists what are referred to as consistency conditions; in this case for the WALK FRAME. These are expressions that are given a logical interpretation and constrain the way in which a WALK instance can be instantiated. Note that at the end of each of these lines the name WALK and a relation or slot are shown in bold. For example, the first is WALK FROMLOC. This constraint expression is attached to this slot. For example, the first constraint states that the LOCATION Z that is placed in the fromloc slot is a location such that Z is not the same as the location placed in the toloc slot; and it is the LOCATION that is in the slot loc that is associated with the PERSON P in the agent slot of WALK. More simply put; when you walk you must change location and the starting location of the Walk is the same as where you are when you begin this action.

*Caution* Use a better programming paradigm for knowledge representation.

*Task* List out the three level of knowledge representation.

## Self Assessment

State whether the following statements are true or false:

9. The expansion history of a particular case can not be represented as a syntactic tree structure.

10. A frame is a data-structure for representing a stereotyped situation.

# 8.6 Conceptual Dependency Theory and Object-oriented Programming

## 8.6.1 Conceptual Dependency Theory

Conceptual dependency theory is a model of natural language understanding used in artificial intelligence systems. Roger Schank at Stanford University introduced the model in 1969, in the early days of artificial intelligence. This model was extensively used by Schank's students at Yale University such as Robert Wilensky, Wendy Lehnert, and Janet Kolodner.

**Notes**

Schank developed the model to represent knowledge for natural language input into computers. Partly influenced by the work of Sydney Lamb, his goal was to make the meaning independent of the words used in the input, i.e. two sentences identical in meaning, would have a single representation. The system was also intended to draw logical inferences.

The model uses the following basic representational tokens:

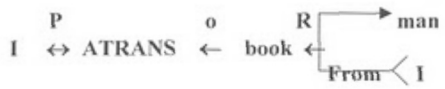
- real world objects, each with some attributes.
- real world actions, each with attributes
- times
- locations

A set of conceptual transitions then act on this representation, e.g. an ATRANS is used to represent a transfer such as "give" or "take" while a PTRANS is used to act on locations such as "move" or "go". An MTRANS represents mental acts such as "tell", etc.

A sentence such as "john gave a book to Mary" is then represented as the action of an ATRANS on two real world objects John and Mary.

This representation is used in natural language processing in order to represent them earning of the sentences in such a way that inference we can be made from the sentences. It is independent of the language in which the sentences were originally stated. CD representations of a sentence is built out of primitives, which are not words belonging to the language but are conceptual, these primitives are combined to form the meanings of the words. As an example, consider the event represented by the sentence.

**I gave the man a book.**
**In CD from the above sentence can be represented with primitives as**




In the above representation, the symbols have the following meaning:

- Arrows indicate direction of dependency
- Double arrow indicates two may link between actor and the action
- P indicates past tense

ATRANS is one of the primitive acts used by the theory. It indicates transfer of possession

- 0 indicates the object case relation
- R indicates the recipient case relation

Conceptual dependency provides a structure in which knowledge can be represented and also a set of building blocks from which representations can be built. A typical set of primitive actions are:

ATRANS – Transfer of an abstract relationship (E.g.: give)

PTRANS – Transfer of the physical location of an object (E.g.: go)

PROPEL – Application of physical force to an object (E.g.: push)

MOVE – Movement of a body part by its owner (e.g.: kick)

Notes

GRASP – Grasping of an object by an actor (E.g.: throw)

INGEST – Ingesting of an object by an animal (E.g.: eat)

EXPEL – Expulsion of something from the body of an animal (cry)

MTRANS – Transfer of mental information (E.g.: tell)

MBUILD – Building new information out of old (E.g.: decide)

SPEAK – Production of sounds (E.g.: say)

ATTEND – Focusing of sense organ toward a stimulus (E.g.: listen)

A second set of building block is the set of allowable dependencies among the conceptualization describe in a sentence.

### 8.6.2 Object-oriented Programming

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The programming challenge was seen as how to write the logic, not how to define the data. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.

*Example:* Of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars).

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called (no surprise here) an "object" or, in some environments, an "instance of a class." The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. We communicate with objects – and they communicate with each other – with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).

Notes

- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

In programming languages, an object is the composition of nouns (like data such as numbers, strings, or variables) and verbs.

### Object-oriented Programming takes a Radically Different Approach

An object-oriented program may be viewed as a collection of interacting objects, as opposed to the conventional model, in which a program is seen as a list of tasks (subroutines) to perform. In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent "machine" with a distinct role or responsibility. Actions (or "methods") on these objects are closely associated with the object. For example, OOP data structures tend to "carry their own operators around with them" (or at least "inherit" them from a similar object or class)—except when they must be serialized.

Simple, non-OOP programs may be one "long" list of commands. More complex programs often group smaller sections of these statements into functions or subroutines—each of which might perform a particular task. With designs of this sort, it is common for some of the program's data to be 'global', i.e., accessible from any part of the program.

*Did u know?* As programs grow in size, allowing any function to modify any piece of data means that bugs can have wide-reaching effects.

In contrast, the object-oriented approach encourages the programmer to place data where it is not directly accessible by the rest of the program. Instead, the data is accessed by calling specially written functions, commonly called methods, which are bundled in with the data. These act as the intermediaries for retrieving or modifying the data they control. The programming construct that combines data with a set of methods for accessing and managing those data is called an object. The practice of using subroutines to examine or modify certain kinds of data was also used in non-OOP modular programming, well before the widespread use of object-oriented programming.

An object-oriented program usually contains different types of objects, each corresponding to a particular kind of complex data to manage, or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer. A program might contain multiple copies of each type of object, one for each of the real-world objects the program deals with. For instance, there could be one bank account object for each real-world account at a particular bank. Each copy of the bank account object would be alike in the methods it offers for manipulating or reading its data, but the data inside each object would differ reflecting the different history of each account.

Objects can be thought of as encapsulating their data within a set of functions designed to ensure that the data are used appropriately, and to assist in that use. The object's methods typically include checks and safeguards specific to the data types the object contains. An object can also offer simple-to-use, standardized methods for performing particular operations on its data, while concealing the specifics of how those tasks are accomplished. In this way, alterations can be made to the internal structure or methods of an object without requiring that the rest of the program be modified. This approach can also be used to offer standardized methods across different types of objects. As an example, several different types of objects might offer print methods. Each type of object might implement that print method in a different way, reflecting the different kinds of data each contains, but all the different print methods might be called in the same standardized manner from elsewhere in the program. These features become especially

Notes

useful when more than one programmer is contributing code to a project or when the goal is to reuse code between projects.

### Advantages of Object-oriented Programming

Object-oriented programming is a style of programming that focuses on using objects to design and build applications. Think of an object as a model of the concepts, processes, or things in the real world that are meaningful to your application. For example, in a project management application, you would have a status object, a cost object, and a client object among others. These objects would work together (and with many other objects) to provide the functionality that you want your project management application to have. Object-oriented programming is used to develop many applications—simple and complex applications, business applications and games, mobile and desktop applications. Developers choose to program in the object-oriented paradigm because the proper use of objects makes it easier to build, maintain, and upgrade an application. Also, developing an application with objects that have been tested increases the reliability of the application.

### Self Assessment

State whether the following statements are true or false:

11. Conceptual dependency theory is a model of natural language understanding used in artificial intelligence systems.
12. Arrows indicate direction of independency.

## 8.7 Object and Classes

In programming terms, an object is a self-contained component that contains properties and methods needed to make a certain type of data useful. An object's properties are what it knows and its methods are what it can do. The project management application mentioned above had a status object, a cost object, and a client object, among others. One property of the status object would be the current status of the project. The status object would have a method that could update that status. The client object's properties would include all of the important details about the client and its methods would be able to change them. The cost object would have methods necessary to calculate the project's cost based on hours worked, hourly rate, materials cost, and fees.

### Self Assessment

State whether the following statements are true or false:

13. An object's properties are what it knows and its methods are what it can not do.
14. One property of the status object would be the current status of the project.

## 8.8 Messages and Methods

In addition to providing the functionality of the application, methods ensure that an object's data is used appropriately by running checks for the specific type of data being used. They also allow for the actual implementation of tasks to be hidden and for particular operations to be standardized across different types of objects. You will learn more about these important capabilities in Object-oriented concepts: Encapsulation. Objects are the fundamental building

Notes

blocks of applications from an object-oriented perspective. Each different type of object comes from a specific class of that type. A class is a blueprint or template or set of instructions to build a specific type of object. Every object is built from a class. Each class should be designed and programmed to accomplish one, and only one, thing. (You'll learn more about the Single Responsibility Principle in Object-oriented programming concepts: Writing classes.) Because each class is designed to have only a single responsibility, many classes are used to build an entire application. An instance is a specific object built from a specific class. It is assigned to a reference variable that is used to access all of the instance's properties and methods. When you make a new instance the process is called instantiation and is typically done using the new keyword. Think about classes, instances, and instantiation like baking a cake.

A class is like a recipe for chocolate cake. The recipe itself is not a cake. You can't eat the recipe (or at least wouldn't want to). If you correctly do what the recipe tells you to do (instantiate it) then you have an edible cake. That edible cake is an instance of the chocolate cake class. We can bake as many cakes as you would like using the same chocolate cake recipe. Likewise, you can instantiate as many instances of a class as you would like. Pretend you are baking three cakes for three friends who all have the same birthday but are different ages. You will need some way to keep track of which cake is for which friend so you can put on the correct number of candles. A simple solution is to write each friend's name on the cake. Reference variables work in a similar fashion. A reference variable provides a unique name for each instance of a class. In order to work with a particular instance, you use the reference variable it is assigned to.

*Caution* Learn the basic concepts of OOPs before implementing it.

*Task* Make a Java program for small calculator using objects and classes.

## Self Assessment

State whether the following statements are true or false:

15. Every object is built from a class.
16. Objects are the fundamental building blocks of applications from an object-oriented perspective.

## 8.9 Summary

- The Hopfield is an auto-associative network often used (or derivatives of it) in monochrome image recognition.
- A script is a data structure used to represent a sequence of events. Scripts are used for interpreting stories. Popular examples have been script driven systems that can interpret and extract facts from Newspaper Stories.
- The object-oriented approach to programming is based on an intuitive correspondence between a software simulation of a physical system and the physical system itself.
- Frame is a type of schema used in many AI applications including vision and natural language processing. Frames provide a convenient structure for representing objects that are typical to stereotypical situations.

Notes

- Conceptual Dependency originally developed to represent knowledge acquired from natural language Input.

## 8.10 Keywords

***Conceptual Dependency:*** It is a model of natural language understanding used in artificial intelligence systems.

***Expert System:*** An expert system is a computer system that emulates the decision-making ability of a human expert.

***Frame:*** A frame is an artificial intelligence data structure used to divide knowledge into substructures by representing stereotyped situations.

***Knowledge Representation (KR):*** It is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inference from those knowledge elements, creating new elements of knowledge.

***Scripts:*** Scripts are a method of representing procedural knowledge. They are very much like frames, except the values that fill the slots must be ordered.

## 8.11 Review Questions

1. What are the needs of associative networks?
2. Explain the frame structures and scripts.
3. What is the conceptual dependency (CD)?
4. Define object-oriented representation.
5. Write the advantages of object-oriented programming.
6. What are the purposes of object classes?
7. Explain the messages and methods in artificial intelligence.
8. What are the uses of frames and scripts?

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | True | 4. | False |
| 5. | True | 6. | True |
| 7. | True | 8. | False |
| 9. | False | 10. | True |
| 11. | True | 12. | False |
| 13. | False | 14. | True |
| 15. | True | 16. | True |

Notes

## 8.12 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

html/disciascio02a.html ww.businessdictionary.com/definition/knowledge-structure.html

http://deepblue.lib.umich.edu/bitstream/handle/2027.42/43836/11229_2004_Article_BF00413665.pdf?sequence=1

http://krchowdhary.com/ai/Conceptual%20dependencies.pdf

http://merode.econ.kuleuven.ac.be/publications%5Cmerobk01.pdf

http://whttp://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/disciascio02a

http://www.cs.cf.ac.uk/Dave/AI2/node69.html

http://www.cs.wustl.edu/~schmidt/PDF/ood-overview4.pdf

http://www.slideshare.net/adrijovin/an-overview-of-object-oriented-systems-development

# Unit 9: Search and Control Strategies

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Identify Preliminary Concepts
- Discuss the examples of Search Problems
- Explain the concept of Uninformed or Blind Search
- Describe Informed Search
- Discuss the Problem Reduction (AND/OR-Tree Search)

Notes

## Introduction

In the field of artificial intelligence, "planning" is defined as designing the behavior of some entity that acts an individual, a group, or an organization. The output is some kind of blueprint for behavior, which we call a plan. There are wide variety of planning problems, differentiated by the types of their inputs and outputs. Typically, planning problems get more and more difficult as more flexible inputs are allowed and fewer constraints on the output are required. As this flexibility increases, the space of possibilities needing to be explored by the planning algorithm grows extremely quickly. This problem is the essence of the planning problem, and it arises regardless of which specific planning methodology is used (nonlinear planning, deductive planning, hierarchical planning, etc.) – in all of these controlling an exponentially growing search space is a central problem. In all planning formalizations, it is critical that some sort of knowledge (heuristic or otherwise) is used to make reasonable decisions at any of the many choice points which arise in planning. Such choice points can concern:

- ordering of subgoals
- selection of operators/control structures
- resolution of conflicts/threats by different techniques
- choosing between differing commitment strategies
- selecting/choosing the right control regime

In all of these cases, picking the right control knowledge can result in an algorithm that is able to identify and prune many dead-end branches of the search space before the algorithm explores them, ideally while preserving the soundness and completeness of the planner. However, designing search control approaches is difficult and it is often impossible to ensure various qualitative and quantitative properties of the "controlled algorithms".

## 9.1 Preliminary Concepts

Deductive planning systems rely on an expressive logical representation formalism, a proper formal semantics, and a calculus the rules of which are used to implement the planner. Plans are generated by constructive proof of so-called plan specification formulae. In the simplest case, specification formulae describe the initial state and the goal state and demand for the existence of a plan that transforms the one into the other. Starting from a plan specification, a proof (tree) is developed by applying logical inference rules in a backward-chaining manner, and by instantiating the plan variable accordingly. We have introduced the modal temporal planning logic TPL, as an example, and have demonstrated how deductive planning works in this context.

TPL is an expressive formalism that allows for the distinction of rigid and flexible symbols and provides a programming language for plans, including control structures like conditionals and loops. The search problems and solutions we have addressed are concerned with the guidance of the theorem proving = planning process, the selection of appropriate basic actions, and the selection of subgoals.

*Notes* Since a carefully designed domain model is an essential prerequisite for acting safely and efficiently in this context, the planning environment we have introduced provides deductive support in setting up provably consistent domain models.

The general context for the research is the design of integrated planning and scheduling architectures for constraint-based activity management. A number of aspects useful to improve

Notes

the applicability of such architectures have been addressed. One aspect is connected with the involvement of users in actively exploiting such systems. Relevant problems are: the definition of a domain representation language powerful enough to represent the physical constraints of a domain, and endowed with clear semantics as a basis for automated verification tools; the investigation of various aspects of the interaction with the user like the "cognitive intelligibility" of the plan representation and the planning process.

A second aspect consists in the study of specialized classes of constraint propagation techniques: interesting results on the efficiency and flexibility of manipulating quantitative temporal networks have been obtained through the synthesis of dynamic algorithms for constraint posting and retraction; also the problem of mixed resource and time constraints representation has been studied and some techniques proposed for the synthesis of implicit temporal constraints from the analysis of resource representation. A further aspect consists in the integration of the incremental constructive way of building a solution with local search techniques: in particular a taboo search algorithm has been proposed that take advantage of the given temporal representation to solve planning problems requiring multiple resources.

Early AI researchers developed algorithms that imitated the step-by-step reasoning that humans use when they solve puzzles or make logical deductions. By the late 1980s and 1990s, AI research had also developed highly successful methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.

*Did u know?* For difficult problems, most of these algorithms can require enormous computational resources – most experience a "combinatorial explosion": the amount of memory or computer time required becomes astronomical when the problem goes beyond a certain size. The search for more efficient problem-solving algorithms is a high priority for AI research.

Human beings solve most of their problems using fast, intuitive judgments rather than the conscious, step-by-step deduction that early AI research was able to model. AI has made some progress at imitating this kind of "sub-symbolic" problem solving: embodied agent approaches emphasize the importance of sensor motor skills to higher reasoning; neural net research attempts to simulate the structures inside the brain that give rise to this skill; statistical approaches to AI mimic the probabilistic nature of the human ability to guess.

### Self Assessment

State whether the following statements are true or false:

1. Plans are generated by constructive proof of so-called plan specification formulae.
2. By the late 1940s and 1960s, AI research had also developed highly successful methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.
3. AI has made some progress at imitating kind of "sub-symbolic" problem solving.
4. TPL is an expressive formalism that allows for the distinction of rigid and flexible symbols.

## 9.2 Examples of Search Problems

### 9.2.1 Water Container Problem

Search plays a major role in solving many Artificial Intelligence (AI) problems. Search is a universal problem-solving mechanism in AI. In many problems, sequence of steps required to

Notes

solve is not known in advance but must be determined by systematic trial-and-error exploration of alternatives. The problems that are addressed by AI search algorithms fall into three general classes: single-agent pathfinding problems, two players games, and constraint-satisfaction problems

*Example:* There is a 4l container and 3l container; neither has any measuring markers on it. There is a pump that can be used to fill the containers with water. Problem to solve is to get exactly two liters of water in the 4l container.

***Solution:*** From initial state to goal state through appropriate sequence of moves or actions such as filling and emptying the containers are described as below:

Content of the two containers at any given time is a problem state:

Let x - content of the 4l container
y - content of the 3l container

Then (x,y) - problem state represented by an ordered pair. The set of all ordered pairs is the space of problem states or the state-space of the problem.

State-space: { (x,y) | x = 0,1,2,3,4 y=0,1,2,3 }

Data structure to represent the state-space can be:

- Vectors
- Sets
- Arrays
- Lists

### 9.2.2 Problem Characteristics

The problem characteristics can be illustrated as:

Suppose: initial state (0,0)
and Goal state (2, y) where y = any possible number.

Moves transform from one state into another state. Operators determine the moves. Operators for the problem state-space:

1. Fill the 4l container
2. Fill the 3l container
3. Empty the 3l container
4. Empty the 3l container
5. Pour water from 3l container into 4l container until 4l container is full
6. Pour water from 4l container into the 3l container until the 3l container is full
7. Pour all the water from 3l container into the 4l container
8. Pour all the water from 4l container into the 3l container

Preconditions need to be satisfied before an operator can be applied.

e.g.: # 1 can be applied if there is less than 4l water in the container.

Notes

IF there is less than 4l in the 4l container THEN fill the 4l container.

Adding preconditions to operators => generation of production rules.

Forwarded form of rule # 1:

IF (x,y | x?4) THEN (4,y)

The forwarded set of production rules:

R1 IF (x,y | x?4) THEN (4,y)
R2 IF (x,y | y?3) THEN (x,3)
R3 IF (x,y | x>0) THEN (0,y)
R4 IF (x,y | y>0) THEN (x,0)
R5 IF (x,y | x+y>=4 ? y>0 ? x?4) THEN (4,y-(4-x))
R6 IF (x,y | x+y>=3 ? x>0 ? y?3) THEN (x-(3-y),3)
R7 IF (x,y | x+y?=4 ? y>0) THEN (x+y,0)
R8 IF (x,y | x+y?=3 ? x>0) THEN (0,x+y)

In certain states, more than one rule can be applied.

e.g.: (4,0) satisfies the preconditions of R2, R3 ? R6

***Control or Search Strategy:*** Selecting rules; keeping track of those sequences of rules that have already been tried and the states produced by them. Goal state provides a basis for the termination of the problem solving task.

(i) ***Pattern Matching Stage:*** Execution of a rule requires a match.

| | | |
|---|---|---|
| preconditions of a rule when match is found several rules may be applicable | match <=====> => rule is applicable | content of the working memory. |

(ii) ***Conflict Resolution (Selection Strategy) Stage:*** Selecting one rule to execute;

(iii) ***Action Stage:*** Applying the action part of the rule => changing the content of the workspace =>new patterns, new matches => new set of rules eligible for execution

Recognize-act control cycle

### *Production System*

1. Content of workspace => candidate rules for execution (search strategy)
2. Each rule surveys the entire workspace
3. A rule activates itself when its condition matches the content of the workspace

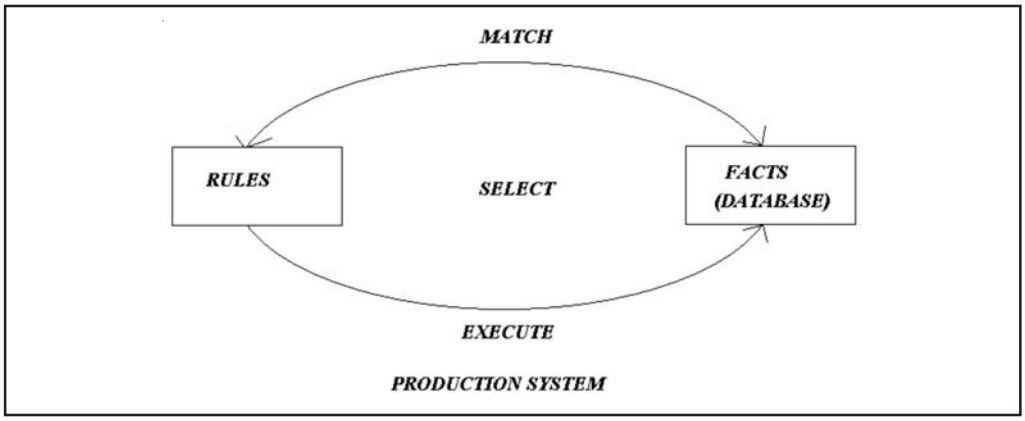
***Adv:*** Adding, deleting, revising rules without disturbing the rest of the system.

### *Water Container Problem*

| **Production Rules** | **Facts in Working Memory** |
|---|---|
| IF (x<4) THEN FILL x | (4,y) |
| IF (y<3) THEN FILL y | (x,3) |
| IF (x>0) THEN EMPTY x | (0,y) |
| IF (y>0) THEN EMPTY y | (x,0) |
| IF (x+y>=4 AND y>0) THEN FILL x FROM y | (4,y-(4-x)) |
| IF (x+y>=3 AND x>0) THEN FILL y FROM x | (x-(3-y),3) |

Notes

IF (x+y<=4 AND y>0) THEN FILL x FROM y (x+y,0)
IF (x+y<=3 AND x>0) THEN FILL y FROM x (0,x+y)




*Production System*

no information is available to judge that one rule is better than the rest of the applicable rules

=>systematic search of the state-space is necessary for finding a solution.

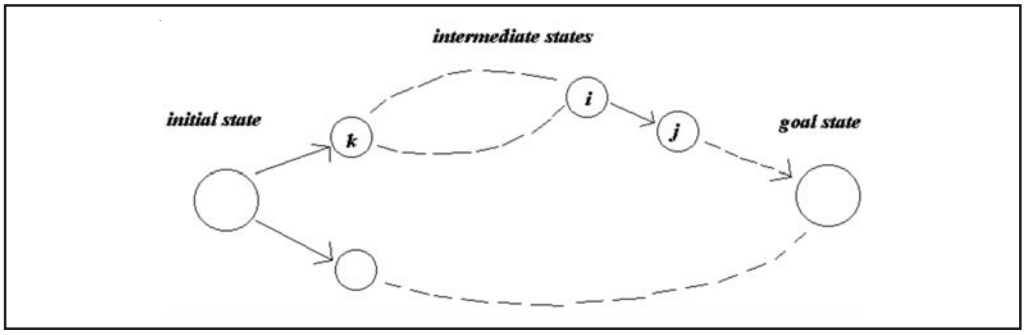
=>no information which action is better than the others

=>no information on the progress towards solution

blind search (exhaustive search) is being done.

When number of states is high it is not an efficient method.

*Standard Terminology*




State j is an immediate successor of state i.

State i is a successor of state k.

All states that can be reached from initial state are search space. Each intermediate state generates an exhaustive set of successors. Exhaustive search leading to combinational explosion. To prevent combinational explosion, information at each intermediate state to identify which immediate successor is most promising.

### 9.2.3 Means-End Analysis

The most basic definition is "A problem is any given situation that differs from a desired goal". This definition is very useful for discussing problem solving in terms of evolutionary adaptation,

Notes

as it allows to understand every aspect of (human or animal) life as a problem. This includes issues like finding food in harsh winters, remembering where you left your provisions, making decisions about which way to go, learning, repeating and varying all kinds of complex movements, and so on. Though all these problems were of crucial importance during the evolutionary process that created us the way we are, they are by no means solved exclusively by humans. We find a most amazing variety of different solutions for these problems in nature (just consider, e.g., by which means a bat hunts its prey, compared to a spider). For this, we will mainly focus on those problems that are not solved by animals or evolution, that is, all kinds of abstract problems (e.g. playing chess).

Furthermore, we will not consider those situations as problems that have an obvious solution: Imagine Knut decides to take a sip of coffee from the mug next to his right hand. He does not even have to think about how to do this. This is not because the situation itself is trivial (a robot capable of recognising the mug, deciding whether it is full, then grabbing it and moving it to Knut's mouth would be a highly complex machine) but because in the context of all possible situations it is so trivial that it no longer is a problem our consciousness needs to be bothered with. The problems we will discuss in the following all need some conscious effort, though some seem to be solved without us being able to say how exactly we got to the solution. Still we will find that often the strategies we use to solve these problems are applicable to more basic problems, too.

For many abstract problems it is possible to find an algorithmic solution. We call all those problems well-defined that can be properly formalised, which comes along with the following properties:

- The problem has a clearly defined given state. This might be the line-up of a chess game, a given formula you have to solve, or the set-up of the towers of Hanoi game.
- There is a finite set of operators, that is, of rules you may apply to the given state. For the chess game, e.g., these would be the rules that tell you which piece you may move to which position.
- Finally, the problem has a clear goal state: The equations is resolved to x, all discs are moved to the right stack, or the other player is in checkmate.

Not surprisingly, a problem that fulfils these requirements can be implemented algorithmically. Therefore many well-defined problems can be very effectively solved by computers, like playing chess.

Though many problems can be properly formalized (sometimes only if we accept an enormous complexity) there are still others where this is not the case. Good examples for this are all kinds of tasks that involve creativity, and, generally speaking, all problems for which it is not possible to clearly define a given state and a goal state: Formalizing a problem of the kind "Please paint a beautiful picture" may be impossible. Still this is a problem most people would be able to access in one way or the other, even if the result may be totally different from person to person. And while Knut might judge that picture X is gorgeous, you might completely disagree. Nevertheless ill-defined problems often involve sub-problems that can be totally well-defined. On the other hand, many everyday problems that seem to be completely well-defined involve- when examined in detail- a big deal of creativity and ambiguities. If we think of Knut's fairly ill-defined task of writing an essay, he will not be able to complete this task without first understanding the text he has to write about. This step is the first sub goal Knut has to solve. Interestingly, ill-defined problems often involve sub problems that are well-defined.

*Caution* Searching should be handled carefully.

Notes


*Task* Learn and revise various searching methods studied earlier in data structure.

## Self Assessment

State whether the following statements are true or false:

5. Preconditions need to be satisfied before an operator can be applied.
6. A problem is any given situation that does not differs from a desired goal.
7. Though all the problems were not of crucial importance.
8. For many abstract problems it is possible to find an algorithmic solution.

## 9.3 Uninformed or Blind Search

Blind search, also called uninformed search, works with no information about the search space, other than to distinguish the goal state from all the others. The following applets demonstrate four different blind search strategies, using a small binary tree whose nodes contain words. Just enter a word in the text input field (your word doesn't have to be in the tree) and click on the "Start Search" button in order to begin the search. The nodes will change color to red as they are visited by the search.

**Uninformed vs. Informed Search**

- ***Uninformed search strategies:***
  - Also known as "blind search," uninformed search strategies use no information about the likely "direction" of the goal node(s)
  - Uninformed search methods: Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional.
- ***Informed search strategies:***
  - Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
  - Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*.

### 9.3.1 Breadth-first Search

- Enqueue nodes on nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes

Notes

- For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree. If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

### 9.3.2 Depth-first Search

- Enqueue nodes on nodes in **LIFO** (last-in, first-out) order. That is, nodes used as a stack data structure to order nodes.
- **May not terminate** without a "depth bound," i.e., cutting off search below a fixed depth D ("depth-limited search")
- **Not complete** (with or without cycle detection, and with or without a cutoff depth)
- **Exponential time**, $O(b^d)$, but only **linear space**, O(bd)
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits a dead-end, can only back up one level at a time even if the "problem" occurs because of a bad operator choice near the top of the tree.

### Self Assessment

State whether the following statements are true or false:

9. Blind search works with information about the search space.
10. Uninformed search strategies also known as heuristic search.
11. Depth-first Search may terminate with a depth bound.
12. Breadth-first Search enqueue nodes on nodes in FIFO.

## 9.4 Informed Search

It is not difficult to see that uninformed search will pursue options that lead away from the goal as easily as it pursues options that lead to wards the goal. For any but the smallest problems this leads to searches that take unacceptable amounts of time and/or space. Informed search tries to reduce the amount of search that must be done by making intelligent choices for the nodes that are selected for expansion. This implies the existence of some way of evaluating the likelihood that a given node is on the solution path. In general, this is done using a **heuristic** function.

Unfortunately, uninformed search methods are very inefficient in most cases.

We now show how heuristic search strategies – ones that use problem specific knowledge – can find solutions more efficiently. In computer science, a **search algorithm** is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or may be elements of a search space defined by a mathematical formula or procedure, such as the roots of an equation with integer variables; or a combination of the two, such as the Hamiltonian circuits of a graph. Algorithms for searching virtual spaces are used in constraint satisfaction problem, where the goal is to find a set of value assignments to certain variables that will satisfy specific mathematical equations and inequations. They are also used when the goal is to find a variable assignment that will maximize or minimize a certain function of those variables.

Notes

*Notes* Algorithms for these problems include the basic brute-force search (also called "naïve" or "uninformed" search), and a variety of heuristics that try to exploit partial knowledge about structure of the space, such as linear relaxation, constraint generation, and constraint propagation.

This class also includes various tree search algorithms, that view the elements as vertices of a tree, and traverse that tree in some special order. Examples of the latter include the exhaustive methods such as depth-first search and breadth-first search, as well as various heuristic-based search tree pruning methods such as backtracking and branch and bound. Unlike general metaheuristics, which at best work only in a probabilistic sense, many of these tree-search methods are guaranteed to find the exact or optimal solution, if given enough time. Another important sub-class consists of algorithms for exploring the game tree of multiple-player games, such as chess or backgammon, whose nodes consist of all possible game situations that could result from the current situation. The goal in these problems is to find the move that provides the best chance of a win, taking into account all possible moves of the opponent(s). Similar problems occur when humans or machines have to make successive decisions whose outcomes are not entirely under one's control, such as in robot guidance or in marketing, financial or military strategy planning. This kind of problem – combinatorial search – has been extensively studied in the context of artificial intelligence. Examples of algorithms for this class are the minimax algorithm, alpha-beta pruning, and the A* algorithm.

### 9.4.1 Hill Climbing Methods

Hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

*Example:* Hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained. One of the problems with hill climbing is getting stuck at the local minima and this is what happens when you reach F. An improved version of hill climbing (which is actually used practically) is to restart the whole process by selecting a random node in the search tree and again continue towards finding an optimal solution. If once again you get stuck at some local minima you have to restart again with some other random node.

Generally, there is a limit on the no. of time you can re-do this process of finding the optimal solution. After you reach this limit, you select the least amongst all the local minima's you reached during the process. Though, it is still not complete but this one has better chances of finding the global optimal solution. Hill climbing has no guarantee against getting stuck in a local minima/maxima. However, only the purest form of hill climbing doesn't allow you to either backtrack. A simple riff on hill climbing that will avoid the local minima issue (at the expense of more time and memory) is a tabu search, where you remember previous bad results and purposefully avoid them.

Notes

### 9.4.2 Best-first Search

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. It is described as estimating the promise of node n by a "heuristic evaluation function which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called greedy best-first search. Efficient selection of the current best candidate for extension is typically implemented using a priority queue. The A* search algorithm is an example of best-first search, as is B*. Best-first algorithms are often used for path finding in combinatorial search.

#### Algorithms

```
OPEN = [initial state]
CLOSED = []
while OPEN is not empty
do
1.  Remove the best node from OPEN, call it n, add it to CLOSED.
2.  If n is the goal state, backtrace path to n (through recorded
    parents) and return path.
3.  Create n's successors.
4.  For each successor do:
    (a) If it is not in CLOSED: evaluate it, add it to OPEN, and
        record its parent.
    (b) Otherwise: change recorded parent if this new path is better
        than previous one.
done
```

### 9.4.3 A* Algorithm

The A* algorithm combines features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions. A* algorithm is a best-first search algorithm in which the cost associated with a node is f(n) = g(n) + h(n), where g(n) is the cost of the path from the initial state to node n and h(n) is the heuristic estimate or the cost or a path from node n to a goal. Thus, f(n) estimates the lowest total cost of any solution path going through node n. At each point, a node with lowest f value is chosen for expansion. Ties among nodes of equal f value should be broken in favor of nodes with lower h values. The algorithm terminates when a goal is chosen for expansion.

A* algorithm guides an optimal path to a goal if the heuristic function h(n) is admissible, meaning it never overestimates actual cost.

*Example:* Since airline distance never overestimates actual highway distance, and Manhattan distance never overestimates actual moves in the gliding tile.

For Puzzle, A* algorithm, using these evaluation functions, can find optimal solutions to these problems. In addition, A* makes the most efficient use of the given heuristic function in the following sense: among all shortest-path algorithms using the given heuristic function h(n). A* algorithm expands the fewest number of nodes. The main **drawback** of A* algorithm and indeed of any best-first search is its memory requirement. Since at least the entire open list must

Notes

be saved, A* algorithm is severely space-limited in practice, and is no more practical than best-first search algorithm on current machines. For example, while it can be run successfully on the eight puzzle, it exhausts available memory in a matter of minutes on the fifteen puzzle.

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way. It uses a knowledge-plus-heuristic cost function of node $x$ (usually denoted $f(x)$) to determine the order in which the search visits nodes in the tree. The cost function is a sum of two functions:

- the past path-cost function, which is the known distance from the starting node to the current node $x$ (usually denoted $g(x)$)
- a future path-cost function, which is an admissible "heuristic estimate" of the distance from $x$ to the goal (usually denoted $h(x)$).

The $h(x)$ part of the $f(x)$ function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, $h(x)$ might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.

If the heuristic $h$ satisfies the additional condition $h(x) \leq d(x, y) + h(y)$ for every edge $x, y$ of the graph (where $d$ denotes the length of that edge), then $h$ is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see *closed set* below)—and A* is equivalent to running Dijkstra's algorithm with the reduced cost $d(x, y) := d(x, y) - h(x) + h(y)$.

First search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals).

*Caution* Apply full algorithm of any searching technique.

*Task* Apply BFS in finding the shortest path to reach from one railway station to another railway station.

## Self Assessment

State whether the following statements are true or false:

13. Informed search tries to increase the amount of search that must be done by making intelligent choices for the nodes that are selected for expansion.
14. A search algorithm is an algorithm for finding an item with specified properties among a collection of items.
15. Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
16. A* uses a best-first search and finds a expensive path from a given initial node to one goal node.

## 9.5 Problem Reduction (AND/OR-Tree Search)

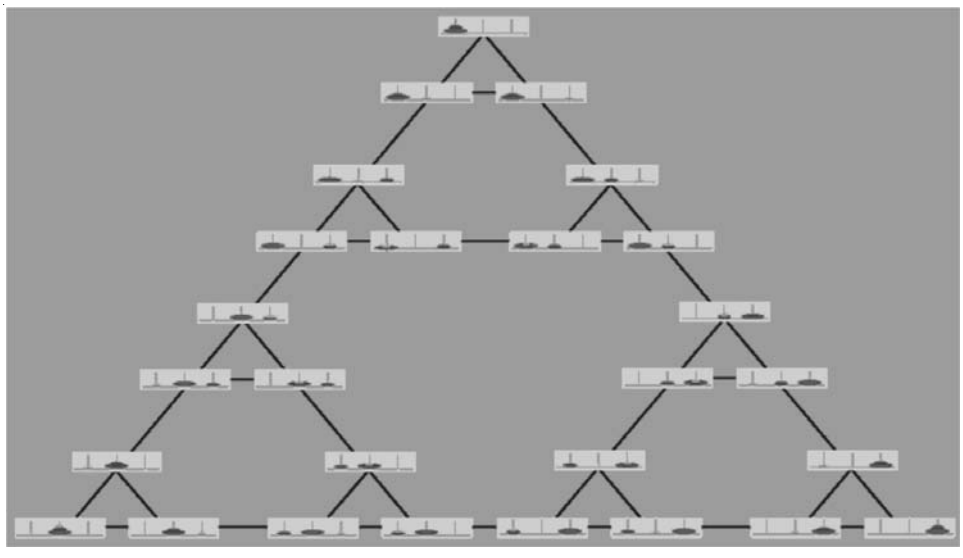
Notes

The general shape of all states was the same. They were distinguished by the values of their internal components. The basic idea here is that a problem is represented as one or more goals to be solved and a step in the problem solving process is either the immediate solution of one of these goals or its division into a set of (one hopes) simpler sub goals. The technique in question, construed in abstract terms, is basically just an exploitation of the idea that for any given goal there are a set of sub goals, such that satisfying the sub goals satisfies the goal. The process of decomposing a goal into its sub goals is referred to as goal-reduction. We will start by taking a very simple case where there is only one way of decomposing a goal into sub goals and therefore there is no search. To illustrate what this means we will look at the Tower of Hanoi problem.

### 9.5.1 Problem-Reduction and the Tower of Hanoi

Sometimes problems only seem hard to solve. A hard problem may be one that can be reduced to a number of simple problems and, when each of the simple problems is solved, then the hard problem has been solved. This is the basic intuition behind the method of problem reduction. We will have much more to say about this method of search when we cover problem solving and planning in the second half of the course. At this point, our goal is simply to introduce you to this type of search and illustrate the way in which it differs from state space search. The typical problem that is used to illustrate problem reduction search is the Tower of Hanoi problem because this problem has a very elegant solution using this method. The story that is typically quoted to describe the Tower of Hanoi problem describes the specific problem faced by the priests of Brahmah. Just in case you didn't decide to read this story, the gist of it is that 64 size ordered disks occupy one of 3 pegs and must be transferred to one of the other pegs. But, only one disk can be moved at a time; and a larger disk may never be placed on a smaller disk.

**Figure 9.1: State Space for the 3 Disk Tower of Hanoi Problem**

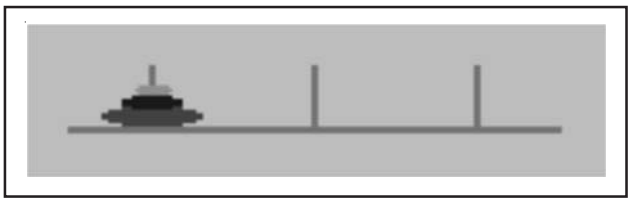


Rather than deal with the 64 disk problem faced by the priests, we will consider only three disks...the minimum required to make the problem mildly interesting and useful for our purpose here...namely to illustrate problem reduction search. In Figure shows the state space associated with a 3-disk Tower of Hanoi Problem. The problem involves moving from a state where the disks are stacked on one of the pegs and moving them so that they end up stacked on a different

Notes

peg. In this case, we will consider the state at the top of the figure the starting state. In this case, all three disks are on the left-most peg. And, we will consider the state at the bottom right to be the goal state. In this state, the three disks are now all stacked on the right-most peg.

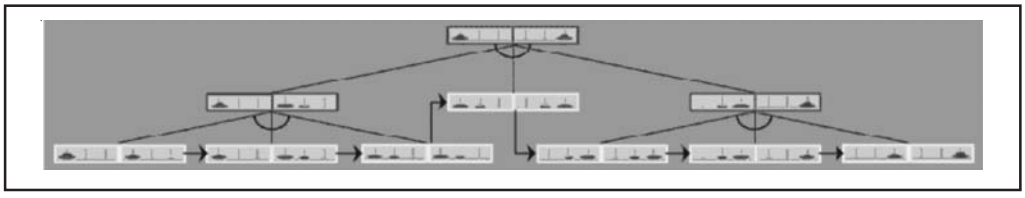
Recall that in state space search the generators correspond to moves in the state space. Thus, the two states below the top state in the triangle of states corresponds to the movement of the smallest disk either to the rightmost peg or to the middle peg. The shortest solution to this problem corresponds to the path down the right side of the state space. This solution is shown in the animation below.



In problem reduction search, the problem space consists of an AND/OR graph of (partial) state pairs. These pairs are referred to as (sub)problems. The first element of the pair is the starting state of the (sub)problem and the second element of the pair is the goal state (sub)problem. There are two types of generators: non-terminal rules and terminal rules. Non-terminal rules decompose a problem pair, <s0, g0> into an ANDed set of problem pairs {<si,gi>, <sj,gj>, ...>. The assumption is that the set of subproblems are in some sense simpler problems than the problem itself. The set is referred to as an ANDed set because the assumption is that the solution of all of the subproblems implies that the problem has been solved. Note all of the subproblems must be solved in order to solve the parent problem.

Any subproblem may itself be decomposed into subproblems. But, in order for this method to succeed, all subproblems must eventually terminate in primitive subproblems. A primitive subproblem is one which can not be decomposed (i.e., there is no non-terminal that is applicable to the subproblem) and its solution is simple or direct. The terminal rules serve as recognizers of primitive subproblems.

The symmetry of the state space shown above may have led you to suspect that the Tower of Hanoi problem can be elegantly solved using the method of problem decomposition. The AND tree that solves the 3 disk problem is shown below.



## 9.5.2 Implementing Arbitrary AND/OR-Trees

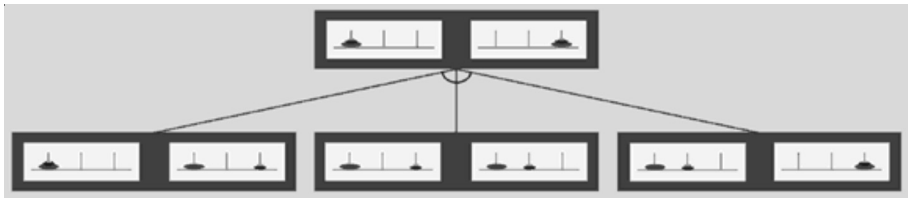
Let us number the state space solution shown in the state space above 1 through 8 so that we can refer to the states by number. 1 corresponds to the topmost or starting state and 8 to the right corner or goal state. These two states are the first and second element in the problem shown as the root node in the AND tree above. The red arc is used to indicate that the node is an AND node.

The problem is decomposed into three subproblems. The left subproblem consists of states 1 and 4; the middle subproblem consists of states 4 and 5; and the right corresponds to states 5 and 8. Note that the left and the right subproblems correspond to the top and bottom nodes of the

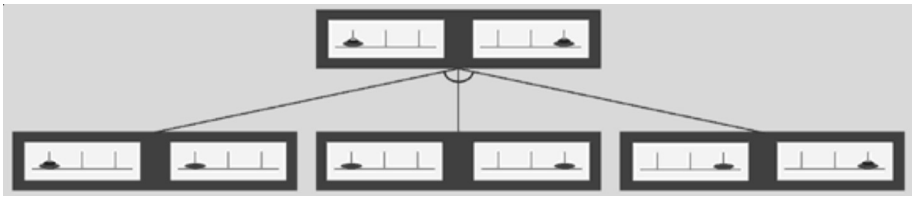
Notes

upper and lower triangles respectively. The middle subproblem corresponds to the move that links these two triangles of states. Note that this middle subproblem has no further decomposition. It is a primitive problem that corresponds to moving the large disk from the first to the third peg. The yellow border is used to depict primitive or terminal subproblems. The left and right subproblems are not primitive subproblems and they are each decomposed further. The three subproblems for each of these subproblems are primitive and correspond to the first three and the last three moves of the solution.

In this example, only AND nodes are shown. An OR node corresponds to the case where one or more different non-terminal rules are applied to a particular subproblem. For an OR node, at least one of the OR nodes must be solved in order to solve the (sub)problem.



Note that syntactically a decomposition is simply a set of (partial) state descriptions. *(This definition can be relaxed even further, but that is a rather long story.)* Thus, there are very many possible non-terminal reduction rules that could be defined for this 3 Disk Tower of Hanoi Problem. The figure above illustrates another possible non-terminal rule for this problem. In this case, the first subproblem is to move the middle and small sized disks to the right peg. The second subproblem is to move these disks from the right peg to the middle peg. This corresponds to moving down the left side of the triangle in the state space above and then moving across to the right side. Although this involves unnecessary moves, it leads to a valid solution when coupled with the appropriate further rules of decomposition.



The next figure above illustrates a non-terminal rule for this problem which involves *partial* states. Note that the first subproblem specifies as the goal state a situation where the large disk is on the left peg and no disks are on the large disk. (Of course, our picture can't express this last condition since the other two pegs aren't in our picture at all.) The next subproblem involves a goal situation where the large disk is now on the left peg and again no disks are on the large disk.

*Notes* That there are four states in the state space that satisfy the goal of the first subproblem and also four that satisfy the goal of the second subproblem. This observation forces us to recognize that implicit in the use of non-terminal rules in problem reduction is the assumption that there is a path in the state space that will be discovered when the partial state descriptions are bound to an appropriate state space.

Notes

### 9.5.3 Implementing the AND/OR-Tree Search Function

Another important feature of problem reduction rules is that their use allows the *order of problem solving* to differ from the o*rder of problem execution*. In the state space above, these two are obviously the same (although one could also search backward from the goal, but in the case of either forward or backward search the solution involves the same linear order of states.) In the present case, the problem solver could choose any of the subproblems to work on first.

In the Tower of Hanoi problem, there is no obvious advantage to solving the problem in an order that differs from the order of problem execution. However, in some problems one may find that some subproblems are easier than others and their solution may serve to simplify the solution of the remaining subproblems. A Cryptarithmetic Problem has been developed which illustrates this point. In this case, the subproblems correspond to the different equations that are formed in the algebraic representation of the problem. Note, that in this case the state space has 10! states and solving it via state space search is out of the question for any sane human mind. It thus uses depth first search and exits as soon as the first solution is found. The function will systematically explore the whole space if no solution can be found (if there are no loops) before returning <false>. We have used recursion to deal with the other goals. It would have been just as easy to have used a loop instead which looped through each of the goals input to the search function and merely called itself recursively in dealing with each of their sub-goals.

### 9.5.4 Implementing Planning as AND/OR-Tree Search

To implement a simple form of planning behaviour using our generic, AND/OR-tree search function, we first need to set up the appropriate AND/OR search space (i.e. rule-base). Below is a set rule which define an implicit AND/OR tree for a "book" problem. This is a variant on the "cake" problem of Burton and Shad bolt, 1987.

We have placed facts earlier in the rule-base than rules. This is to ensure that in searching, the system checks facts first before continuing the search via the rules. All the entries are lists of elements; in each case the first element corresponds to a goal and the remaining elements form the corresponding sub-goals or "preconditions". Calling the new search function on the Input corresponding to a goal of the planning task considered produces behaviour.

### 9.5.5 Implementing Reasoning as AND/OR-Tree Search

The AND/OR-tree search function is not only applicable in the case where we wish to solve a simplified planning problem. It can also be applied in many other cases. Any problem whose solution we can think of in terms of goal-reduction can be solved by the application of AND/OR-tree search. To obtain a specific implementation we only need to set up the appropriate rule base.

*Example:* AND/OR-tree search can be used to solve simple reasoning problems. In this application goals correspond roughly to "conclusions" and sub-goals correspond roughly to bits of "evidence". Search rules state that certain collections of evidence (sub-goals) allow the drawing (achievement) of a given conclusion (goal).

### Self Assessment

State whether the following statements are true or false:

17. The process of decomposing a goal into its sub goals is referred to as goal-reduction.

Notes

18. Any subproblem may not be decomposed into subproblems.

19. The AND/OR-tree search function is only applicable in the case where we wish to solve a simplified planning problem.

20. To obtain a specific implementation we only need to set up the appropriate rule base.

## 9.6 Summary

- The AI system are developed using the knowledge of an expert person (or persons) in the field where the system will be applied.
- Each node or vertex in the graph corresponds to a problem state, and arcs between nodes correspond to transformations or mapping between the states.
- Search can be characterized as finding a path through a graph or tree structure.
- The process of generating all of the children of a parent is also known as expanding the node.
- A search procedure is a strategy for selecting the order in which nodes are generated and a given path selected.
- The problem space of means-end analysis has as initial state and one or more goal states, a set of operators Ok with given preconditions for their application, and a difference function that computes the difference between two states Si and Sj.
- Information might relate to the problem space as a whole or to only some states.
- The depth-first search is preferred over the breadth-first when the search tree is known to have a plentiful number of goals. Otherwise, depth-first may never find a solution.

## 9.7 Keywords

***Backtracking:*** It helps in undoing what has been done so far and permits to try a totally different path to attain the global peak.

***Blind or Uninformed:*** It is a search algorithm that is one that uses no information other than the initial state, the search operators, and a test for a solution.

***Breadth-first Searches:*** These are performed by exploring all nodes at a given depth before proceeding to the next level.

***Depth-first Searches:*** These are performed by diving downward into a tree as quickly as possible

***Hill Climbing:*** It is like depth-first searching where the most promising child is selected for expansion.

***Informed Search Methods:*** These often depend on the use of heuristic information i.e. information about the problem (the nature of the states, the cost of transforming from one state to another, the promise of taking a certain path, and the characteristics of the goals) can sometimes be used to help guide the search more efficiently.

***Local Maximum:*** It is a state that is better than al its neighbours but not so when compared to states to states that are farther away.

***Plateau:*** A flat area of the search space, in which all neighbours have the same value.

***Ridge:*** Described as a long and narrow stretch of elevated ground or a narrow elevation or raised path running along or across a surface.

Notes

*Worst Case:* This situation the only information available will be the ability to distinguish goal from non-goal nodes.

## 9.8 Review Questions

1. Explain search and control strategies.
2. List four examples of search problems.
3. Explain and solve the water container problem.
4. Draw a diagram for blind search.
5. Explain with an example breadth-first search. Write its algorithm also.
6. Explain with an example depth-first search. Write its algorithm also.
7. Explain hill climbing methods.
8. Write the A* algorithm.

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | True | 4. | True |
| 5. | True | 6. | False |
| 7. | False | 8. | True |
| 9. | True | 10. | False |
| 11. | False | 12. | True |
| 13. | False | 14. | True |
| 15. | True | 16. | False |
| 17. | True | 18. | False |
| 19. | False | 20. | True |

## 9.9 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

Notes

*Online links*

ari.cankaya.edu.tr/AI/~ceng462/303/lect/chp3.htmlý

http://ai.stanford.edu/~latombe/cs121/2011/slides/C-blind-search.pdf

http://research.microsoft.com/en-us/um/people/gurevich/Opera/96.pdf

http://www.academia.edu/1056008/Search_and_check._Problem_solving_by_problem_reduction

https://web.cs.dal.ca/~mheywood/CSCI3154/Slides/3154-BlindSearch.pdf

www.dagstuhl.de/Reports/96/9647.pdfý

Notes

# Unit 10: Matching Techniques

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Discuss the Structures used in Matching
- Explain how to measure for Matching
- Describe Matching Like Patterns
- Define Prediction by Partial Matching
- Identify Fuzzy Matching Algorithms
- Discuss the Rete Matching Algorithm

## Introduction

Artificial Intelligence (AI) could be defined as the ability of computer software and hardware to do those things that we, as humans, recognize as intelligent behavior. Traditionally, those things include such activities as:

***Searching:*** Finding "good" material after having been provided only limited direction, especially from a large quantity of available data.

***Surmounting Constraints:*** Finding ways that something will fit into a confined space, taking apart or building a complex object, or moving through a difficult maze.

***Recognizing Patterns:*** Finding items with similar characteristics, or identifying an entity when not all its characteristics are stated or available.

***Making Logical Inferences:*** Drawing conclusions based upon understood reasoning methods such as deduction and induction.

Notes

AI technologies extend from the word *"Technology"* which stems from the Greek word *"technos"*, which means "art" and "skill." A sophisticated technology is then a cumulative building of learned and well-refined skills and processes. In the AI area, these processes have manifested themselves in a number of well-recognized and maturing areas including Neural Networks, Expert Systems, Automatic Speech Recognition, Genetic Algorithms, Intelligent Agents, Natural Language Processing, Robotics, Logic Programming, and Fuzzy Logic. Each of these areas will be examined in some depth here, but it is first important to understand that the importance of these individual areas has changed over the last two decades. These changes have been based upon the progress in each area, and the needs that each area meets. For example, in the early 1980s robotics was a large thrust in artificial intelligence. At that time benefits could be seen in manufacturing applications. In the late 1990s, the blossoming of the Internet pushed the importance of intelligent agents forward for performing routine tasks and complex searches. At the same time, throughout the 1980s and 1990s, orders of magnitude advances in computer processing power have allowed hurdles in speech recognition and image processing to be overcome. The maturity of each of these technology areas also differs. Expert Systems and Automatic Speech Recognition are among the most mature while Natural Language Processing and Intelligent Agents remain in early stages of development. In the next few paragraphs, the basis for each of these technologies will be reviewed. In addition examples where the technologies have been effectively utilized will be presented.

## 10.1 Structures Used in Matching

In computer science, pattern matching is the act of checking a perceived sequence of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact. The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence. Sequence patterns (e.g., a text string) are often described using regular expressions and matched using techniques such as backtracking.

Tree patterns are used in some programming languages as a general tool to process data based on its structure, e.g., Haskell, ML and the symbolic mathematics language Mathematica have special syntax for expressing tree patterns and a language construct for conditional execution and value retrieval based on it. For simplicity and efficiency reasons, these tree patterns lack some features that are available in regular expressions. Often it is possible to give alternative patterns that are tried one by one, which yields a powerful conditional programming construct. Pattern matching sometimes include support for guards.

The simplest pattern in pattern matching is an explicit value or a variable.

*Example:* Consider a simple function definition in Haskell syntax (function parameters are not in parentheses but are separated by spaces, = is not assignment but definition):

```
f 0 = 1
```

Here, 0 is a single value pattern. Now, whenever f is given 0 as argument the pattern matches and the function returns 1. With any other argument, the matching and thus the function fail. As the syntax supports alternative patterns in function definitions, we can continue the definition extending it to take more generic arguments:

```
f n = n * f (n-1)
```

Here, the first n is a single variable pattern, which will match absolutely any argument and bind it to name n to be used in the rest of the definition. Patterns are tried in order so the first

Notes

definition still applies in the very specific case of the input being 0, while for any other argument the function returns n `* f (n-1)` with n being the argument. The wildcard pattern (often written as _) is also simple: like a variable name, it matches any value, but does not bind the value to any name.

### Self Assessment

State whether the following statements are true or false:

1. Tree patterns are used in all programming languages as a general tool to process data based on its structure.
2. Pattern matching sometimes include support for guards.
3. Sequence patterns are often described using regular expressions and matched using techniques.

## 10.2 Measure for Matching

More complex patterns can be built from the primitive ones of the previous section, usually in the same way as values are built by combining other values. The difference then is that with variable and wildcard parts, a pattern doesn't build into a single value, but matches a group of values that are the combination of the concrete elements and the elements that are allowed to vary within the structure of the pattern.

A tree pattern describes a part of a tree by starting with a node and specifying some branches and nodes and leaving some unspecified with a variable or wildcard pattern. It may help to think of the abstract syntax tree of a programming language and algebraic data types.

In Haskell, the following line defines an algebraic data type `Color` that has a single data constructor `ColorConstructor` that wraps an integer and a string.

```
data Color = ColorConstructor Integer String
```

The constructor is a node in a tree and the integer and string are leaves in branches.

When we want to write functions to make Color an abstract data type, we wish to write functions to interface with the data type, and thus we want to extract some data from the data type, for example, just the string or just the integer part of Color.

If we pass a variable that is of type Color, how can we get the data out of this variable? For example, for a function to get the integer part of `Color`, we can use a simple tree pattern and write:

```
integerPart (ColorConstructor theInteger _) = theInteger
```

As well:

```
stringPart (ColorConstructor _ theString) = theString
```

The creations of these functions can be automated by Haskell's data record syntax.

### Filtering Data with Patterns

Pattern matching can be used to filter data of a certain structure. For instance, in Haskell a list comprehension could be used for this kind of filtering:

```
[A x|A x <- [A 1, B 1, A 2, B 2]]
```

evaluates to

```
[A 1, A 2]
```

Notes

### Pattern Matching in Mathematica

In Mathematica, the only structure that exists is the tree, which is populated by symbols. In the Haskell syntax used thus far, this could be defined as:

```
data SymbolTree = Symbol String [SymbolTree]
```

An example tree could then look like

```
Symbol "a" [Symbol "b" [], Symbol "c" [] ]
```

In the traditional, more suitable syntax, the symbols are written as they are and the levels of the tree are represented using [], so that for instance `a[b,c]` is a tree with *a* as the parent, and *b* and *c* as the children.

A pattern in Mathematica involves putting "_" at positions in that tree. For instance, the pattern

```
A[_]
```

will match elements such as A[1], A[2], or more generally A[*x*] where *x* is any entity. In this case, A is the concrete element, while _ denotes the piece of tree that can be varied. A symbol prepended to _ binds the match to that variable name while a symbol appended to _ restricts the matches to nodes of that symbol.

The Mathematica function `Cases` filters elements of the first argument that match the pattern in the second argument:

```
Cases[{a[1], b[1], a[2], b[2]}, a[_] ]
```

evaluates to

```
{a[1], a[2]}
```

Pattern matching applies to the *structure* of expressions. In the example below,

```
Cases[ {a[b], a[b, c], a[b[c], d], a[b[c], de, a[b[c], d, e]}, a[b[_],
_] ]
```

returns

```
{a[b[c],d], a[b[c],de}
```

because only these elements will match the pattern `a[b[_],_]` above.

In Mathematical, it is also possible to extract structures as they are created in the course of computation, regardless of how or where they appear. The function `Trace` can be used to monitor a computation, and return the elements that arise which match a pattern. For example, we can define the Fibonacci sequence as

```
fib[0|1]:=1
fib[n_]:= fib[n-1] + fib[n-2]
```

Then, we can ask the question: Given fib[3], what is the sequence of recursive Fibonacci calls?

`Trace[fib[3], fib[[_]]` returns a structure that represents the occurrences of the pattern `fib[_]` in the computational structure:

```
{fib[3],{fib[2],{fib[1]},{fib[0]}},{fib[1]}}
```

### Declarative Programming

In symbolic programming languages, it is easy to have patterns as arguments to functions or as elements of data structures. A consequence of this is the ability to use patterns to declaratively make statements about pieces of data and to flexibly instruct functions how to operate.

Notes

*Example:* The Mathematical function `Compile` can be used to make more efficient versions of the code. In the following example, the details do not particularly matter; what matters is that the sub expression {{com[_], Integer}} instructs `Compile` that expressions of the form `com[_]` can be assumed to be integers for the purposes of compilation:

```
com[i_]:= Binomial[2i, i]
Compile[{x, {i, _Integer}}, x^com[i], {{com[_], Integer}}]
```

### Pattern Matching and Strings

By far the most common form of pattern matching involves strings of characters. In many programming languages, a particular syntax of strings is used to represent regular expressions, which are patterns describing string characters. However, it is possible to perform some string pattern matching within the same framework that has been discussed throughout this article.

### Tree Patterns for Strings

Mathematically, strings are represented as trees of root String Expression and all the characters in order as children of the root. Thus, to match "any amount of trailing characters", a new wildcard ___ is needed in contrast to _ that would match only a single character. In Haskell and functional programming languages in general, strings are represented as functional lists of characters. A functional list is defined as an empty list, or an element constructed on an existing list. In Haskell syntax:

```
[] - an empty list
x:xs - an element x constructed on a list xs
```

The structure for a list with some elements is thus `element:list`. When pattern matching, we assert that a certain piece of data is equal to a certain pattern. For example, in the function:

```
head (element:list) = element
```

We assert that the first element of head's argument is called element, and the function returns this.

*Notes* We know that this is the first element because of the way lists are defined, a single element constructed onto a list. This single element must be the first. The empty list would not match the pattern at all, as an empty list does not have a head (the first element that is constructed).

In the example, we have no use for `list`, so we can disregard it, and thus write the function:

```
head (element:_) = element
```

The equivalent Mathematical transformation is expressed as

```
head[element, ]:=element
```

*Example:* It can be represented as,

```
StringExpression["a", ]
```

will match a string that has two characters and begins with "a".

The same pattern in Haskell:

```
['a', _]
```

Notes

Symbolic entities can be introduced to represent many different classes of relevant features of a string. For instance,

```
StringExpression[LetterCharacter, DigitCharacter]
```

will match a string that consists of a letter first, and then a number.

Guards could be used to achieve the same matches:

```
[letter, digit] | isAlpha letter && isDigit digit
```

The main advantage of symbolic string manipulation is that it can be completely integrated with the rest of the programming language, rather than being a separate, special purpose subunit. The entire power of the language can be leveraged to build up the patterns themselves or analyze and transform the programs that contain them.

*Caution* Use of pattern matching carefully.

*Task* Learn the working of character recognition from several websites.

### Self Assessment

State whether the following statements are true or false:

4. Strings are not represented as trees of root String Expression.
5. The function Trace can be used to monitor a computation, and return the elements that arise which match a pattern.
6. In Mathematica, some structure that exists is the tree, which is populated by symbols.

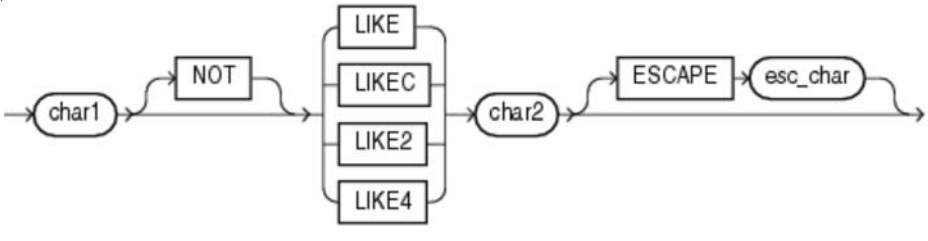
## 10.3 Matching Like Patterns

The pattern-matching conditions compare character data.

#### LIKE Condition

The LIKE conditions specify a test involving pattern matching. Whereas the equality operator (=) exactly matches one character value to another, the LIKE conditions match a portion of one character value to another by searching the first value for the pattern specified by the second. LIKE calculates strings using characters as defined by the input character set. LIKEC uses Unicode complete characters. LIKE2 uses UCS2 code points. LIKE4 uses UCS4 code points.

like_condition::=



**Notes**

In this syntax:

- *char1* is a character expression, such as a character column, called the **search value**.
- *char2* is a character expression, usually a literal, called the **pattern**.
- *esc_char* is a character expression, usually a literal, called the **escape character**.

The LIKE condition is the best choice in almost all situations. Use the following guidelines to determine whether any of the variations would be helpful in your environment:

- Use LIKE2 to process strings using UCS-2 semantics. LIKE2 treats a Unicode supplementary character as two characters.
- Use LIKE4 to process strings using UCS-4 semantics. LIKE4 treats a Unicode supplementary character as one character.
- Use LIKEC to process strings using Unicode complete character semantics. LIKEC treats a composite character as one character.

If *esc_char* is not specified, then there is no default escape character. If any of *char1*, *char2*, or *esc_char* is null, then the result is unknown. Otherwise, the escape character, if specified, must be a character string of length 1.

All of the character expressions (*char1*, *char2*, and *esc_char*) can be of any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. If they differ, then Oracle converts all of them to the datatype of *char1*.

The pattern can contain special pattern-matching characters:

- An underscore (_) in the pattern matches exactly one character (as opposed to one byte in a multibyte character set) in the value.
- A percent sign (%) in the pattern can match zero or more characters (as opposed to bytes in a multibyte character set) in the value. The pattern '%' cannot match a null.

*Did u know?* We can include the actual characters % or _ in the pattern by using the ESCAPE clause, which identifies the escape character. If the escape character precedes the character % or _ in the pattern, then **Oracle** interprets this character literally in the pattern rather than as a special pattern-matching character.

You can also search for the escape character itself by repeating it. For example, if @ is the escape character, then you can use @@ to search for @.

The table shown below describes the LIKE conditions.

| Type of Condition | Operation | Example |
|---|---|---|
| `x [NOT] LIKE y [ESCAPE 'z']` | `TRUE` if *x* does [not] match the pattern *y*. Within *y*, the character `%` matches any string of zero or more characters except null. The character _ matches any single character. Any character can follow `ESCAPE` except percent (%) and underbar (_). A wildcard character is treated as a literal if preceded by the escape character. | `SELECT last_name FROM employees WHERE last_name LIKE '%A\_B%' ESCAPE '\';` |

To process the LIKE conditions, Oracle divides the pattern into subpatterns consisting of one or two characters each. The two-character subpatterns begin with the escape character and the other character is %, or _, or the escape character.

Notes

Let $P_1$, $P_2$, ..., $P_n$ be these subpatterns. The like condition is true if there is a way to partition the search value into substrings $S_1$, $S_2$, ..., $S_n$ so that for all *i* between 1 and *n*:

- If $P_i$ is _, then $S_i$ is a single character.
- If $P_i$ is %, then $S_i$ is any string.
- If $P_i$ is two characters beginning with an escape character, then $S_i$ is the second character of $P_i$.
- Otherwise, $P_i = S_i$.

With the LIKE conditions, you can compare a value to a pattern rather than to a constant. The pattern must appear after the LIKE keyword. For example, you can issue the following query to find the salaries of all employees with names beginning with R:

```
SELECT salary
  FROM employees
  WHERE last_name LIKE 'R%';
```

The following query uses the = operator, rather than the LIKE condition, to find the salaries of all employees with the name 'R%':

```
SELECT salary
  FROM employees
  WHERE last_name = 'R%';
```

The following query finds the salaries of all employees with the name 'SM%'. Oracle interprets 'SM%' as a text literal, rather than as a pattern, because it precedes the LIKE keyword:

```
SELECT salary
  FROM employees
  WHERE 'SM%' LIKE last_name;
```

### *Case Sensitivity*

Case is significant in all conditions comparing character expressions that use the LIKE condition and the equality (=) operators. You can perform case or accent insensitive LIKE searches by setting the NLS_SORT and the NLS_COMP session parameters.

### Pattern Matching on Indexed Columns

When you use LIKE to search an indexed column for a pattern, Oracle can use the index to improve performance of a query if the leading character in the pattern is not % or _. In this case, Oracle can scan the index by this leading character. If the first character in the pattern is % or _, then the index cannot improve performance because Oracle cannot scan the index.

### *LIKE Condition: General Examples*

This condition is true for all last_name values beginning with Ma:

```
last_name LIKE 'Ma%'
```

All of these last_name values make the condition true:

```
Mallin, Markle, Marlow, Marvins, Marvis, Matos
```

Case is significant, so last_name values beginning with MA, ma, and mA make the condition false.

Consider this condition:

```
last_name LIKE 'SMITH_'
```

Notes

This condition is true for these last_name values:

```
SMITHE, SMITHY, SMITHS
```

This condition is false for SMITH because the special underscore character (_) must match exactly one character of the last_name value.

### ESCAPE Clause Example

The following example searches for employees with the pattern A_B in their name:

```
SELECT last_name
  FROM employees
  WHERE last_name LIKE '%A\_B%' ESCAPE '\';
```

The ESCAPE clause identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes Oracle to interpret the underscore literally, rather than as a special pattern matching character.

### Patterns Without % Example

If a pattern does not contain the % character, then the condition can be true only if both operands have the same length. Consider the definition of this table and the values inserted into it:

```
CREATE TABLE ducks (f CHAR(6), v VARCHAR2(6));
INSERT INTO ducks VALUES ('DUCK', 'DUCK');
SELECT '*'||f||'*' "char",
'*'||v||'*' "varchar"
FROM ducks;
char varchar
---- ---
*DUCK * *DUCK*
```

In many pattern, many object pattern matching, a collection of patterns is compared to a collection of objects, and all the matches are determined. That is, the pattern matcher finds every object that matches each pattern. This kind of pattern matching is used extensively in Artificial Intelligence programs today. For instance, it is a basic component of production system interpreters. The interpreters use it to determine which productions have satisfied condition parts. Unfortunately, it can be slow when large numbers of patterns or objects are involved. Some systems have been observed to spend more than nine-tenths of their total run time performing this kind of pattern matching. The simplest patterns contain only constant symbols and numbers. A pattern containing only constants matches a working memory element if every constant in the pattern occurs in the corresponding position in the working memory element.

We have seen SQL SELECT command to fetch data from MySQL table. We can also use a conditional clause called WHERE clause to select required records.

A WHERE clause with equal sign (=) works fine where we want to do an exact match. Like if "tutorial_author = 'Sanjay'". But there may be a requirement where we want to filter out all the results where tutorial_author name should contain "jay". This can be handled using SQL LIKE clause alongwith WHERE clause.

If SQL LIKE clause is used along with % characters then it will work like a meta character (*) in Unix while listing out all the files or directories at command prompt.

Without a % character LIKE clause is very similar to equal sign alongwith WHERE clause.

**Syntax:** **Notes**

Here is generic SQL syntax of SELECT command along with LIKE clause to fetch data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
```

### Self Assessment

State whether the following statements are true or false:

7. The LIKE conditions specify a test involving pattern matching.

8. LIKE2 uses USC4 code points.

9. *Char1* is a character expression, such as a character column, called the pattern.

## 10.4 Prediction by Partial Matching (PPM)

Prediction by Partial Matching (PPM) is an adaptive statistical data compression technique based on context modeling and prediction. PPM models use a set of previous symbols in the uncompressed symbol stream to predict the next symbol in the stream. PPM algorithms can also be used to cluster data into predicted groupings in cluster analysis.

Predictions are usually reduced to symbol rankings. The number of previous symbols, *n*, determines the order of the PPM model which is denoted as PPM(*n*). Unbounded variants where the context has no length limitations also exist and are denoted as *PPM**. If no prediction can be made based on all n context symbols a prediction is attempted with $n - 1$ symbols. This process is repeated until a match is found or no more symbols remain in context. At that point a fixed prediction is made. PPM compression implementations vary greatly in other details. The actual symbol selection is usually recorded using arithmetic coding, though it is also possible to use Huffman encoding or even some type of dictionary coding technique.

The underlying model used in most PPM algorithms can also be extended to predict multiple symbols. It is also possible to use non-Markov modeling to either replace or supplement Markov modeling. The symbol size is usually static, typically a single byte, which makes generic handling of any file format easy. Prediction by Partial Matching is a method to predict the next symbol depending on n previous. This method is else called prediction by Markov Model of order n. In case of text in natural language like English it is clear intuitively and proved by some researchers that probability of every next symbol is highly dependent on previous symbols. The most obvious method to use this feature is to incorporate it with one of known entropy compressions which have to gather statistics. The natural candidates are Huffman and Arithmetic compressions. It can be Adaptive or Static variants of this algorithms.

The key issue is to change alphabet to be alphabet of sequences and not alphabet of single symbols:

- Assume Markov model of order n.
- Let the input stream to be sequence x.
- For each symbol $x_i$ that is in sequence x on place i:
  - Update probability of sequence y, $y = [x_{i-n} \ldots x_i]$.
  - Let y to be a symbol in the target alphabet.

Notes

- Update all relevant structures in the basis algorithm.
- Perform the rest of needed steps required in the basis algorithm.

Notice definition of sequence y:

$$y = [x_{i-n} \dots x_i].$$

If $x_{i-n}$ or other particles of the sequence are not available, there are some options to handle this situation:

- Output $x_i$ uncoded and skip to the next symbol
- Add shortened symbol to the alphabet.

The first option is obvious and do not need farther clarification. The second option can be enhanced and should be explained more detailed. I will not touch this option directly because it can be implemented as variant of Multilevel PPM.

### Self Assessment

State whether the following statements are true or false:

10. Partial matching (PPM) is an adaptive statistical data compression technique based on context modeling and prediction.
11. Predictions are increased to symbol rankings.
12. The symbol size is usually static.

## 10.5 Fuzzy Matching Algorithms

Fuzzy matching is a mathematical process used to determine how similar one piece of data is to another.

*Example:* If you had 'John Arun Smith' in one system and wanted to find similar names from another system, the fuzzy matching process may return a list like this:

- J A Smith
- John A Smith
- JA Smith
- Jon Smith
- Jonathan A Smith
- Jon A Smithe

For each entry examined, the fuzzy matching process can give a probability score as to the accuracy of the match. For example, 'John A Smith' might receive a 95% score of similarity whereas 'JA Smith' would only get a 72% score.

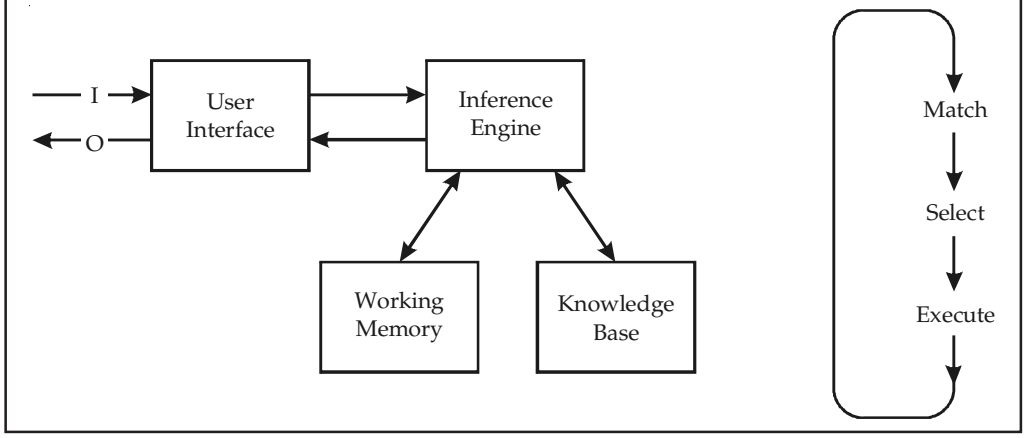
Fuzzy matching attempts to improve recall by being less strict but without sacrificing relevance. With fuzzy matching the algorithm is designed to find documents containing terms related to the terms used in the query. The assumption is that related words (in the English language) are likely to have the same core and differ at the beginning and/or end. A search for "matching", for example, would also return documents containing match, matched, etc. Unfortunately, it will also return documents containing unrelated words like matchbox, etc.

## 10.6 The Rete Matching Algorithm

Notes

- 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 Figure 10.1. These operations are performed as follows:

**Figure 10.1: 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

Notes

determine which rules have their LHS conditions satisfied with consistent bindings to working memory terms.

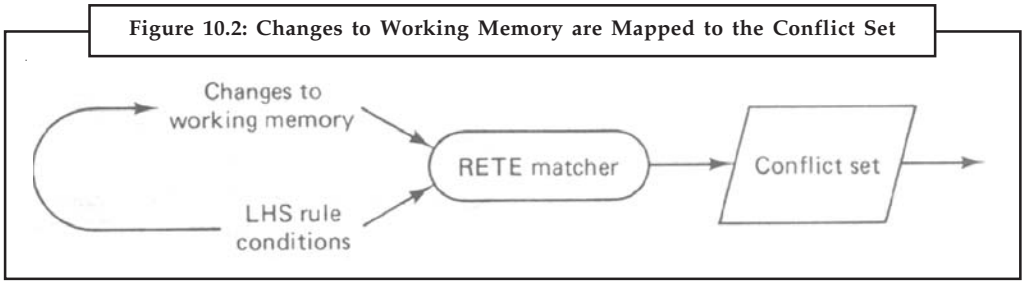
- Rules which are found to be applicable are put in a conflict set.

- *Select:*
  - From the conflict set, one of the rules is selected to execute. The selection strategy may depend on recency of usage, 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 Figure 10.2 shown below. 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.

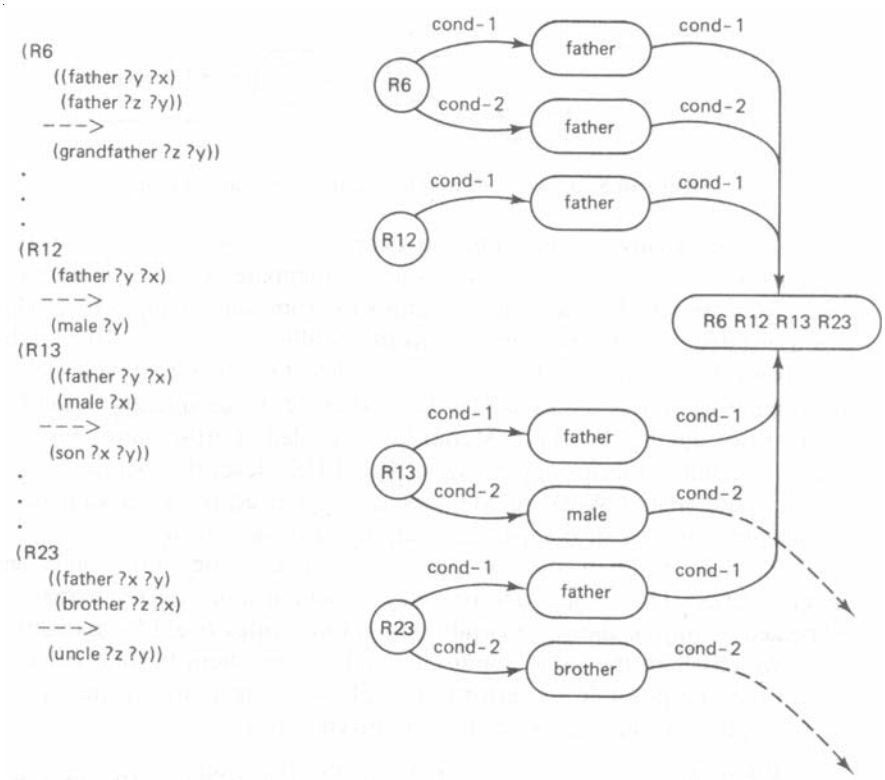
**Figure 10.2: Changes to Working Memory are Mapped to the Conflict Set**




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 Figure 10.3 shown below:

Notes

**Figure 10.3: Typical Rules and a Portion of a Compiled Network**




*Caution* Use of proper algorithm in pattern matching before applying it.

*Task* Make a list of five expert systems.

## Self Assessment

State whether the following statements are true or false:

13. The Rete is a directed, acyclic, rooted graph.
14. Fuzzy matching is a mathematical process used to determine how similar one piece of data is to another.
15. Fuzzy matching attempts to improve recall by being more strict but with sacrificing relevance.

## 10.7 Summary

- Matching is the process of comparing two or more structures to discover their likenesses or differences.

Notes

- 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.
- 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.
- A graph is a collection of points called vertices, some of which are connected by line segments called edges.
- 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.

## 10.8 Keywords

***Expert System:*** An expert system is a computer system that emulates the decision-making ability of a human expert.

***Fuzzy Logic:*** Is a form of many-valued logic or probabilistic logic; it deals with reasoning that is approximate rather than fixed and exact.

***LISP:*** It is a family of computer programming languages with a long history and a distinctive, fully parenthesized Polish prefix notation.

***Probabilistic Logic:*** It is to combine the capacity of probability theory to handle uncertainty with the capacity of deductive logic to exploit structure.

***RETE Algorithm:*** The RETE algorithm is an efficient pattern matching algorithm for implementing production rule systems.

## 10.9 Review Questions

1. What is pattern matching?
2. What are the different matching measurements for matching?
3. How is pattern matching important in real life?
4. Explain Prediction by partial matching.
5. Describe fuzzy matching algorithms.
6. Explain image processing.
7. Write the different probabilistic measures.
8. How does AI help in pattern matching?
9. Describe the working of any real life expert system.
10. Write the RETE algorithm.

### Answers: Self Assessment

1. True
2. True
3. True
4. False
5. True
6. False

7. True
8. False
9. False
10. True
11. False
12. True
13. True
14. True
15. False

Notes

## 10.10 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://blog.melissadata.com/mt-search.cgi?blog_id=2&tag=fuzzy%20matching%20algorithms&limit=20

http://docs.racket-lang.org/reference/match.html

http://social.msdn.microsoft.com/Forums/en-US/e7f51f55-0e03-4773-946b-fb0d45713e1d/fuzzy-matching-scoring-algorithm

http://www.cis.temple.edu/~giorgio/cis587/readings/rete.html

http://www.cs.princeton.edu/~rs/AlgsDS07/21PatternMatching.pdf

http://www.knockyoursocksoff.com/dataservices/Fuzzy.htm

http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/10dea1d3-fbef-2d10-0e89-a7447f95bc0e?QuickLink=index&overridelayout=true

Notes

# Unit 11: Knowledge Organization and Management

**CONTENTS**



## Objectives

After studying this unit, you will be able to:

- Discuss the nature of Knowledge Organizations
- Identify Indexing and Retrieval Techniques
- Explain Partial Match Techniques
- Describe Integrating Knowledge in Memory
- Explain Memory Organization System

## Introduction

A knowledge organization is a management idea, describing an organization in which people use systems and processes to generate, transform, manage, use, and transfer knowledge-based

Notes

products and services to achieve organizational goals. A knowledge organization also links past, present, and future by capturing and preserving knowledge in the past, sharing and mobilizing knowledge today, and knowledge organizations can be viewed from a number of perspectives: their general nature, networks, behavior, human dimensions, communications, intelligence, functions, and services.

## 11.1 Nature of Knowledge Organizations

The nature of an organization is based on knowledge rather than industrial society notions of land, labor or capital was not well understood. The core competencies are not what an organization owns but rather what it knows. Knowledge organizations have a network dimension. Davis (1977) states that networks would not replace hierarchies, but that the two would coexist within a broader organizational concept. Similarly, Amidon (1997) points out that traditional industrial-era hierarchies are neither flexible nor fluid enough to mobilize an organization's intellectual capacity and that much less constrained networked organizational forms are needed for modern decision making. There is an underlying logic and order to the emerging digital organizational form. It is networked, involves multiple enterprises, is based on core competencies, and knowledge is actively created, exchanged, and used.

There is also a behavioral approach. The organizational structure is just a skeleton. Knowledge organizations also have a physiology in the form of the flow of information and knowledge, as lifeblood. They also have a psychology represented by people's values and how they act as individuals and collectively.

Knowledge is created and used by people. Strassman (1985) described the transformation of work in the electronic age from the standpoint of education and training for managers and employees, human aspects of the working environment, and issues of morale, motivation, privacy, and displacements.

*Did u know?* The empowerment is not possible in an autocratic organization that networks cannot be sustained in fixed hierarchical structure, and that learning is not possible in an environment constrained by rigid policies and procedures.

Davenport (1997) used an information ecology approach, in which he explored the use and abuse of information in the context of infighting, resource hoarding, and political battles as well as appropriate management in such a context. Simard (2000) states that knowledge is inextricably linked to organizational mandates. Some providers strive for objectivity, others selectively disseminate information and knowledge, while still others use information to further their agenda. Users must understand that information is not innocent, and that all information is not created equal.

Knowledge organizations also have collective intelligence. From a functional perspective, in a knowledge organization, content (objects, data, information, knowledge, and wisdom) are generated by knowledge workers. Content is captured, organized, and preserved to enable its reuse and leveraging by people and groups other than those who generated it. Infrastructure is in place to enable sharing of content across all elements of an organization and with external partners, as appropriate. Procedures are in place to integrate content from multiple sources and mobilize it to achieve organizational goals and objectives. A learning culture promotes not only individual learning but also results in a shared understanding. Finally, the organization embraces continuous evolutionary change to sustain itself in a constantly changing environment. Simard et al. (2007) described five functions of a knowledge-service organization: generate content transform content into useful products and services preserve and manage content to enable organizational use and external transfer use content to achieve organizational goals, and

Notes

transfer content externally, in the form of products and services. Functions 1, 3, and 5 are essential and cannot be bypassed.

The knowledge in organizations is more complicated. Except from formal and informal documents, Davenport & Prusak's (1998) also introduced routines, processes, practices and norms. It may, therefore, be clear that organizational knowledge is much more than a sum of all the individual knowledge. It became clear that most of the knowledge an organization needs, already exists in the organization, but that finding and identifying it were the problems. This has several explanations.

First of all, it depends how the knowledge in an organization is organized. Hansen et al. (1999) made a distinction between a codification and personalization strategy. A codification strategy focuses as the word already implicates, to codify knowledge in a company, where a personalization strategy implies personal interaction as the main factor for exchanging knowledge. Both strategies have their problems. First of all, it is in many cases not possible to codify knowledge because of e.g. modifiability and complexity. This is called cognitive limitations.

"As expertise increases, mental representations become more abstract and simplified" And even if people can share knowledge, sometimes they do not want to. Hinds & Pfeffer (2001) call this motivational limitations. Hall et al. (2001) see rewards and informal activities with colleagues as the solution for the so-called motivational limitations. They divided the rewards in two groups, the explicit and the soft rewards. Explicit rewards are e.g. economic incentives and career advancement. Soft rewards are non-economic rewards such as reputation and satisfaction.

All this theory is based on formal structures in an organization. Organizations all also exist out informal ties and networks. This complicates organizational goals of formal knowledge sharing. A company will lose track of who has what knowledge because of the informal networks. This does not mean that informal networks are negative, just on the contrary, informal networks can create ties and surroundings which are necessary for knowledge sharing. The implication and unfortunately also the complication of informal networks is that an organization loose the general view of the knowledge in the organization. The informal structure significantly differs of the formal one. To effectively maximize the potential the authors argue that it is necessary to analyze both structures. They introduce the so-called Social Network analysis which systematically assesses informal networks. The result is that knowledge consequently can be collected from formal and informal networks to create a socio-knowledge matrix in organizations.

### Self Assessment

State whether the following statements are true or false:

1. Knowledge is created and used by people.
2. A learning culture promotes not only individual learning.
3. Functions 1, 3, and 5 are essential and can be bypassed.

## 11.2 Indexing and Retrieval Techniques

Following are the indexing and retrieval techniques:

### 11.2.1 The Frame Problem

A fluid mix of framed experience, values and contextual information that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied

Notes

in the minds of "knowers". In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms.

- Actionable information.
- The integration of ideas, experience, intuition, skill, and lessons learned that has the potential to create value for a business, its employees, products and services, customers and ultimately shareholders by informing decisions and improving actions.
- Knowledge is information combined with understanding and capability; it "lives" in the minds of people. Typically, knowledge provides a level of predictability that usually stems from the recognition of patterns.
- Knowledge is information that has been generalized to increase applicability.

Knowledge can be represented in following ways:

- ***Frame:*** A single know-what knowledge structure containing slots.
- ***Slot:*** Element of the frame that contains one or more facets.
- ***Facets:*** Element that describes something about a slot.
- ***Demons:*** Procedures attached to slots that are fired circumstantially.
- ***Instance:*** Frame example.

Also, we can have the following relationships between frames:

- ***Slot Sub-concepts:*** Contains links to other frames which represent sub-concepts.
- ***Slot Type:*** GENERIC or INSTANCE.
- Slot with facet other containing another frame.

Facets may take one of the following forms:

- ***Values:*** Contains the slot (single or multiple) value.
- ***Default:*** Used if there is not other value present.
- ***Range:*** Informs about the kind of information the slot can contain.
- ***if-added:*** Procedural attachment which specifies an action to be taken when a value in the slot is added or modified (forward chaining, data-driven, event-driven or bottom-up reasoning).
- ***if-needed:*** Procedural attachment which triggers a procedure which goes out to get information which the slot doesn't have (backward chaining, goal driven, expectation driven or top-down reasoning).
- ***Other:*** May contain frames, rules, semantic networks, or other types of knowledge

### 11.2.2 Indexed Organization

Each record in the file has one or more embedded keys (referred to as key data items); each key is associated with an index. An index provides a logical path to the data records according to the contents of the associated embedded record key data items. Indexed files must be direct-access storage files. Records can be fixed length or variable length.

Each record in an indexed file must have an embedded prime key data item. When records are inserted, updated, or deleted, they are identified solely by the values of their prime keys. Thus, the value in each prime key data item must be unique and must not be changed when the record

Notes

is updated. You tell COBOL the name of the prime key data item in the RECORD KEY clause of the file-control paragraph.

In addition, each record in an indexed file can contain one or more embedded alternate key data items. Each alternate key provides another means of identifying which record to retrieve. You tell COBOL the name of any alternate key data items on the ALTERNATE RECORD KEY clause of the file-control paragraph.

*Caution* The key used for any specific input-output request is known as the key of reference.

An indexed file contains records ordered by a record key. Each record contains a field that contains the record key. The record key uniquely identifies the record and determines the sequence in which it is accessed with respect to other records. A record key for a record might be, for example, an employee number or an invoice number.

An indexed file can also use alternate indexes, that is, record keys that let you access the file using a different logical arrangement of the records. For example, you could access the file through employee department rather than through employee number.

The record transmission (access) modes allowed for indexed files are sequential, random, or dynamic. When indexed files are read or written sequentially, the sequence is that of the key values.

### 11.2.3 Hashed Files

Computer security is an important aspect for most businesses. Many organizations use a process of hashing data into hash files to encrypt important data. Hash data is a numerical representation of data and is not easy for a human to interpret. A hash file is a file that has been converted into a numerical string by a mathematical algorithm. This data can only be understood after it has been unencrypted with a hash key.

The process of hashing is the mathematical conversion of a string of characters into a smaller value that is typically called a hash key. This new value represents the original character string after it has been encrypted. Hashing is often used in databases as a method of creating an index. Because hashed values are smaller than strings, the database can perform reading and writing functions faster.

*Notes* Hash files are commonly used as a method of verifying file size. This process is called check-sum verification. When a file is sent over a network, it must be broken into small pieces and reassembled after it reaches its destination. In this situation, the hash number represents the size of the data in a file. The hash can then be used as a tool for validating the entire file was successfully transmitted over the network.

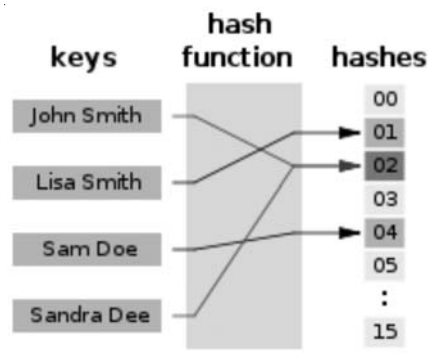
Hash functions are mostly used to accelerate table lookup or data comparison tasks such as finding items in a database, detecting duplicated or similar records in a large file, finding similar stretches in DNA sequences, and so on.

They represent one-way algorithms with a predefined size. If there was a decryption to hash functions, we would have a reversible function which is not available in this scenario. However, if there was, we would have the following. We know that the length could be any. Take a file with 1024 TB in size and parse it through a hash function. We get a block of [i]N (say N is 64 bits)

Notes

which is eight characters that we can even memorize. Now suppose you place those characters in your 'decrypter' and you get 1024 TB 'decompressed'. Most plainly said, we have a simple equation where we have three variables (a, b and c). Within hash functions we only know the value of the output which would be (if c = (a + b)) the letter 'c'. So we need 2*c attempts to find out the result which means brute-force.

**Figure 11.1: Working of Hash Function**




A hash function should be referentially transparent (stable), i.e., if called twice on input that is "equal" (for example, strings that consist of the same sequence of characters), it should give the same result. There is a construct in many programming languages that allows the user to override equality and hash functions for an object: if two objects are equal, their hash codes must be the same. This is crucial to finding an element in a hash table quickly, because two of the same element would both hash to the same slot.

All hash functions that map a larger set of data to a smaller set of data cause collisions. Such hash functions try to map the keys to the hash values as evenly as possible because collisions become more frequent as hash tables fill up. Thus, single-digit hash values are frequently restricted to 80% of the size of the table. Depending on the algorithm used, other properties may be required as well, such as double hashing and linear probing. Although the idea was conceived in the 1950s, the design of good hash functions is still a topic of active research.

Hash functions are related to (and often confused with) checksums, check digits, fingerprints, randomization functions, error correcting codes, and cryptographic hash functions. Although these concepts overlap to some extent, each has its own uses and requirements and is designed and optimized differently.

## Hash Tables

Hash functions are primarily used in hash tables, to quickly locate a data record (e.g., a dictionary definition) given its search key (the headword). Specifically, the hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored. Hash tables, in turn, are used to implement associative arrays and dynamic sets.

Typically, the domain of a hash function (the set of possible keys) is larger than its range (the number of different table indexes), and so it will map several different keys to the same index. Therefore, each slot of a hash table is associated with (implicitly or explicitly) a set of records, rather than a single record. For this reason, each slot of a hash table is often called a *bucket*, and hash values are also called *bucket indices*.

Thus, the hash function only hints at the record's location—it tells where one should start looking for it. Still, in a half-full table, a good hash function will typically narrow the search

Notes

down to only one or two entries. Hash functions are also used to build caches for large data sets stored in slow media. A cache is generally simpler than a hashed search table, since any collision can be resolved by discarding or writing back the older of the two colliding items. This is also used in file comparison.

### 11.2.4 Conceptual Indexing

Teaching for conceptual change primarily involves – uncovering students' preconceptions about a particular topic or phenomenon and using various techniques to help students change their conceptual framework. The vast majority of research on conceptual change instruction has been confined to science education, while research in other subject areas has been scarce. However, outside of school, students develop strong (mis)conceptions about a wide range of concepts related to non-scientific domains, such as how the government works, principles of economics, the utility of mathematics, the reasons for the Civil Rights movement, the nature of the writing process, and the purpose of the electoral college. Conceptual change instruction can help students overcome misconceptions and learn difficult concepts in all subject areas.

Conceptual change is not only relevant to teaching in the content areas, but it is also applicable to the professional development of teachers and administrators.

*Example:* As constructivist approaches to teaching gain popularity, the role of the teacher changes. Teachers must learn different instructional strategies, but they must also reconceptualize or change their conception about the meaning of teaching. This change implies conceiving of teaching as facilitating, rather than managing learning and changing roles from the "sage on the stage" to a "guide on the side." Likewise, a shift has occurred for school media specialists (formerly known as librarians). Their role in the school has changed from being "keeper of the books" to "collaborative planner," working in partnership with teachers, school administrators and the community.

Conceptual change is of particular relevance in business and professional communities. Companies often restructure, changing their business strategies and processes to remain competitive and responsive to the needs of their customers. The advancement of technology has also initiated a trend in the restructuring of industrialization. Lansky states that technological innovation, globalization, and industrial relocation are leaving only two general types of paid work in advanced industrialized countries: technical jobs, which center on problem-solving, and interpersonal jobs, which require a "human touch". Lansky (2000) observes that the contemporary work force can be divided into three categories. The first is "highly skilled and highly paid technicians [and] providers of interpersonal services". The second group consists of "lower paid technicians and lower paid providers of interpersonal services". The third group comprises workers "without the education, skills or connections needed to become technicians or interpersonal workers" (Lansky, 2000). Due to this trend toward "upskilling," the first and second groups benefit from the changes in industry; the third group faces the possibility of unemployment.

Conceptual change is not limited to the company level; it can also occur at the level of an entire industry. Hospitals are now known as "medical centers." Those who receive care in such facilities are now termed "clients" rather than "patients." Banks have been replaced by "financial centers." Employees of organizations are now considered "associates." Companies are beginning to view themselves as "learning organizations" instead of "corporations." These are not simply changes in terminology; the changes signify conceptual change.

Restructuring in companies or entire industries often requires employees to re-conceptualize their roles and responsibilities and change the way they perform their jobs. Teaching new job

Notes

skills and new procedures to employees is relatively easy to accomplish; bringing about a conceptual change in how these workers view their organizational roles is a far more difficult undertaking. One such example is the shift in scope of practice for health psychologists. The role of these practitioners has shifted away from that of "peripheral case consultants" who simply provide psychological interventions to "primary care case managers," who actively manage the care of their clients. In reflecting on training health psychologists as primary care case managers, James and Folen emphasize "*that the most arduous task in bringing about this shift was neither the training nor the demanding workload. Rather, it was bringing about a conceptual shift in traditionally trained psychology interns, whose training resists going beyond traditional psychological interventions*". In the early 1980s, a group of science education researchers and science philosophers at Cornell University developed a theory of conceptual change . This theory is based on Piaget's notions of disequilibration and accommodation as well as Thomas Kuhn's description of scientific revolution (Kuhn, 1970). According to Kuhn, scientific revolutions have followed a consistent pattern. Firstly, a dominant scientific paradigm – a basic way of perceiving, thinking, valuing, and doing – fell into a "state of crisis" by failing to provide solutions or explanations to deal with significant problems identified by the scientific community. Secondly, an alternative paradigm with the potential to solve these problems had to be available. The existence of these two conditions increased the probability of a "paradigm shift," or universal adoption of a new framework for thinking.

### 11.2.5 Retrieval Techniques

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval, a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

An object is an entity that is represented by information in a database. User queries are matched against the database information. Depending on the application the data objects may be, for example, text documents, images, audio, mind maps or videos. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates or metadata.

Most IR systems compute a numeric score on how well each object in the database matches the query, and rank the objects according to this value. The top ranking objects are then shown to the user. The process may then be iterated if the user wishes to refine the query.

*Caution* Don't apply all information. First extract it and convert it into knowledge.

*Task* Draw a knowledge discovery diagram (KDD).

## Self Assessment

State whether the following statements are true or false:

4. Knowledge is information that has been generalized to increase applicability.
5. Values informs about the kind of information the slot can contain.
6. Records can not be fixed length or variable length.

Notes

## 11.3 Partial Match Techniques

Many different measures for evaluating the performance of information retrieval systems have been proposed. The measures require a collection of documents and a query. All common measures described here assume a ground truth notion of relevancy: every document is known to be either relevant or non-relevant to a particular query. In practice, queries may be ill-posed and there may be different shades of relevancy.

Precision is the fraction of the documents retrieved that are relevant to the user's information need. In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. Note that the meaning and usage of "precision" in the field of Information Retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

In binary classification, recall is often called sensitivity. So it can be looked at as the probability that a relevant document is retrieved by the query.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available.

### 11.3.1 Feature-based Techniques

The feature-based techniques extract local regions of interest (features) from the images and identify the corresponding features in each image of the sequence. Such algorithms have initially been developed for tracking a small number of salient features in long image sequences. The tracking process can be divided into two major subtasks: feature extraction and feature tracking. One can extract features only in the first image and search for the same features in the subsequent images. This dynamic feature extraction scheme is advocated by the KLT Tracker. Alternatively, one can find features in each static image prior to tracking and then find the corresponding features along the sequence. Most of the feature tracking algorithms, including our IPAN Tracker, operate in this way, although the dynamic feature extraction seems to be more natural and reliable than the static one: the temporal information helps decide where and how to search for features in the next frame.

In order to make use of features in processing geometric models, some core common aspects exist. These apply irrespective of the domain of problems and the particular problems at hand. The relevance of individual aspects may vary. These are as follows:

1. ***Identication of Features:*** A particular feature of interest in a given 3D model must be specified in a way such that it can be algorithmically identified in the 3D data. This definition can be a mix of topological properties, geometric metrics, colour and texture attributes and so on. Some higher-level features may be defined in terms of simpler low-level features. Such hierarchies of features are common in literature.
2. ***Recognition of Features:*** Recognition of a feature in the given 3D model is an essential computational part feature-based processing of geometric models. For detecting an individual model, typically specialised filters need to be implemented.
3. ***Suppressing Features:*** By suppressing a feature we mean removing a local instance of the feature while minimally disturbing the data around the feature.

Notes

4. ***Reconstruction of Features:*** This is an inverse of "suppressing features." By reconstruction we mean restoration of a previously suppressed feature in the data, preferably, without any loss of information of the original data.
5. ***Encoding of Features:*** A machine representation of features, preferably a compact one, must be developed.
6. ***Feature Space Conversion:*** Given a model described in one feature space, often there is a requirement to convert the description to another feature space.

### 11.3.2 Structure-based Techniques

Structure-based techniques serve two purposes: test coverage measurement and structural test case design. They are often used first to assess the amount of testing performed by tests derived from specification-based techniques, i.e. to assess coverage. They are then used to design additional tests with the aim of increasing the test coverage. Structure-based test design techniques are a good way of generating additional test cases that are different from existing tests. They can help ensure more breadth of testing, in the sense that test cases that achieve 100% coverage in any measure will be exercising all parts of the software from the point of view of the items being covered.

Software testing is a process of executing a program or application with the intent of finding the software bugs.

It can also be stated as the process of validating and verifying that a software program or application or product. Meets the business and technical requirements that guided it's design and development. It provides Works as expected. It can be implemented with the same characteristic.

Let's break the definition of software testing into the following parts:

1. ***Process:*** Testing is a process rather than a single activity.
2. ***All Life Cycle Activities:*** Testing is a process that's take place throughout the Software Development Life Cycle (SDLC).

The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it's referred as "verifying the test basis via the test design". The test basis includes documents such as the requirements and design specifications.

***Static Testing:*** It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example, reviewing, walkthrough, inspection, etc.

***Dynamic Testing:*** In dynamic testing, the software code is executed to demonstrate the result of running tests. It's done during validation process. For example, unit testing, integration testing, system testing, etc.

***Planning:*** We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.

***Preparation:*** We need to choose what testing we will do, by selecting test conditions.

***Evaluation:*** During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.

***Software Products and Related Work Products:*** Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

Notes

### Self Assessment

State whether the following statements are true or false:

7. In binary classification, precision is analogous to negative predictive value.
8. Recognition of a feature in the given 3D model is an essential computational part feature-based processing of geometric models.
9. Structure-based test design techniques are a good way of generating additional test cases that are different from existing tests.

## 11.4 Integrating Knowledge in Memory

***Knowledge integration*** is the process of synthesizing multiple knowledge models (or representations) into a common model (representation).

Comparison of information integration involves merging information having different schemas and representation models, knowledge integration focuses more on synthesizing the understanding of a given subject from different perspectives.

*Example:* Multiple interpretations are possible of a set of student grades, typically each from a certain perspective.

An overall, integrated view and understanding of this information can be achieved if these interpretations can be put under a common model, say, a student performance index. The Web-based Inquiry Science Environment (WISE), from the University of California at Berkeley has been developed along the lines of knowledge integration theory.

***Knowledge integration*** has also been studied as the process of incorporating new information into a body of existing knowledge with an interdisciplinary approach. This process involves determining how the new information and the existing knowledge interact, how existing knowledge should be modified to accommodate the new information, and how the new information should be modified in light of the existing knowledge.

A learning agent that actively investigates the consequences of new information can detect and exploit a variety of learning opportunities; e.g., to resolve knowledge conflicts and to fill knowledge gaps. By exploiting these learning opportunities the learning agent is able to learn beyond the explicit content of the new information. The machine learning program KI, developed by Murray and Porter at the University of Texas at Austin, was created to study the use of automated and semi-automated knowledge integration to assist knowledge engineers constructing a large knowledge base. A possible technique which can be used is semantic matching. More recently, a technique useful to minimize the effort in mapping validation and visualization has been presented which is based on Minimal Mappings. Minimal mappings are high quality mappings such that (i) all the other mappings can be computed from them in time linear in the size of the input graphs, and (ii) none of them can be dropped.

### 11.4.1 Hypertext

A special type of database system, invented by Ted Nelson in the 1960s, in which objects (text, pictures, music, programs, and so on) can be creatively linked to each other. When you select an object, you can see all the other objects that are linked to it. You can move from one object to another even though they might have very different forms.

Notes

*Example:* While reading document about Mozart, you might click on the phrase *Violin Concerto in A Major,* which could display the written score or perhaps even invoke a recording of the concerto. Clicking on the name *Mozart* might cause various illustrations of Mozart to appear on the screen. The icons that you select to view associated objects are called *Hypertext links* or buttons.

Hypertext systems are particularly useful for organizing and browsing through large databases that consist of disparate types of information. There are several Hypertext systems available for Apple Macintosh computers and PCs that enable you to develop your own databases. Such systems are often called *authoring systems*. HyperCard software from Apple Computer is the most famous.

Hypertext is the presentation of information as linked network of nodes which readers are free to navigate in a non-linear fashion. It allows for multiple authors, a blurring of the author and reader functions, extended works with diffuse boundaries, and multiple reading paths.

The term "hypertext" was defined as "non-sequential writing".

Many subsequent writers have taken hypertext to be a distinctly electronic technology – one which must involve a computer.

*Did u know?* Janet Fiderio, in her overview "A Grand Vision," writes:

"*Hypertext, at its most basic level, is a DBMS that lets you connect screens of information using associative links. At its most sophisticated level, hypertext is a software environment for collaborative work, communication, and knowledge acquisition. Hypertext products mimic the brain's ability to store and retrieve information by referential links for quick and intuitive access.*"

### Self Assessment

State whether the following statements are true or false:

10. Hypertext systems are particularly useful for organizing and browsing through large databases.
11. The term "hypertext" was defined as sequential writing.
12. HyperCard software from Apple Computer is the most famous.

## 11.5 Memory Organization System

In the following aspects, you can see the Memory Organization System:

### 11.5.1 HAM, a Model of Memory

Storage in human memory is one of three core process of memory, along with recall and encoding. It refers to the retention of information, which has been achieved through the encoding process, in the brain for a prolonged period of time until it is accessed through recall. Modern memory psychology differentiates the two distinct type of memory storage: short-term memory and long-term memory. In addition, different memory models have suggested variations of existing short-term and long-term memory to account for different ways of storing memory. Varieties of different memory models have been proposed to account for different types of recall processes, including cued recall, free recall, and serial recall. In order to explain the recall

Notes

process, however, the memory model must identify how an encoded memory can reside in the memory storage for a prolonged period of time until the memory is accessed again, during the recall process.

*Notes* Not all models use the terminology of short-term and long-term memory to explain memory storage; the Dual-Store theory and refined version of Atkinson-Shiffrin Model of Memory (Atkinson 1968) uses both short-term and long-term memory storage, but others do not.

The short-term memory refers to the ability to hold information from immediate past for a short duration of time. According to the Atkinson-Shiffrin Model of Memory, in the process of Encoding, perceived memory enters the brain and can be quickly forgotten if the sensory information is not stored further in the short-term memory. The information is readily accessible in the short-term memory for only a short time. Baddeley suggested that memory stored in short-term memory is continuously deteriorating, which can eventually lead to forgetting in the absence of rehearsal. George A. Miller suggested in his paper that the capacity of the short-term memory storage is approximately seven items, plus or minus two, but modern researchers are showing that this itself is subject to numerous variability, including the stored items' phonological properties.

### 11.5.2 Memory Organization with E-MOPs

Roger Schank and his students at Yale University have developed several computer systems which perform different functions related to the use of natural language, text, knowledge representation, and memory organization. One system of particular interest was developed by Janet Kolodner (1983a, 1983b, 1984) to study problems associated with the retrieval and organization of reconstructive memory. Her system, 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 posted in English. The system has many of the other features. The basic memory model in CYRUS is a network consisting of Episodie 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-MOPs are indexed in memory by one or more distinguishing features.

*Example:* There are basic E-MOPs for diplomatic meetings with foreign dignitaries, specialized political conferences, traveling, sight-seeing, negotiations, state dinners, as well as other basic events related to diplomatic state functions.

The diplomatic-meeting E-MOP, called $MEET, contains information which is common to all diplomatic meeting events. The common information which characterizes such an E-MOP is called its content.

*Caution* Use of truth table should be handled carefully.

*Task* Find the effect of propositional logic in real life.

## Self Assessment

Notes

State whether the following statements are true or false:

13. The long-term memory refers to the ability to hold information from immediate past for a short duration of time.
14. Varieties of same memory models have been proposed to account for different types of recall processes.
15. E-MOPs are indexed in memory by one or more distinguishing features.

## 11.6 Summary

- The maim aspect of knowledge organization is frame problem.
- Among the different file organization is indexed organizations that allows efficient access are based on the use of a hash function.
- One of the important issue is retrieval techniques is concerned with a specific aspect of research and development (R&D) in information retrieval (IR) systems–that is, the means for identifying retrieving, and/or ranking texts in a collection of texts, that might be relevant to a given query (or useful for resolving a particular problem).
- Hypertext systems are examples of information organized through associative links, like associative networks.
- The HAM is informed of new sentences; they are parsed and formed into new tree-like memory structures or integrated with existing ones.

## 11.7 Keywords

***Exact Match Retrieval Techniques:*** These are those that require that the request model be contained, precisely as represented in the query formulation, within the text representation.

***Expert System:*** It can be expected to have thousands or even tens of thousands of rules (or their equivalent) in its KB.

***Index File:*** It is searched to find the desired record key and obtain the corresponding block address the block is then accessed using this address.

***Indexing:*** It is accomplished by organizing the information in some way for easy access one way to index is by segregating knowledge.

***Object Attributes:*** It can also serve as indices to locate items based on the attribute values.

***Retrieval Techniques:*** These are based on formal models of document retrieval and indexing.

## 11.8 Review Questions

1. Explain knowledge organization and management in AI.
2. Explain the indexing and retrieval technique.
3. What is integrating knowledge in memory?
4. Explain the important characteristics should a computer memory organization system possess.
5. Describe the basic operations must a program perform in order to access specific chunks of knowledge?

Notes

## Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | False | 4. | True |
| 5. | False | 6. | False |
| 7. | False | 8. | True |
| 9. | True | 10. | True |
| 11. | False | 12. | True |
| 13. | False | 14. | False |
| 15. | True | | |

## 11.9 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://www.cs.cmu.edu/~wing/publications/Wing79.pdf

http://www.cs.tut.fi/~moncef/publications/MTA_Editorial_09_2006.pdf

http://www.di.unito.it/~phd/documents/tesi/XVII/ValenteFinal.pdf

http://www.mif.vu.lt/~algis/DSA/sciencertree9.pdf

http://www.open.edu/openlearn/money-management/management/technology-management/knowledge-technologies-context/content-section-3.4.1

http://www.rci.rutgers.edu/~ph221/publications/Hemmer_Steyvers_PB_R.pdf

http://www.unc.edu/~sunnyliu/inls258/Introduction_to_ Knowledge_Management.html

# Unit 12: Natural Language Processing

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Discuss an overview of Linguistics
- Explain Grammars and Languages in AI
- Identify various Basic Parsing Techniques
- Describe Semantic Analysis and Representation Structures
- Discuss Natural Language Generalization

## Introduction

**Natural Language Processing** (**NLP**) is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human – computer interaction. Many challenges in NLP involve natural language understanding – that is, enabling computers to derive meaning from

Notes

human or natural language input. Up to the 1980s, most NLP systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a revolution in NLP with the introduction of learning algorithms for language processing. This was due both to the steady increase in computational power resulting from Moore's Law and the gradual lessening of the dominance of Chomskyan theories of linguistics (e.g. transformational grammar), whose theoretical underpinnings discouraged the sort of corpus linguistics that underlies the machine-learning approach to language processing. Some of the earliest-used machine learning algorithms, such as decision trees, produced systems of hard if-then rules similar to existing hand-written rules. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to the features making up the input data. The caches language models upon which many speech recognition systems now rely are examples of such statistical models. Such models are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data), and produce more reliable results when integrated into a larger system comprising multiple subtasks.

Many of the notable early successes occurred in the field of machine translation, due especially to work at IBM Research, where successively more complicated statistical models were developed. These systems were able to take advantage of existing multilingual textual corpora that had been produced by the Parliament of Canada and the European Union as a result of laws calling for the translation of all governmental proceedings into all official languages of the corresponding systems of government. However, most other systems depended on corpora specifically developed for the tasks implemented by these systems, which was (and often continues to be) a major limitation in the success of these systems. As a result, a great deal of research has gone into methods of more effectively learning from limited amounts of data.

Recent research has increasingly focused on unsupervised and semi-supervised learning algorithms. Such algorithms are able to learn from data that has not been hand-annotated with the desired answers, or using a combination of annotated and non-annotated data. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, there is an enormous amount of non-annotated data available (including, among other things, the entire content of the World Wide Web), which can often make up for the inferior results.

## 12.1 Overview of Linguistics

Linguistics is the study of human languages. It follows scientific approach. So it is also referred to as linguistic science. Linguistics deals with describing and explaining the nature of human languages. It treats language and the ways people use it as phenomena to be studied. Linguist is one who is expertise in linguistics. Linguist studies the general principles of language organization and language behavior. Linguistic analysis concerns with identifying the structural units and classes of language. Linguists also attempt to describe how smaller units can be combined to form larger grammatical units such as how words can be combined to form phrases, phrases can be combined to form clauses, and so on. They also concerns what constrains the possible meanings for a sentence. Linguists use intuitions about well-formedness and meaning and mathematical models of structure such as formal language theory and model theoretic semantics. Structure of language include morphemes, words, phrases, and grammatical classes. Sub-fields with respect to linguistic structure are phonetics, phonology, morphology, syntax, semantics, pragmatics, and discourse analysis. There are many branches of linguistics including applied linguistics, computational linguistics, evolutionary linguistics, neurolinguistics, cognitive linguistics and psycholinguistics. A linguist in the academic sense is a person who studies natural language (an academic discipline known as linguistics).

Notes

*Notes* Ambiguously, the word is sometimes also used to refer to a polyglot (one who knows several languages), or a grammarian (a scholar of grammar), but these two uses of the word are distinct (and one does not have to be a polyglot in order to be an academic linguist).

### 12.1.1 Computational Linguistics

Computational linguistics is an interdisciplinary field dealing with the statistical or rule-based modeling of natural language from a computational perspective.

Traditionally, computational linguistics was usually performed by computer scientists who had specialized in the application of computers to the processing of a natural language. Computational linguists often work as members of interdisciplinary teams, including linguists (specifically trained in linguistics), language experts (persons with some level of ability in the languages relevant to a given project), and computer scientists. In general, computational linguistics draws upon the involvement of linguists, computer scientists, experts in artificial intelligence, mathematicians, logicians, philosophers, cognitive scientists, cognitive psychologists, psycholinguists, anthropologists and neuroscientists, among others.

Computational linguistics has theoretical and applied components, where theoretical computational linguistics takes up issues in theoretical linguistics and cognitive science, and applied computational linguistics focuses on the practical outcome of modelling human language use. Computational linguistics (CL) is a discipline between linguistics and computer science which is concerned with the computational aspects of the human language faculty. It belongs to the cognitive sciences and overlaps with the field of artificial intelligence (AI), a branch of computer science aiming at computational models of human cognition. Computational linguistics has applied and theoretical components.

### Self Assessment

State whether the following statements are true or false:

1. Linguistics is the study of human languages.
2. There are some branches of linguistics.
3. Linguists also attempt to describe how bigger units can be combined to form smaller grammatical units.

## 12.2 Grammars and Languages

In formal language theory, a grammar (when the context isn't given, often called a formal grammar for clarity) is a set of production rules for strings in a formal language. The rules describe how to form strings from the language's alphabet that are valid according to the language's syntax. A grammar does not describe the meaning of the strings or what can be done with them in whatever context—only their form.

Formal language theory, the discipline which studies formal grammars and languages, is a branch of applied mathematics. Its applications are found in theoretical computer science, theoretical linguistics, formal semantics, mathematical logic, and other areas.

Notes

A formal grammar is a set of rules for rewriting strings, along with a "start symbol" from which rewriting starts. Therefore, a grammar is usually thought of as a language generator. However, it can also sometimes be used as the basis for a "recognizer"—a function in computing that determines whether a given string belongs to the language or is grammatically incorrect. To describe such recognizers, formal language theory uses separate formalisms, known as automata theory.

*Did u know?* One of the interesting results of automata theory is that it is not possible to design a recognizer for certain formal languages.

Parsing is the process of recognizing an utterance (a string in natural languages) by breaking it down to a set of symbols and analyzing each one against the grammar of the language. Most languages have the meanings of their utterances structured according to their syntax—a practice known as compositional semantics. As a result, the first step to describing the meaning of an utterance in language is to break it down part by part and look at its analyzed form.

A grammar mainly consists of a set of rules for transforming strings. (If it only consisted of these rules, it would be a semi-Thue system.) To generate a string in the language, one begins with a string consisting of only a single start symbol. The production rules are then applied in any order, until a string that contains neither the start symbol nor designated non-terminal symbols is produced. A production rule is applied to a string by replacing one occurrence of its left-hand side in the string by its right-hand side (cf. the operation of the theoretical Turing machine). The language formed by the grammar consists of all distinct strings that can be generated in this manner. Any particular sequence of production rules on the start symbol yields a distinct string in the language. If there are multiple ways of generating the same single string, the grammar is said to be ambiguous.

*Example:* Assume the alphabet consists of *a* and *b*, the start symbol is *S*, and we have the following production rules:

$$S \rightarrow aSb$$

$$S \rightarrow ba$$

Then we start with *S*, and can choose a rule to apply to it. If we choose rule 1, we obtain the string *aSb*. If we then choose rule 1 again, we replace *S* with *aSb* and obtain the string *aaSbb*. If we now choose rule 2, we replace *S* with ba and obtain the string aababb, and are done. We can write this series of choices more briefly, using symbols:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb.$$

The language of the grammar is then the infinite set $\{a^n bab^n \mid n \geq 0\} = \{ba, abab, aababb, aaababbb, \ldots\}$, where $a^k$ is *a* repeated *k* times.

A context-free grammar is a grammar in which the left-hand side of each production rule consists of only a single non-terminal symbol. This restriction is non-trivial; not all languages can be generated by context-free grammars. Those that can are called context-free languages.

The language defined above is not a context-free language, and this can be strictly proven using the pumping lemma for context-free languages, but for example the language $\{a^n b^n \mid n \geq 1\}$ (at least 1 *a* followed by the same number of *b*'s) is context-free, as it can be defined by the grammar $G_2$ with $N = \{S\}$, $\Sigma = \{a, b\}$, *S* the start symbol, and the following production rules:

1. $S \rightarrow aSb$
2. $S \rightarrow ab$

Notes

A context-free language can be recognized in $O(n^3)$ time. For every context-free language, a machine can be built that takes a string as input and determines in $O(n^3)$ time whether the string is a member of the language, where $n$ is the length of the string. Deterministic context-free languages are a subset of context-free languages that can be recognized in linear time. There exist various algorithms that target either this set of languages or some subset of it.

In regular grammars, the left hand side is again only a single non-terminal symbol, but now the right-hand side is also restricted. The right side may be the empty string, or a single terminal symbol, or a single terminal symbol followed by a non-terminal symbol, but nothing else.

The language defined above is not regular, but the language $\{a^n b^n \mid m, n \geq 1\}$ (at least 1 $a$ followed by at least 1 $b$, where the numbers may be different) is, as it can be defined by the grammar $G_3$ with $N = \{S, A, B\}$, $\Sigma = \{a, b\}$, $S$ the start symbol, and the following production rules:

1. $S \rightarrow aA$
2. $A \rightarrow aA$
3. $A \rightarrow bB$
4. $B \rightarrow bB$
5. $B \rightarrow \epsilon$

All languages generated by a regular grammar can be recognized in linear time by a finite state machine. Although, in practice, regular grammars are commonly expressed using regular expressions, some forms of regular expression used in practice do not strictly generate the regular languages and do not show linear recognitional performance due to those deviations.

### Self Assessment

State whether the following statements are true or false:

4. A grammar is a set of production rules for strings in a formal language.
5. Grammar is the process of recognizing an utterance by breaking it down to a set of symbols and analyzing each one against the grammar of the language.
6. A production rule is applied to a string by replacing one occurrence of its right-hand side in the string by its left-hand side.

## 12.3 Basic Parsing Techniques

'Parsing' is the term used to describe the process of automatically building syntactic analyses of a sentence in terms of a given grammar and lexicon. The resulting syntactic analyses may be used as input to a process of semantic interpretation, (or perhaps phonological interpretation, where aspects of this, like prosody, are sensitive to syntactic structure). Occasionally, 'parsing' is also used to include both syntactic and semantic analysis. We use it in the more conservative sense here, however. In most contemporary grammatical formalisms, the output of parsing is something logically equivalent to a tree, displaying dominance and precedence relations between constituents of a sentence, perhaps with further annotations in the form of attribute-value equations ('features') capturing other aspects of linguistic description. However, there are many different possible linguistic formalisms, and many ways of representing each of them, and hence many different ways of representing the results of parsing. We shall assume here a simple tree representation, and an underlying context-free grammatical (CFG) formalism. However, all of the algorithms described here can usually be used for more powerful unification based formalisms, provided these retain a context-free 'backbone', although in these cases their

Notes

complexity and termination properties may be different. Parsing algorithms are usually designed for classes of grammar rather than tailored towards individual grammars.

*Notes* There are several important properties that a parsing algorithm should have if it is to be practically useful. It should be 'sound' with respect to a given grammar and lexicon; that is, it should not assign to an input sentence analyses which cannot arise from the grammar in question.

It should also be 'complete'; that it should assign to an input sentence all the analyses it can have with respect to the current grammar and lexicon. Ideally, the algorithm should also be 'efficient', entailing the minimum of computational work consistent with fulfilling the first two requirements, and 'robust': behaving in a reasonably sensible way when presented with a sentence that it is unable to fully analyse successfully. In this discussion, we generally ignore the latter two requirements: to some extent, they are covered by the companion section on 'chart parsing'. There are several other dimensions on which is useful to characterise the behaviour of parsing algorithms. One can characterise their search strategy in terms of the characteristic alternatives of depth-first or breadth-first (q.v.). Orthogonally, one can characterise them in terms of the direction in which a structure is built: from the words upwards ('bottom up'), or from the root node downwards ('top down'). A third dimension is in terms of the sequential processing of input words: usually this is left-to-right, but right-to-left or 'middle-out' strategies are also feasible and may be preferable in some applications (e.g. parsing the output of a speech recogniser).

### 12.3.1 Top Down Parsing

We can illustrate a simple top down algorithm for a CFG as follows. Let the grammar contain the following rules (which for simplicity also introduce lexical entries instead of presupposing a separate component for this):

S → NP VP

NP → they | fish

VP → Aux VP

VP → Vi

VP → Vt NP

Aux → can

Vi → fish

Vt → can

This will assign to the sentence 'they can sh' two distinct analyses corresponding to two interpretations 'they are able/permitted to sh' and 'they put sh in cans'. (It will also generate several other good sentences and lots of odd ones). The top down algorithm is very simple. We begin with the start symbol, in this case, S, and see what rules it figures in as the mother. Then we look at the daughters of these rules and see whether the first one matches the next word in the input. If it does, we do the same for the remaining daughters. If not, we go through the same loop again, this time matching the daughters of the rules already found with mothers of other rules. The algorithm is easily illustrated on the input sentence 'they sh': the 'dot' precedes the daughter of the rule we are currently looking at, and the level of indentation indicates which rule has invoked the current action.

| Rule | Input | Notes |
|---|---|---|
| S → .NP VP | they fish | |
| NP → .they | they fish | |
| NP → they. | fish | |
| S → NP .VP | fish | |
| VP → .Vi | fish | |
| Vi → .fish | fish | |
| Vi → fish. | | |
| VP → Vi. | | |
| S → NP VP. | | |

Of course, in parsing this sentence we have magically chosen the correct rule at each point, and ignored choices which would not lead to a successful parse. To fully specify the algorithm we would need to eliminate all this magic and provide an exhaustive mechanism for trying out all possibilities. We can do this by non-deterministically trying out all alternatives at each point as they arise, or, more flexibly, by casting our algorithm in the form of a set of operations on representations that encode the state of a parse. Each parsing step takes an 'item', as we shall call them, and produces a set of new items. These alternatives can be pursued in whatever order is convenient. Parsing proceeds from an initial seed item and ends when no more steps are possible. Each remaining item, if there are any, will then represent a successful parse of the input. An item consists of a list of 'dotted trees' with the most recent to the right, and a list of unconsumed words. 'Dotted trees' are derived from dotted rules in an obvious way (a rule like S -¿ NP .VP is represented [S NP .VP]) and represent the partial analyses built so far. The list of words is the input remaining to be parsed. There are three basic operations that can be performed on an item:

(i) If there is a word after a dot in the most recent tree that matches the next word in the input, make a new item like the current one except that the dot follows the word, and the next word is removed from the input. For example, an item of the form. < ... [NP .they], [they can fish]> will yield a new item:

< ... [NP they.], [can fish] >

(ii) If the most recent tree has the dot at the end of its daughters, integrate it with the tree to its left, if there is one. So an item like:

<[S .NP VP], [NP they.], [can fish]>

yields a new one:

<[S [NP they] .VP], [can fish]>

(iii) If there is a rule whose left hand side matches the category following the dot in the most recent tree, make a new item like the old one with the addition of a new tree derived from the rules, with the dot preceding its daughters.

An item like

<[S .NP VP], [they can fish]>

yields one like:

<[S .NP VP], [NP .they], [they can fish]>

Notes

The sequence of items corresponding to the sample parse of 'they sh' above begins:

0. <[.S], [they fish]>

1. < [S .NP VP], [they fish]>

2. < [S .NP VP], [NP .they], [they fish]>

3. < [S .NP VP], [NP .fish], [they fish]>

4. < [S .NP VP], [NP they.], [ fish]>

5. < [S [NP they] .VP], [ fish]>

6. < [S [NP they] .VP], [VP .Vi], [ fish]>

7. < [S [NP they] .VP], [VP .Vt NP], [ fish]>

etc.

Notice that both 2 and 3 were produced from 1, but that 3 was discarded because none of the actions applied to it. The order in which new items are processed can be varied in order to give depth first or breadth first search behaviour. Depending on the search strategy followed, the top down algorithm may go into a loop when certain types of rule are found in the grammar. For example, if we added rules for more complex types of NP, we would encounter 'left recursion' of the category NP:

NP -> NP 's N:possessive NPs like [[John] 's sister]

Now from an item like:

< S -> .NP VP ....>

we would produce one like

< NP -> .NP 's N ....>

and this would in turn produce another

< NP -> .NP 's N ....>

and so on. There are two solutions: either rewrite the grammar to eliminate left recursion, which is always possible in principle, or add an extra step to the algorithm to check for repeated items. Some grammars will potentially send any sound and complete parsing algorithm into a loop: namely, those that contain cycles of the form A -> ..... -> A, where some symbol exhaustively dominates itself. This is not necessarily a bug in the parsing algorithms: such grammars will assign an infinite set of parse trees to any structure containing an A. Under these circumstances, the parser is merely trying to do what it is supposed to do and find all possible parses.

### 12.3.2 Bottom Up Parsing

The operation of a bottom up algorithm for CFG can be illustrated by the following sequence of operations for 'they sh': Structure Input so far remaining

[they fish]

[NP they] [fish]

[NP they][Vi fish] []

[NP they][Vp [Vi fish]] []

[S [NP they][Vp [Vi fish]]] []

Notes

We proceed by matching words to the right hand sides of rules, and then matching the resulting symbols, or sequences of symbols, to the right hand sides of rules until we have an S covering the whole sentence. Again, we have magically made the right choices at each point. We can provide a complete algorithm for one type of left-to-right, bottom up parsing procedure in terms of operations on items like those used in the top down parser. (Since we shall only be dealing with completely parsed constituents we can dispense with the dots.)

(i) If there is a rule whose right hand side matches the next word in the input, create a new item with a new tree made from that rule on the end of the tree list, and the remainder of the input.

*Example:*

<[], [they fish]>

becomes:

<[NP they], [fish] >

(ii) If there is a rule with n daughters matching the n most recent trees on the list, in order, create a new item which combines those daughters into an instance of the mother:

< [NP they], [VP [Vi fish]], []>

becomes:

< [S [NP they][VP [Vi fish]]], []>

A successful parse is represented by an item with no more input and a single S rooted tree on the list.

The sequence of items produced by this method in parsing 'they sh' is:

| | | |
|---|---|---|
| 1. | [] | [they fish] |
| 2. | [NP they], | [fish] |
| 3. | [NP they], [NP fish] | [] |
| 4. | [NP they], [Vi fish] | [] |
| 5. | [NP they], [VP [Vi fish]] | [] |
| 6. | [S [NP they] [VP [Vi fish]]] | [] |

Nothing can happen to item 3 and it is discarded. This particular type of bottom up algorithm is known as a 'shift-reduce' parser. Operation (i) shifts a word from the input on to the list of trees, which is operating like a 'stack' (a last-in-first-out store) insofar as it is only possible to get at the most recent items. Operation (ii) 'reduces' the list of trees by combining several of them from the top of the stack into one constituent. A generalization of this type of algorithm is familiar from computer science: the LR(k) family can be seen as shift-reduce algorithms with a certain amount ('k' words) of look ahead to determine, for a set of possible states of the parser, which action to take. The sequence of actions from a given grammar can be pre computed to give a 'parsing table' saying whether a shift or reduce is to be performed, and which state to go to next. The use of this technique for natural language parsing has been promoted by Tomita (1987), among others.

While in general, bottom up algorithms are more efficient than top down algorithms, one particular phenomenon that they deal with only clumsily are 'empty rules': rules in which the right hand side is the empty string. Such rules are theoretically dispensable in a context free

Notes

framework (any grammar containing them can in principle be converted to one, usually much larger, recognizing the same language which does not) but they are practically very useful in describing, for example, movement phenomena, in a reasonably concise way. Bottom up parsers find instances of such rules applied at every possible point in the input which can lead to much wasted effort. A better parsing algorithm than either pure top down or pure bottom up can be got by combining features of both. Operating bottom up means that the process is guided by what is actually in the input, rather than what the grammar predicts might be there (which will give a very large number of possibilities for a realistic grammar and lexicon). But imposing some top down constraints means that we can use what we have already found, with our knowledge about possible grammatical structures, to guide the search for what follows.

One such combined algorithm is the following. It is a member of the family of 'left-corner' parsing algorithms, since it begins at the left corner of the constituent it is trying to build, when viewed pictorially:

Mother

/\

/\

'Left Corner' .....

It is also a 'predictive' parser, in that it uses grammatical knowledge to predict what should come next, given what it has found already.

To describe this left-corner predictive parsing algorithm we extend our notion of an 'item' to be:

<CurrentConstituent, RemainingInput, MotherCatSought,

DaughtersFound, DaughtersSought, StackOfIncompleteConstituents>

There are four operations creating new items from old:

- ***'Shift'*** is as before, taking the next word from RemainingInput and making it (suitably labelled) the CurrentConstituent.
- ***'Predict'*** finds a rule whose leftmost daughter matches the CurrentConstituent, and then fills the MotherCatSought, DaughtersFound, and DaughtersSought fields of the new item as appropriate, copying these parts of the old item onto the StackOfIncompleteConstituents.
- ***'Match'*** applies when the CurrentConstituent is the next item in DaughtersSought: the new item has a null CurrentConstituent, and undergoes appropriate changes to the two Daughters fields.
- ***'Reduce'*** applies when there are no more DaughtersSought and creates an item with a new CurrentConstituent built from the MotherCatSought and the DaughtersFound, popping the most recent StackOfIncompleteConstituents entry to form the new Mother and Daughters fields.

With the Predict operation defined as above the algorithm will be sound and complete. However, it will derive no benefit from the top down information it carries, for that is not yet being used to constrain parsing hypotheses. What we need to do is to check that the mother categories on the rules found by Predict can be a 'left corner' of the current MotherCatSought, i.e. can be the leftmost daughter of some tree dominated by the MotherCat. This information can be calculated from the grammar fairly easily: we will not show how to do this here but for the grammar given above the 'left-corner-of' relation is:

<NP,S>

<Aux,VP>

**Notes**

<Vi,VP>

<Vt,VP>

We illustrate with the more complex and ambiguous sentence 'they can sh', assuming that Predict uses a left-corner check to filter candidate rules. For purposes of illustration, we will expand the grammar with an extra rule Sigma -> S, where Sigma, by convention, is the start symbol. Thus the full left-corner-of relation is:

<NP,Sigma>

8<NP,S>

<Aux,VP>

<Vi,VP>

<Vt,VP>

The initial item is based on the rule expanding the start category Sigma:

| Operation | Current | Input | Mother | Found | Sought | Stack |
|---|---|---|---|---|---|---|
| - | - | they can fish | Sigma | [] | [S] | - |
| 1. Shift: | [NP they] | can fish | Sigma | [] | [S] | - |
| 2. Predict 1: | - | can fish | S | [NP they] | [VP] | <Sigma, [], [S]> |
| 3. Shift 2: | [Vt can] | fish | S | [NP they] | [VP] | <Sigma, [], [S]> |
| 4. Shift 2: | [Aux can] | fish | S | [NP they] | [VP] | <Sigma, [], [S]> |
| 5. Predict 3: | - | fish | VP | [Vt can] | [NP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 6. Predict 4: | - | fish | VP | [Aux can] | [VP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 7. Shift 6: | [NP fish] | - | VP | [Aux can] | [VP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 8. Shift 6: | [Vi fish] | - | VP | [Aux can] | [VP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 9. Shift 5: | [NP fish] | - | VP | [Vt can] | [NP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 10. Shift 5: | [Vi fish] | - | VP | [Vt can] | [NP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 11. Predict 8: | - | - | VP | [Vi fish] | [] | <VP,[Aux can],[VP]><br><S,[NP they],[VP]><br><Sigma, [], [S]> |
| 12. Reduce 11: | [VP [Vi fish]] | - | VP | [Aux can] | [VP] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 13. Match 12: | - | | VP | [Aux can],<br>[VP [Vi fish]] | [] | <S,[NP they],[VP]><br><Sigma, [], [S]> |
| 14. Reduce 13: | [VP can [VP fish]] | | S | [NP they] | [VP] | <Sigma, [], [S]> |
| 15. Match 14: | - | | S | [NP they],<br>[VP can [VP fish]] | [] | <Sigma, [], [S]> |
| 16. Reduce 15: | [S they can fish] | | Sigma | [] | [S] | - |
| 17. Match 16: | - | | Sigma | [S they can fish] | [] | - |

*Contd...*

Notes

| | | | | |
|---|---|---|---|---|
| 18. Reduce 17 [Sigma [S they can fish]] | - | - | - | - |
| 19. Match 9: - | VP | [Vt can], [NP fish] | [] | <S,[NP they],[VP]> <Sigma, [], [S]> |
| 20. Reduce 19: [VP can [NP fish]] | S | [NP they] | [VP] | <Sigma, [], [S]> |
| 21. Match 20: - | S | [NP they], [VP can [NP fish] | [] | <Sigma, [], [S]> |
| 22. Reduce 21: [S they can fish] | Sigma | [] | [S] | - |
| 23. Match 22: - | Sigma | [S they can fish] | [] | - |
| 24. Reduce 23: [Sigma [S they can fish]] | - | - | - | - |

***Source:*** http://www.cs.ox.ac.uk/files/219/parsing.pdf

Notice that the ?ltering done by Predict eliminates several possible items based on the S -> NP VP rule at the point where the NP '?sh' is the current constituent (items 7 and 9), because S is not a left corner of VP, which is the current MotherCatSought. Items marked * lead to parsing paths that do not go anywhere: these too could be ruled out by doing a left corner check when Shifting: if the category of the item shifted is not a left corner of the ?rst item in DaughtersSought, then no parse will result.

*Caution* Each tagger has its meaning and importance.

*Task* Read parts of speech (POS) tagger and their meaning.

### Self Assessment

State whether the following statements are true or false:

7. Parsing algorithms are usually designed for classes of grammar rather than tailored towards individual grammars.
8. There is only one dimension on which is useful to characterise the behaviour of parsing algorithms.
9. A successful parse is represented by an item with no more input and a single S rooted tree on the list.

## 12.4 Semantic Analysis and Representation Structures

Computers are very fast and powerful machines, however, they process texts written by humans in an entirely mindless way, treating them merely as sequences of meaningless symbols. The main goal of language analysis is to obtain a suitable representation of text structure and thus make it possible to process texts based on their content. This is necessary in various applications, such as spell- and grammar-checkers, intelligent search engines, text summarization, or dialogue systems.

Natural language text can be analyzed on various levels, depending on the actual application setting. With regard to automatic processing of language data, the following analysis levels can be distinguished: Morphological analysis gives a basic insight into natural language by studying how to distinguish and generate grammatical forms of words arising through inflection

Notes

(i.e., declension and conjugation). This involves considering a set of tags describing grammatical categories of the word form concerned, most notably, its base form (lemma) and paradigm. Automatic analysis of word forms in free text can be used for instance in grammar checker development, and can aid corpus tagging, or semi-automatic dictionary compiling.

### 12.4.1 Semantic Analysis

Semantic and pragmatic analysis make up the most complex phase of language processing as they build up on results of all the above mentioned disciplines. Based on the knowledge about the structure of words and sentences, the meaning of words, phrases, sentences and texts is stipulated, and subsequently also their purpose and consequences. From the computational point of view, no general solutions that would be adequate have been proposed for this area. There are many open theoretical problems, and in practice, great problems are caused by errors on lower processing levels. The ultimate touchstone on this level is machine translation, which hasn't been implemented for Czech with satisfactory results yet.

One of the long-term projects of the NLP laboratory is the use of Transparent Intensional Logic (TIL) as a semantic representation of knowledge and subsequently as a transfer language in automatic machine translation. At the current stage, it is realistic to process knowledge in a simpler form – considerably less complex tasks have been addressed, such as machine translation for a restricted domain (e.g., official documents and weather reports), or semi-automatic machine translation between close languages.

*Caution* The resources exploited in these applications are corpora, semantic nets, and electronic dictionaries.

In linguistics, c to the level of the writing as a whole, to their language-independent meanings. It also involves removing features specific to particular linguistic and cultural contexts, to the extent that such a project is possible. The elements of idiom and figurative speech, being cultural, are often also converted into relatively invariant meanings in semantic analysis.

### 12.4.2 Syntactic Analysis

Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. The term "parsing" comes from Latin "pars", meaning part (of speech).

The term has slightly different meanings in different branches of linguistics and computer science. Traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence, sometimes with the aid of devices such as sentence diagrams. It usually emphasizes the importance of grammatical divisions such as subject and predicate.

Within computational linguistics the term is used to refer to the formal analysis by computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information.

*Did u know?* The term "syntactic analysis" is also used in psycholinguistics when describing language comprehension. In this context, parsing refers to the way that human beings analyze a sentence or phrase (in spoken language or text) "in terms of grammatical constituents, identifying the parts of speech, syntactic relations, etc." This term is especially common when discussing what linguistic cues help speakers to interpret garden-path sentences.

Notes

Within computer science, the term is used in the analysis of computer languages, referring to the syntactic analysis of the input code into its component parts in order to facilitate the writing of compilers and interpreters.

### 12.4.3 Knowledge Representation (KR)

A knowledge representation is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it. It is a fragmentary theory of intelligent reasoning, expressed in terms of three components:

(i) the representation's fundamental conception of intelligent reasoning;

(ii) the set of inferences the representation sanctions; and

(iii) the set of inferences it recommends.

It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences. It is a medium of human expression, i.e., a language in which we say things about the world. Understanding the roles and acknowledging their diversity has several useful consequences. Firstly, each role requires something slightly different from a representation; each accordingly leads to an interesting and different set of properties we want a representation to have. Secondly, we believe the roles provide a framework useful for characterizing a wide variety of representations. We suggest that the fundamental "mindset" of a representation can be captured by understanding how it views each of the roles, and that doing so reveals essential similarities and differences.

Thirdly, we believe that some previous disagreements about representation are usefully disentangled when all five roles are given appropriate consideration. We demonstrate this by revisiting and dissecting the early arguments concerning frames and logic.

Finally, we believe that viewing representations in this way has consequences for both research and practice. For research, this view provides one direct answer to a question of fundamental significance in the field. It also suggests adopting a broad perspective on what's important about a representation, and it makes the case that one significant part of the representation endeavor – capturing and representing the richness of the natural world – is receiving insufficient attention. We believe this view can also improve practice by reminding practitioners about the inspirations that are the important sources of power for a variety of representations.

### Self Assessment

State whether the following statements are true or false:

10. Computers are very fast and powerful machines.

11. Semantic and pragmatic analysis make up the most simple phase of language processing as they build up on results of all the above mentioned disciplines.

12. The term parsing comes from Latin pars, meaning part (of speech).

## 12.5 Natural Language Generalization

Generalization and memory are part of natural language understanding. The generalization process includes the retrieval of relevant examples from long-term memory so that the concepts

Notes

to be created can be determined when new stories are read. Generalizations are analyzed as to their critical parts and evaluated in light of later evidence. The knowledge structures used and a number of examples of the system in operation are presented.

### 12.5.1 Natural Language Systems

The Natural Language System is a software system designed to answer questions that are posed to it in natural language. START parses incoming questions, matches the queries created from the parse trees against its knowledge base and presents the appropriate information segments to the user. In this way, START provides untrained users with speedy access to knowledge that in many cases would take an expert some time to find.

START (SynTactic Analysis using Reversible Transformations) was developed by Boris Katz at MIT's Artificial Intelligence Laboratory. Currently, the system is undergoing further development by the InfoLab Group, led by Boris Katz. This system was first connected to the World Wide Web in December, 1993, and in its several forms has to date answered millions of questions from users around the world.

A key technique called "natural language annotation" helps to connect information seekers to information sources. This technique employs natural language sentences and phrases – annotations – as descriptions of content that are associated with information segments at various granularities. An information segment is retrieved when its annotation matches an input question. This method allows this system to handle all variety of media, including text, diagrams, images, video and audio clips, data sets, Web pages, and others.

The natural language processing component of this system consists of two modules that share the same grammar. The understanding module analyzes English text and produces a knowledge base that encodes information found in the text. Given an appropriate segment of the knowledge base, the generating module produces English sentences. Used in conjunction with the technique of natural language annotation, these modules put the power of sentence-level natural language processing to use in the service of multimedia information access.

### 12.5.2 Recognition and Classification Process

It is generally easy for a person to differentiate the sound of a human voice, from that of a violin; a handwritten numeral "3," from an "8"; and the aroma of a rose, from that of an onion. However, it is difficult for a programmable computer to solve these kinds of perceptual problems. These problems are difficult because each pattern usually contains a large amount of information, and the recognition problems typically have an inconspicuous, high-dimensional, structure.

Pattern recognition is the science of making inferences from perceptual data, using tools from statistics, probability, computational geometry, machine learning, signal processing, and algorithm design. Thus, it is of central importance to artificial intelligence and computer vision, and has far-reaching applications in engineering, science, medicine, and business. In particular, advances made during the last half century, now allow computers to interact more effectively with humans and the natural world (e.g., speech recognition software). However, the most important problems in pattern recognition are yet to be solved.

It is natural that we should seek to design and build machines that can recognize patterns. From automated speech recognition, fingerprint identification, optical character recognition, DNA sequence identification, and much more, it is clear that reliable, accurate pattern recognition by machine would be immensely useful. Moreover, in solving the indefinite number of problems required to build such systems, we gain deeper understanding and appreciation for pattern recognition systems. For some problems, such as speech and visual recognition, our design

Notes

efforts may in fact be influenced by knowledge of how these are solved in nature, both in the algorithms we employ and in the design of special-purpose hardware. Feature can be defined as any distinctive aspect, quality or characteristic which, may be symbolic (i.e., color) or numeric (i.e., height). The combination of d features is represented as a d-dimensional column vector called a feature vector. The d-dimensional space defined by the feature vector is called feature space. Objects are represented as points in feature space. This representation is called a scatter plot.

Pattern is defined as composite of features that are characteristic of an individual. In classification, a pattern is a pair of variables {x,w} where x is a collection of observations or features (feature vector) and w is the concept behind the observation (label). The quality of a feature vector is related to its ability to discriminate examples from different classes.

*Did u know?* Here are two fundamental approaches for implementing a pattern recognition system: statistical and structural. Each approach employs different techniques to implement the description and classification tasks. Hybrid approaches, sometimes referred to as a unified approach to pattern recognition, combine both statistical and structural techniques within a pattern recognition system.

Statistical pattern recognition draws from established concepts in statistical decision theory to discriminate among data from different groups based upon quantitative features of the data. There are a wide variety of statistical techniques that can be used within the description task for feature extraction, ranging from simple descriptive statistics to complex transformations.

*Examples:* Of statistical feature extraction techniques include mean and standard deviation computations, frequency count summarizations, Karhunen-Lóeve transformations, Fourier transformations, wavelet transformations, and Hough transformations.

The quantitative features extracted from each object for statistical pattern recognition are organized into a fixed length feature vector where the meaning associated with each feature is determined by its position within the vector (i.e., the first feature describes a particular characteristic of the data, the second feature describes another characteristic, and so on). The collection of feature vectors generated by the description task are passed to the classification task. Statistical techniques used as classifiers within the classification task include those based on similarity (e.g., template matching, k-nearest neighbor), probability (e.g., Bayes rule), boundaries (e.g., decision trees, neural networks), and clustering (e.g., k-means, hierarchical).

The quantitative nature of statistical pattern recognition makes it difficult to discriminate (observe a difference) among groups based on the morphological (i.e., shape based or structural) subpatterns and their interrelationships embedded within the data. This limitation provided the impetus for the development of a structural approach to pattern recognition that is supported by psychological evidence pertaining to the functioning of human perception and cognition. Object recognition in humans has been demonstrated to involve mental representations of explicit, structure-oriented characteristics of objects, and human classification decisions have been shown to be made on the basis of the degree of similarity between the extracted features and those of a prototype developed for each group. For instance, the recognition by components theory explains the process of pattern recognition in humans: (1) the object is segmented into separate regions according to edges defined by differences in surface characteristics (e.g., luminance, texture, and color), (2) each segmented region is approximated by a simple geometric shape, and (3) the object is identified based upon the similarity in composition between the geometric representation of the object and the central tendency of each group. This theorized functioning of human perception and cognition serves as the foundation for the structural approach to pattern recognition.

Notes

Structural pattern recognition, sometimes referred to as syntactic pattern recognition due to its origins in formal language theory, relies on syntactic grammars to discriminate among data from different groups based upon the morphological interrelationships (or interconnections) present within the data. Structural features, often referred to as primitives, represent the subpatterns (or building blocks) and the relationships among them which constitute the data. The semantics associated with each feature are determined by the coding scheme (i.e., the selection of morphologies) used to identify primitives in the data. Feature vectors generated by structural pattern recognition systems contain a variable number of features (one for each primitive extracted from the data) in order to accommodate the presence of superfluous structures which have no impact on classification. Since the interrelationships among the extracted primitives must also be encoded, the feature vector must either include additional features describing the relationships among primitives or take an alternate form, such as a relational graph, that can be parsed by a syntactic grammar.

The emphasis on relationships within data makes a structural approach to pattern recognition most sensible for data which contain an inherent, identifiable organization such as image data (which is organized by location within a visual rendering) and time-series data (which is organized by time); data composed of independent samples of quantitative measurements, lack ordering and require a statistical approach. Methodologies used to extract structural features from image data such as morphological image processing techniques result in primitives such as edges, curves, and regions; feature extraction techniques for time-series data include chain codes, piecewise linear regression, and curve fitting which are used to generate primitives that encode sequential, time-ordered relationships. The classification task arrives at an identification using parsing: the extracted structural features are identified as being representative of a particular group if they can be successfully parsed by a syntactic grammar. When discriminating among more than two groups, a syntactic grammar is necessary for each group and the classifier must be extended with an adjudication scheme so as to resolve multiple successful parsings.

*Task* Analyze the approaches for implementing a pattern recognition system.

## Self Assessment

State whether the following statements are true or false:

13. Generalization and memory are part of natural language understanding.
14. The natural language processing component of this system consists of four modules that share the same grammar.
15. Structural pattern recognition sometimes referred to as syntactic pattern recognition.

## 12.6 Summary

- Natural Language Processing (NLP) is the computerized approach to analyzing text that is based on both a set of theories and a set of technologies. And, being a very active area of research and development.
- Modern NLP algorithms are grounded in machine learning, especially statistical machine learning.
- Computational linguistics might be considered as a synonym of automatic processing of natural language, since the main task of computational linguistics is just the construction of computer programs to process words and texts in natural language.

Notes

- Regular grammar was first introduced for studying the properties of neural nets. The languages represented by regular expressions are called the regular languages.
- Semantic and pragmatic analysis make up the most complex phase of language processing as they build up on results of all the above mentioned disciplines.
- The goal of syntactic analysis is to determine whether the text string on Input is a sentence in the given (natural) language. If it is, the result of the analysis contains a description of the syntactic structure of the sentence.

## 12.7 Keywords

***Context-sensitive Grammar:*** It is a formal grammar in which left-hand sides and right hand dies of any production (rewrite) rules may be surrounded by a context of terminal and non terminal symbols.

***Linguistics:*** It is the study of human languages. It follows scientific approach. So it is also referred to as linguistic science.

***NLP:*** It stands Natural Language Processing. NLP is a sub field of Artificial intelligence (AI) Linguistic, devoted to make computer " understand" statements written in human languages.

***Parsing:*** It is a process to determine how a string might be derived using productions (rewrite rules) of a given grammar.

***Semantic Network:*** It is a structure for representing knowledge as a pattern of interconnected nodes and arcs, in a way that the nodes represent concepts of entities, attributes, events and states; and the arcs represent the connections among the concepts.

## 12.8 Review Questions

1. Explain natural language processing.
2. Define linguistics. What is computational linguistics?
3. What is formal language?
4. Explain the role of parsing.
5. Discuss recognition and classification process.
6. Explain natural language systems.
7. Define knowledge representation.
8. What do you understand by syntactic analysis?

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | False | 4. | True |
| 5. | False | 6. | False |
| 7. | True | 8. | False |
| 9. | True | 10. | True |
| 11. | False | 12. | True |

13. True
14. False
15. True

Notes

## 12.9 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://krchowdhary.com/me-nlp/nlp-01.pdf

http://ltrc.iiit.ac.in/downloads/nlpbook/nlp-panini.pdf

http://nltk.org/

http://skr.nlm.nih.gov/papers/references/aral96.pdf

http://www.cs.utexas.edu/~mooney/cs343/slide-handouts/nlp.pdf

http://www.cse.unt.edu/~rada/papers/mihalcea.cicling06a.pdf

http://www.webopedia.com/TERM/N/NLP.html

Notes

# Unit 13: Expert System Architecture

**CONTENTS**



## Objectives

After studying this unit, you will be able to:

- Describe Expert System Architecture
- Discuss Rule-based Systems
- Explain Non-production System Architectures
- Identify how to deal with Uncertainty and Change
- Discuss Knowledge Acquisition and Validation
- Explain distinguishing features of Expert Systems

Notes

## Introduction

In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, like an expert, and not by following the procedure of a developer as is the case in conventional programming. The first expert systems were created in the 1970s and then proliferated in the 1980s. An expert system has a unique structure, different from traditional computer programming. It is divided into two parts, one fixed, independent of the expert system: the inference engine, and one variable: the knowledge base. To run an expert system, the engine reasons about the knowledge base like a human. In the 80s, a third part appeared: a dialog interface to communicate with users. This ability to conduct a conversation with users was later called "conversational".

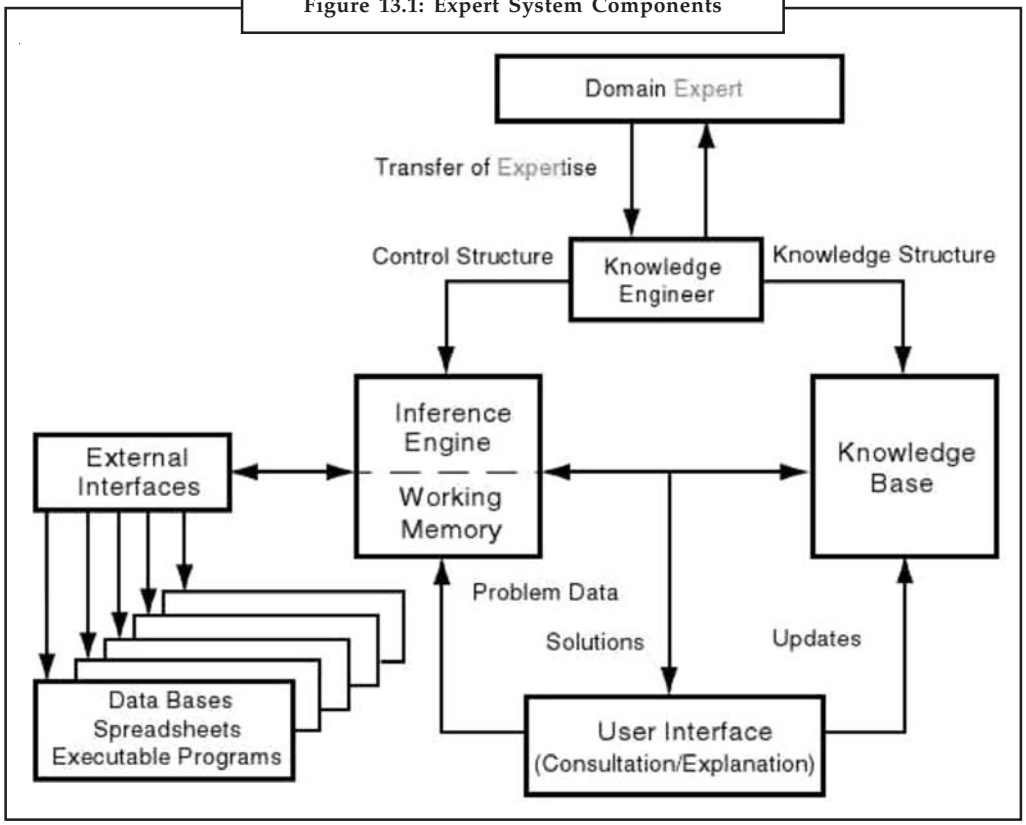
## 13.1 Expert System Architecture

Expert systems typically contain the following four components:

- Knowledge-Acquisition Interface
- User Interface
- Knowledge Base
- Inference Engine

This architecture differs considerably from traditional computer programs, resulting in several characteristics of expert systems.

**Figure 13.1: Expert System Components**



Notes

## Knowledge-Acquisition Interface

The knowledge-acquisition interface controls how the expert and knowledge engineer interact with the program to incorporate knowledge into the knowledge base. It includes features to assist experts in expressing their knowledge in a form suitable for reasoning by the computer. This process of expressing knowledge in the knowledge base is called knowledge acquisition.

*Notes* Knowledge acquisition turns out to be quite difficult in many cases – so difficult that some authors refer to the knowledge acquisition bottleneck to indicate that it is this aspect of expert system development which often requires the most time and effort.

Debugging faulty knowledge bases is facilitated by traces (lists of rules in the order they were fired), probes (commands to find and edit specific rules, facts, and so on), and bookkeeping functions and indexes (which keep track of various features of the knowledge base such as variables and rules).

*Notes* Some rule-based expert system shells for personal computers monitor data entry, checking the syntactic validity of rules.

Expert systems are typically validated by testing their predictions for several cases against those of human experts. Case facilities – permitting a file of such cases to be stored and automatically evaluated after the program is revised – can greatly speed the validation process. Many features that are useful for the user interface, such as on-screen help and explanations, are also of benefit to the developer of expert systems and are also part of knowledge-acquisition interfaces. Expert systems in the literature demonstrate a wide range of modes of knowledge acquisition (Buchanan, 1985). Expert system shells on microcomputers typically require the user to either enter rules explicitly or enter several examples of cases with appropriate conclusions, from which the program will infer a rule.

## User Interface

The user interface is the part of the program that interacts with the user. It prompts the user for information required to solve a problem, displays conclusions, and explains its reasoning.

Features of the user interface often include:

- Doesn't ask "dumb" questions
- Explains its reasoning on request
- Provides documentation and references
- Defines technical terms
- Permits sensitivity analyses, simulations, and what-if analyses
- Detailed report of recommendations
- Justifies recommendations
- Online help
- Graphical displays of information
- Trace or step through reasoning

Notes

The user interface can be judged by how well it reproduces the kind of interaction one might expect between a human expert and someone consulting that expert.

### Knowledge Base

The knowledge base consists of specific knowledge about some substantive domain. A knowledge base differs from a data base in that the knowledge base includes both explicit knowledge and implicit knowledge. Much of the knowledge in the knowledge base is not stated explicitly, but inferred by the inference engine from explicit statements in the knowledge base. This makes knowledge bases have more efficient data storage than data bases and gives them the power to exhaustively represent all the knowledge implied by explicit statements of knowledge.

Knowledge bases can contain many different types of knowledge and the process of acquiring knowledge for the knowledge base (this is often called knowledge acquisition) often needs to be quite different depending on the type of knowledge sought.

### Types of Knowledge

There are many different kinds of knowledge considered in expert systems. Many of these form dimensions of contrasting knowledge:

- Explicit knowledge
- Implicit knowledge
- Domain knowledge
- Common sense or world knowledge
- Heuristics
- Algorithms
- Procedural knowledge
- Declarative or semantic knowledge
- Public knowledge
- Private knowledge
- Shallow knowledge
- Deep knowledge
- Meta knowledge

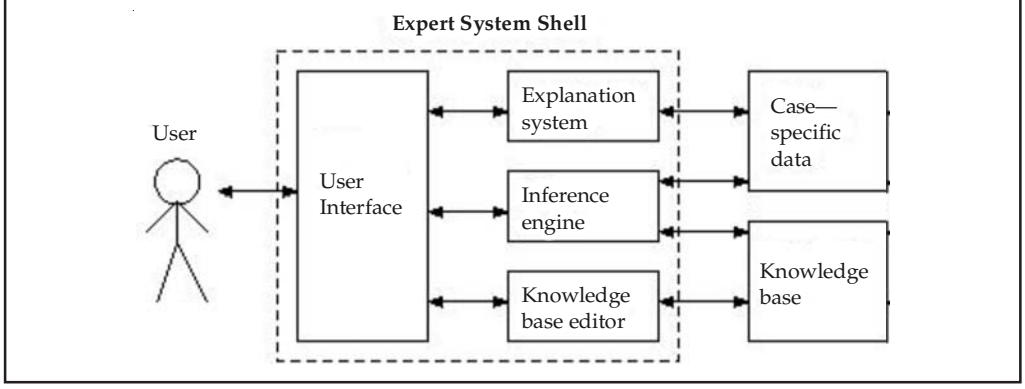
### Another View of Expert System Architecture

Following figure shows the most important modules that make up a rule-based expert system. The user interacts with the system through a *user interface* which may use menus, natural language or any other style of interaction). Then an *inference engine* is used to reason with both the *expert knowledge* (extracted from our friendly expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The *case specific data* includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in *working memory*.

Almost all expert systems also have an *explanation subsystem*, which allows the program to explain its reasoning to the user. Some systems also have a *knowledge base editor* which help the expert or knowledge engineer to easily update and check the knowledge base.

Notes

Figure 13.2: Expert System Shell




One important feature of expert systems is the way they (usually) separate domain specific knowledge from more general purpose reasoning and representation techniques. The general purpose bit (in the dotted box in the figure) is referred to as an *expert system shell*. As we see in the figure, the shell will provide the inference engine (and knowledge representation scheme), a user interface, an explanation system and sometimes a knowledge base editor. Given a new kind of problem to solve (say, car design), we can usually find a shell that provides the right sort of support for that problem, so all we need to do is provide the expert knowledge. There are numerous commercial expert system shells, each one appropriate for a slightly different range of problems. (Expert systems work in industry includes both writing expert system shells and writing expert systems using shells.)

*Caution* Using shells to write expert systems generally greatly reduces the cost and time of development (compared with writing the expert system from scratch).

### Self Assessment

State whether the following statements are true or false:

1. Expert systems typically contain the two components.
2. Expert system shells on microcomputers.
3. The user interface is the part of the program that interacts with the user.

## 13.2 Rule-based Systems

Conventional problem-solving computer programs make use of well-structured algorithms, data structures, and crisp reasoning strategies to find solutions. For the difficult problems with which expert systems are concerned, it may be more useful to employ heuristics: strategies that often lead to the correct solution, but that also sometimes fail. Conventional rule-based expert systems, use human expert knowledge to solve real-world problems that normally would require human intelligence. Expert knowledge is often represented in the form of *rules* or as *data* within the computer. Depending upon the problem requirement, these rules and data can be recalled to solve problems. Rule-based expert systems have played an important role in modern intelligent systems and their applications in strategic goal setting, planning, design, scheduling, fault monitoring, diagnosis and so on. With the technological advances made in the last decade, today's users can choose from dozens of commercial software packages having friendly graphic user interfaces (Ignizio, 1991).

Notes

Conventional computer programs perform tasks using a decision-making logic containing very little knowledge other than the basic algorithm for solving that specific problem. The basic knowledge is often embedded as part of the programming code, so that as the knowledge changes, the program has to be rebuilt.

*Did u know?* Knowledge-based expert systems collect the small fragments of human knowhow into a knowledge base, which is used to reason through a problem, using the knowledge that is appropriate.

An important advantage here is that within the domain of the knowledge base, a different problem can be solved using the same program without reprogramming efforts. Moreover, expert systems could explain the reasoning process and handle levels of confidence and uncertainty, which conventional algorithms do not handle (Giarratano and Riley, 1989). Some of the important advantages of expert systems are as follows:

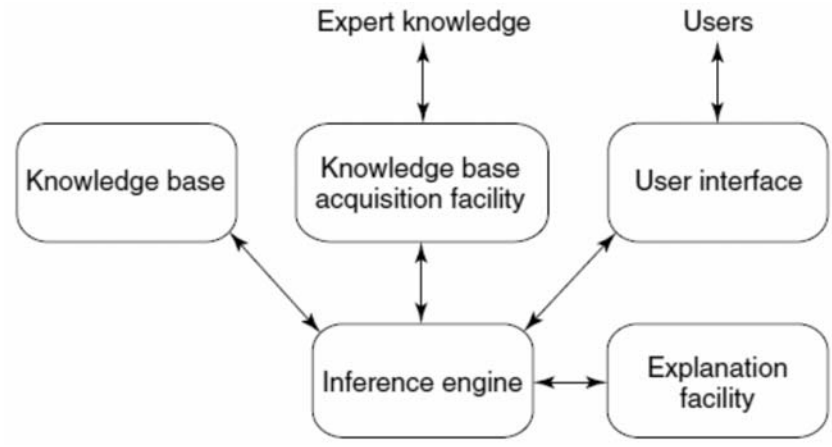
- ability to capture and preserve irreplaceable human experience;
- ability to develop a system more consistent than human experts;
- minimize human expertise needed at a number of locations at the same time (especially in a hostile environment that is dangerous to human health);
- solutions can be developed faster than human experts.

The basic components of an expert system are illustrated. The knowledge base stores all relevant information, data, rules, cases, and relationships used by the expert system. A knowledge base can combine the knowledge of multiple human experts. A rule is a conditional statement that links given conditions to actions or outcomes.

A frame is another approach used to capture and store knowledge in a knowledge base. It relates an object or item to various facts or values. A frame-based representation is ideally suited for object-oriented programming techniques. Expert systems making use of frames to store knowledge are also called *frame-based expert systems*.

The purpose of the inference engine is to seek information and relationships from the knowledge base and to provide answers, predictions, and suggestions in the way a human expert would. The inference engine must find the right facts, interpretations, and rules and assemble them correctly.

**Figure 13.3: Rule-based Systems**



Notes

Two types of inference methods are commonly used – Backward chaining is the process of starting with conclusions and working backward to the supporting facts. Forward chaining starts with the facts and works forward to the conclusions.

### 13.2.1 Forward Chaining (Data-driven Rule-based Expert Systems)

A rule-based system consists of *if-then rules*, a bunch of *facts*, and an *interpreter* controlling the application of the rules, given the facts. These *if-then* rule statements are used to formulate the conditional statements that comprise the complete knowledge base. A single *if-then* rule assumes the form 'if *x* is *A* then *y* is *B*' and the if-part of the rule '*x* is *A*' is called the *antecedent* or *premise*, while the then-part of the rule '*y* is *B*' is called the *consequent* or *conclusion*. There are two broad kinds of inference engines used in rule-based systems: *forward chaining* and *backward chaining* systems. In a *forward chaining* system, the initial facts are processed first, and keep using the rules to draw new conclusions given those facts. In a *backward chaining* system, the hypothesis (or solution/goal) we are trying to reach is processed first, and keep looking for rules that would allow to conclude that hypothesis. As the processing progresses, new subgoals are also set for validation. Forward chaining systems are primarily data-driven, while backward chaining systems are goal-driven. Consider an example with the following set of *if-then* rules

***Rule 1:*** If A and C then Y

***Rule 2:*** If A and X then Z

***Rule 3:*** If B then X

***Rule 4:*** If Z then D

If the task is to prove that D is true, given A and B are true. According to forward chaining, start with Rule 1 and go on downward till a rule that fires is found. Rule 3 is the only one that fires in the first iteration. After the first iteration, it can be concluded that A, B, and X are true. The second iteration uses this valuable information. After the second iteration, Rule 2 fires adding Z is true, which in turn helps Rule 4 to fire, proving that D is true. Forward chaining strategy is especially appropriate in situations where data are expensive to collect, but few in quantity. However, special care is to be taken when these rules are constructed.

### 13.2.2 Backward Chaining (Goal-driven Rule-based Expert Systems)

A rule-based system consists of if-then rules, a bunch of facts, and an interpreter controlling the application of the rules, given the facts. These if-then rule statements are used to formulate the conditional statements that comprise the complete knowledge base. A single if-then rule assumes the form 'if x is A then y is B' and the if-part of the rule 'x is A' is called the antecedent or premise, while the then-part of the rule 'y is B' is called the consequent or conclusion. There are two broad kinds of inference engines used in rule-based systems: forward chaining and backward chaining systems. In a forward chaining system, the initial facts are processed first, and keep using the rules to draw new conclusions given those facts. In a backward chaining system, the hypothesis (or solution/goal) we are trying to reach is processed first, and keep looking for rules that would allow to conclude that hypothesis. As the processing progresses, new subgoals are also set for validation. Forward chaining systems are primarily data-driven, while backward chaining systems are goal-driven. Consider an example with the following set of if-then rules

***Rule 1:*** If A and C then Y

***Rule 2:*** If A and X then Z

***Rule 3:*** If B then X

***Rule 4:*** If Z then D

Notes

If the task is to prove that D is true, given A and B are true. According to forward chaining, start with Rule 1 and go on downward till a rule that fires is found. Rule 3 is the only one that fires in the first iteration. After the first iteration, it can be concluded that A, B, and X are true. The second iteration uses this valuable information. After the second iteration, Rule 2 fires adding Z is true, which in turn helps Rule 4 to fire, proving that D is true. Forward chaining strategy is especially appropriate in situations where data are expensive to collect, but few in quantity. However, special care is to be taken when these rules are constructed.

### Self Assessment

State whether the following statements are true or false:

4. Expert knowledge is often represented in the form of *rules* or as *data* outside the computer.
5. Rule-based expert systems does not play an important role in modern intelligent systems and their applications.
6. The purpose of the inference engine is to seek information and relationships from the knowledge base.

## 13.3 Non-production System Architectures

A production system (or production rule system) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed productions, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system. Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be *triggered*. If a production's action is executed, it is said to have *fired*. A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter.

*Caution* The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered.

### 13.3.1 Associative or Semantic Networks

*Example:* This example shows a set of production rules for reversing a string from an alphabet that does not contain the symbols "$" and "*" (which are used as marker symbols).

```
P1: $$ -> *
P2: *$ -> *
P3: *x -> x*
P4: * -> null & halt
P5: $xy -> y$x
P6: null -> $
```

In this example, production rules are chosen for testing according to their order in this production list. For each rule, the input string is examined from left to right with a moving window to find a match with the LHS of the production rule. When a match is found, the matched substring in the input string is replaced with the RHS of the production rule. In this production system, x and

Notes

y are *variables* matching any character of the input string alphabet. Matching resumes with P1 once the replacement has been made.

The string "ABC", for instance, undergoes the following sequence of transformations under these production rules:

```
$ABC  (P6)
B$AC  (P5)
BC$A  (P5)
$BC$A  (P6)
C$B$A  (P5)
$C$B$A  (P6)
$$C$B$A  (P6)
*C$B$A  (P1)
C*$B$A  (P3)
C*B$A  (P2)
CB*$A  (P3)
CB*A  (P2)
CBA*  (P3)
CBA  (P4)
```

In such a simple system, the ordering of the production rules is crucial. Often, the lack of control structure makes production systems difficult to design. It is, of course, possible to add control structure to the production systems model, namely in the inference engine, or in the working memory.

### 13.3.2 Frame Architecture

In a toy simulation world where a monkey in a room can grab different objects and climb on others, an example production rule to grab an object suspended from the ceiling would look like:

```
(p  Holds::Object-Ceiling
   {(goal  ^status  active  ^type  holds  ^objid  <O1>)  <goal>}
   {(physical-object
      ^id  <O1>
      ^weight  light
      ^at  <p>
      ^on  ceiling)  <object-1>}
   {(physical-object  ^id  ladder  ^at  <p>  ^on  floor)  <object-2>}
   {(monkey  ^on  ladder  ^holds  NIL)  <monkey>}
   -(physical-object  ^on  <O1>)
  -- >
   (write  (crlf)  Grab  <O1>  (crlf))
   (modify  <object1>  ^on NIL)
   (modify  <monkey>  ^holds  <O1>)
   (modify  <goal>  ^status  satisfied)
)
```

In this example, data in working memory is structured and variables appear between angle brackets. The name of the data structure, such as "goal" and "physical-object", is the first literal in conditions; the fields of a structure are prefixed with "^". The "-" indicates a negative condition.

### Self Assessment

State whether the following statements are true or false:

7. A production system also contains a database called working memory.
8. The "-" indicates a positive condition.
9. The lack of control structure makes production systems difficult to design.

## 13.4 Semantic Memory

Notes

Semantic memory refers to the memory of meanings, understandings, and other concept-based knowledge, and underlies the conscious recollection of factual information and general knowledge about the world. Semantic and episodic memory together make up the category of declarative memory, which is one of the two major divisions in memory. With the use of our semantic memory we can give meaning to otherwise meaningless words and sentences. We can learn about new concepts by applying our knowledge learned from things in the past. The counterpart to declarative, or explicit memory, is procedural memory, or implicit memory. TLC is an instance of a more general class of models known as semantic networks. In a semantic network, each node is to be interpreted as representing a specific concept, word, or feature. That is, each node is a symbol. Semantic networks generally do not employ distributed representations for concepts, as may be found in a neural network. The defining feature of a semantic network is that its links are almost always directed (that is, they only point in one direction, from a base to a target) and the links come in many different types, each one standing for a particular relationship that can hold between any two nodes. Processing in a semantic network often takes the form of spreading activation.

*Notes* Semantic networks see the most use in models of discourse and logical comprehension, as well as in Artificial Intelligence. In these models, the nodes correspond to words or word stems and the links represent syntactic relations between them.

### 13.4.1 Feature Models

Feature models view semantic categories as being composed of relatively unstructured sets of features. The semantic feature-comparison model, proposed by Smith, Shoben, and Rips (1974), describes memory as being composed of feature lists for different concepts. According to this view, the relations between categories would not be directly retrieved, they would be indirectly computed. For example, subjects might verify a sentence by comparing the feature sets that represent its subject and predicate concepts. Such computational feature-comparison models include the ones proposed by Meyer (1970), Rips (1975), Smith, et al. (1974).

Early work in perceptual and conceptual categorization assumed that categories had critical features and that category membership could be determined by logical rules for the combination of features. More recent theories have accepted that categories may have an ill-defined or "fuzzy" structure and have proposed probabilistic or global similarity models for the verification of category membership.

### 13.4.2 Associative Models

The "association"—a relationship between two pieces of information—is a fundamental concept in psychology, and associations at various levels of mental representation are essential to models of memory and cognition in general. The set of associations among a collection of items in memory is equivalent to the links between nodes in a network, where each node corresponds to a unique item in memory. Indeed, neural networks and semantic networks may be characterized as associative models of cognition. However, associations are often more clearly represented as an N×N matrix, where N is the number of items in memory. Thus, each cell of the matrix corresponds to the strength of the association between the row item and the column item. Learning of associations is generally believed to be a Hebbian process; that is, whenever two items in memory are simultaneously active, the association between them grows stronger, and the more likely either item is to activate the other.

Notes

### Self Assessment

State whether the following statements are true or false:

10. Semantic and episodic memory together make up the category of declarative memory, which is one of the two major divisions in memory.
11. Processing in a semantic network often takes the form of spreading activation.
12. The set of associations among a collection of items in memory is equivalent to the links between nodes in a network.

## 13.5 Knowledge Acquisition and Validation

The frame contains information on how to use the frame, what to expect next, and what to do when these expectations are not met. Some information in the frame is generally unchanged while other information, stored in "terminals", usually change. Different frames may share the same terminals.

Each piece of information about a particular frame is held in a slot. The information can contain:

- Facts or Data
  - Values (called facets)
- Procedures (also called procedural attachments)
  - IF-NEEDED: deferred evaluation
  - IF-ADDED: updates linked information
- Default Values
  - For Data
  - For Procedures
- Other Frames or Subframes

### Neural Network

The term "neural network" was traditionally used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus, the term may refer to either biological neural networks, made up of real biological neurons, or artificial neural networks, for solving artificial intelligence problems.

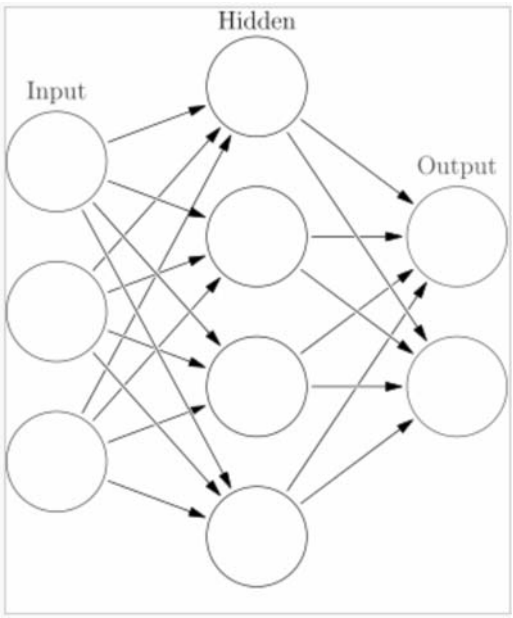
Unlike von Neumann model computations, artificial neural networks do not separate memory and processing and operate via the flow of signals through the net connections, somewhat akin to biological networks.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset.

The word "network" in the term 'artificial neural network' refers to the inter – connections between the neurons in the different layers of each system. An example system has three layers. The first layer has input neurons, which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons with some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations.

Notes

**Figure 13.4: Architecture**

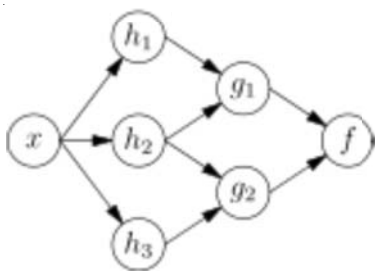



An ANN is typically defined by three types of parameters:

1. The interconnection pattern between different layers of neurons
2. The learning process for updating the weights of the interconnections
3. The activation function that converts a neuron's weighted input to its output activation.

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the *nonlinear weighted sum*, where $f(x) = K(\Sigma_i w_i g_i(x))$, where $K$ (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions $g_i$ as simply a vector $g = (g_1, g_2, \ldots, g_n)$.

**Figure 13.5: ANN Dependency Graph**




This Figure 13.5 depicts such a decomposition of $f$, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input $x$ is transformed into a 3-dimensional vector $h$, which is then transformed into a 2-dimensional vector $g$, which is finally transformed into $f$. This view is most commonly encountered in the context of optimization.

Notes

The second view is the probabilistic view: the random variable $F = f(G)$ depends upon the random variable $G = g(H)$, which depends upon $H = h(X)$, which depends upon the random variable $x$. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of $g$ are independent of each other given their input $h$). This naturally enables a degree of parallelism in the implementation.

## Dealing with Uncertainty and Change

### Fancy Logics

Dealing with uncertainty and change in important in AI because:

- The world changes, sometimes unpredictably, as a result of external actions (ie, actions that we don't perform ourself).
- Our beliefs about the world change (whether or not the world changes itself). If we get new evidence about something, a whole web of related beliefs may change.
- Our beliefs about the world may be uncertain. We may be unsure whether we have observed something correctly, or we may draw plausible but not certain inferences from our existing variably certain beliefs.

Dealing with such things isn't too hard to do in an *ad hoc* way. We can have rules that delete things from working memory when they change, and/or which have rules and facts with numerical certainty factors on them. What is hard is dealing with uncertainty in a principled way. First order predicate logic is inadequate as it is designed to work with information that is complete, consistent and *monotonic* (this basically means that facts only get added, not deleted from the set of things that are known to be true). There is no straightforward way of using it to deal with incomplete, variably certain, inconsistent and non-monotonic inferences. (We also cannot easily use it to explicitly represent *beliefs* about the world, for reasons that should become clear later.) We would like to have a formal, principled, and preferably simple basis for dealing with belief, uncertainty and change.

There are two main approaches to dealing with all this. The first is to use a fancier logic. It is important to remember that first order predicate logic is but one logic among many. New logics pop up every day, and most come with a clear, well defined semantics and proof theory. Some of these fancy logics appear to be just what we need to deal with belief, uncertainty and change (though in practice we tend to find that no logic solves all our problems, so people end up picking a suitable logic off the shelf depending on their application).

The second approach is to base things not on a logic, but on probability theory. This too is a well understood theory, with clear results concerning uncertainty. Unfortunately basic probability theory tends to make too many assumptions concerning the availability and reliability of evidence, so we have to be careful how we use it. However, it still provides a good starting point and a way of assessing more ad hoc approaches to uncertainty - are they consistent with classical probability theory, given certain assumptions?

The first type of logic is *default* logic, which allows us to perform *non monotonic* inferences, where new information may mean that certain facts are no longer believed true.

### Default Logics

Default logics allow us to reason about things where there is a general case, true of almost everything, and some exceptions. We met this idea when looking at frame systems and

Notes

inheritance. Elephants are almost always grey, except Clyde, who's pink. It is very tedious to represent such things in first order predicate logic - you have to specify all the exceptions in the rules, which ends up getting very complex. To avoid this special logics have been developed. There are a variety of such logics, but all (most?) based on the same idea.

Default logics allow you to write rules which refer to whether it is consistent to believe something. So we'd say that if *X* is an elephant and it's consistent to believe that *X* is grey then *X* is grey. This is expressed in various ways in different variants of default logic, but one is to introduce a special operator *M*, so that *M X* means "*X* is consistent with everything else". Given the following rule and facts:

```
X elephant(X)   M grey(X)   grey(X)

grey(clyde)
elephant(nellie)
elephant(clyde)
```

we would conclude that Nellie was grey but we wouldn't conclude that Clyde was grey, as it is not *consistent* with everything else (ie, grey (clyde)).

If we had a hundred elephants, about ten of which were pink (or blue, or..), then we could write the general rule, and specify the exceptions by adding facts like: grey(albert), grey(edward).. If we bought a new elephant we'd just add a new fact of this sort. However, if we were using ordinary predicate logic we'd have to modify the general rule itself, which might look like:

```
X elephant(X)   name(X, clyde)  name(X, albert) name(X, edward) ....

grey(X)
```

If we have more complex defaults then the advantages of using a default logic become more compelling. For example, we might want to say that elephants are generally grey, circus elephants are generally pink, but Nellie is in fact grey. Dealing with all these nested defaults and exceptions explicitly in predicate logic gets very complex.

Default logics can be used to provide semantics. However, remember that in frame systems you sometimes had to choose which parent class to inherit from - if Clyde is both a circus animal and an elephant, is he likely to be tame or not? We have the same problem in default logics. There may be alternative, inconsistent conclusions, and the logic doesn't (usually) specify which is the right one. If we have a default rule that says that circus animals are generally not grey, if it is consistent to believe that they aren't grey, then we won't know whether our circus elephant is grey or not. Different alternative sets of facts we could believe are referred to as different *extensions* of the knowledge base.

### 13.5.1 Methods for Knowledge Acquisition

The aim of a probabilistic logic (also probability logic and probabilistic reasoning) is to combine the capacity of probability theory to handle uncertainty with the capacity of deductive logic to exploit structure. The result is a richer and more expressive formalism with a broad range of possible application areas.

*Did u know?* Probabilistic logics attempt to find a natural extension of traditional logic truth tables: the results they define are derived through probabilistic expressions instead.

A difficulty with probabilistic logics is that they tend to multiply the computational complexities of their probabilistic and logical components. Other difficulties include the possibility of counter-intuitive results, such as those of Dempster-Shafer theory. The need to deal with a broad variety of contexts and issues has led to many different proposals.

Notes

### 13.5.2 Validation of Knowledge

It is used in many computer science domains such as artificial intelligence, including databases, data mining, expert systems, decision support systems and geographic information systems. Knowledge engineering is also related to mathematical logic, as well as strongly involved in cognitive science and socio-cognitive engineering where the knowledge is produced by socio-cognitive aggregates (mainly humans) and is structured according to our understanding of how human reasoning and logic works.

Various activities of KE specific for the development of a knowledge-based system:

- Assessment of the problem
- Development of a knowledge-based system shell/structure
- Acquisition and structuring of the related *information*, *knowledge* and specific *preferences* (IPK model)
- Implementation of the structured knowledge into knowledge bases
- Testing and validation of the inserted knowledge
- Integration and maintenance of the system
- Revision and evaluation of the system.

Being still more art than engineering, KE is not as neat as the above list in practice. The phases overlap, the process might be iterative, and many challenges could appear.

### Self Assessment

State whether the following statements are true or false:

13. Different frames may share the different terminals.
14. Each piece of information about a particular frame is held in a slot.
15. An ANN is defined by four types of parameters.

## 13.6 Distinguishing Features of Expert Systems

#### Quick Availability and Opportunity to Program Itself

As the rule base is in everyday language (the engine is untouchable), expert system can be written much faster than a conventional program, by users or experts, bypassing professional developers and avoiding the need to explain the subject.

#### Ability to Exploit a Considerable Amount of Knowledge

The expert system uses a rule base, unlike conventional programs, which means that the volume of knowledge to program is not a major concern. Whether the rule base has 10 rules or 10,000, the engine operation is the same.

#### Reliability

The reliability of an expert system is the same as the reliability of a database, i.e. good, higher than that of a classical program. It also depends on the size of knowledge base.

Notes

### Scalability

Evolving an expert system is to add, modify or delete rules. Since the rules are written in plain language, it is easy to identify those to be removed or modified.

### Pedagogy

The engines that are run by a true logic are able to explain to the user in plain language why they ask a question and how they arrived at each deduction. In doing so, they show knowledge of the expert contained in the expert system. So, user can learn this knowledge in its context. Moreover, they can communicate their deductions step by step. So, the user has information about their problem even before the final answer of the expert system.

### Preservation and Improvement of Knowledge

Valuable knowledge can disappear with the death, resignation or retirement of an expert. Recorded in an expert system, it becomes eternal. To develop an expert system is to interview an expert and make the system aware of their knowledge. In doing so, it reflects and enhances it.

### New Areas Neglected by Conventional Computing

Automating a vast knowledge, the developer may meet a classic problem: "combinatorial explosion" commonly known as "information overload" that greatly complicates his work and results in a complex and time consuming program. The reasoning expert system does not encounter that problem since the engine automatically loads combinatorics between rules. This ability can address areas where combinatorics are enormous: highly interactive or conversational applications, fault diagnosis, decision support in complex systems, educational software, logic simulation of machines or systems, constantly changing software.

## 13.6.1 Utility of Expert Systems

Expert systems are especially important to organizations that rely on people who possess specialized knowledge of some problem domain, especially if this knowledge and experience cannot be easily transferred. Artificial intelligence methods and techniques have been applied to a broad range of problems and disciplines, some of which are esoteric and others which are extremely practical. One of the early applications, MYCIN, was created to help physicians diagnose and treat bacterial infections. Expert systems have been used to analyze geophysical data in our search for petroleum and metal deposits (e.g., PROSPECTOR). They are used by the investments, banking, and telecommunications industries. They are essential in robotics, natural language processing, theorem proving, and the intelligent retrieval of information from databases. They are used in many other human endeavors which might be considered more practical.

*Notes* Rule-based systems have been used to monitor and control traffic, to aid in the development of flight systems, and by the federal government to prepare budgets.

A rule-based, expert system maintains a separation between its Knowledge-base and that part of the system that executes rules, often referred to as the expert system shell. The system shell is indifferent to the rules it executes. This is an important distinction, because it means that the expert system shell can be applied to many different problem domains with little or no change.

Notes

It also means that adding or modifying rules to an expert system can effect changes in program behavior without affecting the controlling component, the system shell.

The language used to express a rule is closely related to the language subject matter experts use to describe problem solutions. When the subject matter expert composes a rule using this language, he is, at the same time, creating a written record of problem knowledge, which can then be shared with others. Thus, the creation of a rule kills two birds with one stone; the rule adds functionality or changes program behavior, and records essential information about the problem domain in a human-readable form. Knowledge captured and maintained by these systems ensures continuity of operations even as subject matter experts (i.e., mathematicians, accountants, physicians) retire or transfer.

Furthermore, changes to the Knowledge-base can be made easily by subject matter experts without programmer intervention, thereby reducing the cost of software maintenance and helping to ensure that changes are made in the way they were intended. Rules are added to the knowledge-base by subject matter experts using text or graphical editors that are integral to the system shell.

*Caution* Don't apply semantics directly.

*Task* Draw a probabilistic logic to find factorial of any number.

### Self Assessment

State whether the following statements are true or false:

16. The reliability of an expert system is the different from the reliability of a database.
17. Evolving an expert system is to add, modify or delete rules.
18. Valuable knowledge can appear with the death, resignation or retirement of an expert.

## 13.7 Summary

- Expert systems are designed to solve complex problems by reasoning about knowledge, like an expert, and not by following the procedure of a developer as is the case in conventional programming.
- The Knowledge-base Editor is a simple text editor, a graphical editor, or some hybrid of these two types.
- The Rule Engine (often referred to as an inference engine in AI literature) is responsible for executing Knowledge-base rules.
- A rule-based system consists of if-then rules, a bunch of facts, and an interpreter controlling the application of the rules, given the facts.
- The non-production system architecture of certain expert system does not have rule-representation scheme.
- Frames are structured sets of closely related knowledge, such as an object or concept name, the object's main attributes and their corresponding values, and possibly some attached procedure (if needed, if added, if-removed procedures).

## 13.8 Keywords

Notes

***Backward Chaining:*** It is an inference method that can be described (in lay terms) as working backward from the goals.

***Baye's Theorem:*** It provides us with a statistical theory of evidence which allows us to update the probabilities attached to certain facts in the light of new evidence. The fundamental notion is that of conditional probability:

***Expert Systems:*** It is a computer system that emulates the decision-making ability of a human expert.

***Inference Engine:*** It is computer program that tries to derive answers from a knowledge-base. It is the "brain" that expert systems use to reason about the information in the knowledge-base for the ultimate purpose of formulating new conclusions.

***Knowledge Acquisition:*** It is the transformation of knowledge from the form in that it is exist in the world into the form that it can be used by expertise systems.

***Knowledge Engineering:*** It is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise.

## 13.9 Review Questions

1. Why do we need knowledge engineers to develop expert systems? Can we ask experts to develop the system by themselves?
2. What do you think of their approach, which developed ten separate knowledge bases and then put them together through knowledge fusion? What are the pros and cons of this approach?
3. For systems testing and evaluation, they used simulated data rather than real data. Why is this acceptable for real-time expert systems? Can you think of a better approach for system testing and evaluation?
4. What are the benefits of using knowledge acquisition tools?
5. What are the steps in the knowledge engineering process?
6. Describe the evaluation, validation, and verification of knowledge.

### Answers: Self Assessment

1. False
2. True
3. True
4. False
5. False
6. True
7. True
8. False
9. True
10. True
11. True
12. True
13. False
14. True
15. False
16. False
17. True
18. False

Notes

## 13.10 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://cinuresearch.tripod.com/ai/www-cee-hw-ac-uk/_alison/ai3notes/subsection2_5_2_1.html

http://lecturer.ukdw.ac.id/jokop/wp-content/uploads/2011/4-5Architecture%20of%20ES1.pdf

http://www.cvel.clemson.edu/pdf/EMCS02-976.pdf

http://www.ece.unb.ca/jtaylor/Publications/ieee_proc84.pdf

http://www.elsevierdirect.com/companions/9780124438804/expert_sys_toc.pdf

http://www.ijcai.org/Past%20Proceedings/IJCAI-81-VOL-2/PDF/059.pdf

http://www.slideshare.net/poerslide/lecture5-expert-systems-and-artificial-intelligence

# Unit 14: Types of Learning

Notes

CONTENTS



## Objectives

After studying this unit, you will be able to:

- Explain Knowledge Acquisition Difficulties
- Discuss the General Learning Model
- Identify Performance Measures
- Identify various Knowledge System Building Tools
- Explain Learning by Induction
- Discuss Generalization and Specialization
- Define Inductive Bias
- Describe Analogical Reasoning and Learning
- Discuss Explanation-based Learning

Notes

## Introduction

Learning is acquiring new, or modifying existing, knowledge, behaviors, skills, values, or preferences and may involve synthesizing different types of information. The ability to learn is possessed by humans, animals and some machines. Progress over time tends to follow learning curves. Learning is not compulsory; it is contextual. It does not happen all at once, but builds upon and is shaped by what we already know. To that end, learning may be viewed as a process, rather than a collection of factual and procedural knowledge. Learning produces changes in the organism and the changes produced are relatively permanent.

Human learning may occur as part of education, personal development, schooling, or training. It may be goal-oriented and may be aided by motivation. The study of how learning occurs is part of neuropsychology, educational psychology, learning theory, and pedagogy. Learning may occur as a result of habituation or classical conditioning, seen in many animal species, or as a result of more complex activities such as play, seen only in relatively intelligent animals. Learning may occur consciously or without conscious awareness. Learning that an aversive event can't be avoided nor escaped is called learned helplessness. There is evidence for human behavioral learning prenatally, in which habituation has been observed as early as 32 weeks into gestation, indicating that the central nervous system is sufficiently developed and primed for learning and memory to occur very early on in development.

Play has been approached by several theorists as the first form of learning. Children experiment with the world, learn the rules, and learn to interact through play. Lev Vygotsky agrees that play is pivotal for children's development, since they make meaning of their environment through play. 85 percent of brain development occurs during the first five years of a child's life. The context of conversation based on moral reasoning offers some proper observations on the responsibilities of parents.

## 14.1 Knowledge Acquisition Difficulties

Knowledge acquisition is a method of learning, first proposed by Aristotle in his seminal work "Organon". Aristotle proposed that the mind at birth is a blank slate, or tabula rasa. As a blank slate it contains no knowledge of the objective, empirical universe, nor of itself. As a method, it is opposed to the concept of "a priori" knowledge, and to "intuition" when conceived as religious revelation. It has been suggested that the mind is "hard wired" to begin operating at birth, beginning a lifetime process of acquisition through abstraction, induction, and conception. The "five senses" referred to by the word sensation are metaphorically the interface between empirical (sensate) reality and the consciousness of the knowing subject. A knowing subject for the purpose of this discussion of knowledge acquisition may be defined as any conscious creature capable of deriving direct and immediate sensate data from its environment. Sensate data, or sensation, are distinct from perception. Perception is the recognition within the knowing subject of the event of having had a sensation. The tabula rasa and must learn the nature of sensation as the awareness of something which is outside itself. Commonly recognized sensory systems are those for vision, hearing, somatic sensation (touch), taste and olfaction (smell). Perception is the retention of a group of sensations transmitted through the sensory system(s), which gives the knowing subject the ability to be aware, not only of the singularity of stimuli presented by sensation itself, but of an entity, a thing, an existent. Retention of percepts allows the human mind to abstract information from the percepts. The abstraction is considered the extensional definition of the percept. An extension is "every object that falls under the definition of the concept or term in question." This is the same as a universal (metaphysics) or genus or denotation, or class (philosophy).

Notes

Once a universal (class) has been identified, then the next step in the acquisition of knowledge is the abstraction of the intension, which is the particular, the species, or the connotation. Connotation as its meaning as particular is "*the assertion that at least one member of one class of things is either included or excluded as a member of some other class.*" This means, for example, that a poodle is the particular in a class or universal concept called "dog" or "canine".

Some of the most important issues in knowledge acquisition are as follows:

- Most knowledge is in the heads of experts
- Experts have vast amounts of knowledge
- Experts have a lot of tacit knowledge
  - ❖ They don't know all that they know and use
  - ❖ Tacit knowledge is hard (impossible) to describe
- Experts are very busy and valuable people
- Each expert doesn't know everything
- Knowledge has a "shelf life"

### Self Assessment

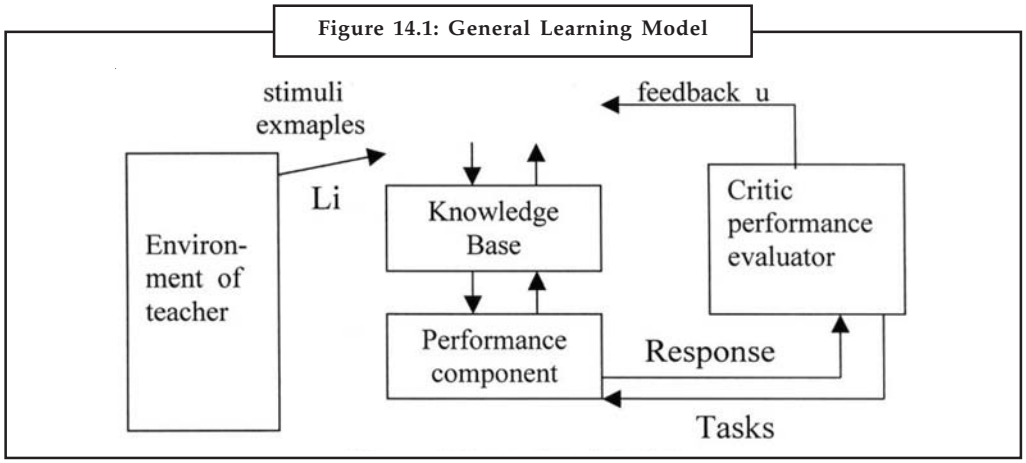
State whether the following statements are true or false:

1. Perception is the recognition within the knowing subject of the event of having had a sensation.
2. Knowledge acquisition is not a method of learning.

## 14.2 General Learning Model

Learning can be accomplished using a number of different methods, such as by memorization facts, by being told, or by studying examples like problem solution. Learning requires that new knowledge structures be created from some form of input stimulus. This new knowledge must then be assimilated into a knowledge base and be tested in some way for its utility. Testing means that the knowledge should be used in performance of some task from which meaningful feedback can be obtained, where the feedback provides some measure of the accuracy and usefulness of the newly acquired knowledge.

**Figure 14.1: General Learning Model**



Notes

General learning model is depicted in figure 14.1 where the environment has been included as a part of the overall learner system. The environment may be regarded as either a form of nature which produces random stimuli or as a more organized training source such as a teacher which provides carefully selected training examples for the learner component. The actual form of environment used will depend on the particular learning paradigm. In any case, some representation language must be assumed for communication between the environment and the learner. The language may be the same representation scheme as that used in the knowledge base (such as a form of predicate calculus). When they are chosen to be the same, we say the single representation trick is being used. This usually results in a simpler implementation since it is not necessary to transform between two or more different representations. For some systems the environment may be a user working at a keyboard. Other systems will use program modules to simulate a particular environment.

*Notes* In even more realistic cases the system will have real physical sensors which interface with some world environment. Inputs to the learner component may be physical stimuli of some type or descriptive, symbolic training examples.

The information conveyed to the learner component is used to create and modify knowledge structures in the knowledge base. This same knowledge is used by the performance component to carry out some tasks, such as solving a problem playing a game or classifying instances of some concept given a task, the performance component produces a response describing its action in performing the task. The critic module then evaluates this response relative to an optimal response. Feedback, indicating whether or not the performance was acceptable, is then sent by the critic module to the learner component for its subsequent use in modifying the structures in the knowledge base. If proper learning was accomplished, the system's performance will have improved with the changes made to the knowledge base.

The cycle described above may be repeated a number of times until the performance of the system has reached some acceptable level, until a known learning goal has been reached, or until changes ceases to occur in the knowledge base after some chosen number of training examples have been observed. There are several important factors which influence a system's ability to learn in addition to the form of representation used. They include the types of training provided, the form and extent of any initial background knowledge, the type of feedback provided, and the learning algorithms used. The type of training used in a system can have a strong effect on performance, much the same as it does for humans. Training may consist of randomly selected instance or examples that have been carefully selected and ordered for presentation. The instances may be positive examples of some concept or task a being learned, they may be negative, or they may be mixture of both positive and negative. The instances may be well focused using only relevant information, or they may contain a variety of facts and details including irrelevant data. Domain-specific learning theories of development hold that we have many independent, specialized knowledge structures, rather than one cohesive knowledge structure. Thus, training in one domain may not impact another independent domain. For example, core knowledge theorists believe we have highly specialized functions that are independent of one another. Jean Piaget's theory of development, on the other hand, believed that knowledge is internalized into a cohesive knowledge structure, favoring the domain-general learning model.

In this model, the purpose of a learning machine is to be able to infer certain facts of some data X from a training set selected from X.

Notes

*Example:* Over the set of all people we wish to automatically distinguish some person as either male or female based on a collection of n observations such as height, weight, pitch of voice, etc. We stack these observations into a feature vector x.

If we have chosen the types of these observations intelligently then the distributions of these characteristics may tell us how to classify some person described by x.

*Example:* Males will on average be taller, heavier, and have a lower pitched voice than females. So based on some instance of these characteristics, the hope is that we will be able to predict the sex of a subject with some amount of certainty.

### Self Assessment

State whether the following statements are true or false:

3. Learning can be accomplished using same methods.
4. The information conveyed to the learner component is used to create and modify knowledge structures in the knowledge base.

## 14.3 Performance Measures

A performance measure is a numeric description of an agency's work and the results of that work. Performance measures are based on data, and tell a story about whether an agency or activity is achieving its objectives and if progress is being made toward attaining policy or organizational goals. In technical terms, a performance measure is a quantifiable expression of the amount, cost, or result of activities that indicate how much, how well, and at what level, products are provided to customers during a given time period. "Quantifiable" means the description can be counted more than once, or measured using numbers. "Activities" mean the work, business processes and functions of Washington state government agencies. "Results" are what the agency's work is intended to achieve or accomplish for its customers. Performance measurement is the process of collecting, analyzing and/or reporting information regarding the performance of an individual, group, organization, system or component. It can involve studying processes/strategies within organizations, or studying engineering processes/ parameters/phenomena, to see whether output are in line with what was intended or should have been achieved. There are several reasons to measure, monitor and report performance of our work. It's the right thing to do: We measure many things in our lives outside work. Our children regularly bring home objective measures of their performance at school (i.e. test scores and report cards). We get monthly measures of performance at home: investment performance, water and electricity usage, and so on. We monitor our health through a variety of measures of how well our body is performing: weight, blood pressure, cholesterol levels. Work performance is another aspect of our lives, and measuring it should be what we do.

### Self Assessment

State whether the following statements are true or false:

5. Performance measures are not based on data.
6. Quantifiable means the description can be counted more than once, or measured using numbers.

Notes

## 14.4 Knowledge System Building Tools

In the best performance management systems, actions and results are logically related to one another by a theory of causality, or "logic model." Potential measures come from understanding the purpose of the organization and what is being done to accomplish the organization's mission. Logic models are a useful tool for this. Agencies exist to carry out certain lines of business or activities. Each activity is accomplished through a business process.

*Example:* The Human Trafficking Reporting System (HTRS) is a secured, online performance measurement portal. The HTRS collects data on suspected human trafficking incidents, offenders, and victims from DOJ-funded human trafficking task forces. These task forces can encompass cities, regions, territories, or states. Information collected includes incident status, type of human trafficking, lead investigating agency, number of known victims, number of known offenders, whether the case was confirmed as human trafficking, the demographic characteristics of offenders and victims, case processing information of offenders, and victim service provision information.

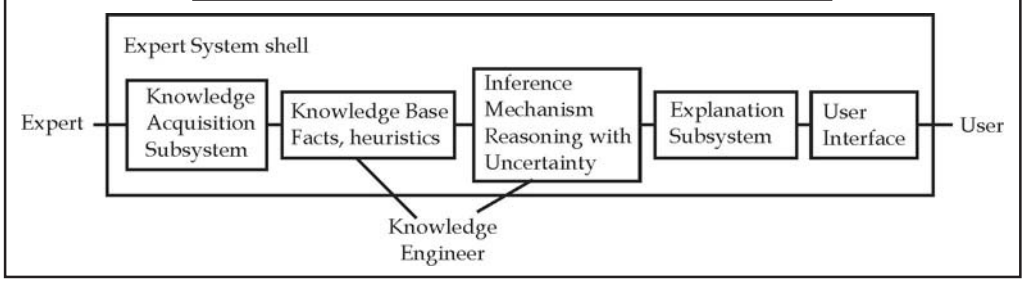
The various learning and their meaning are written below whose knowledge may be measured:

- Perceptual learning – ability to learn to recognize stimuli that have been seen before.
- Stimulus-response learning – ability to learn to perform a particular behavior
- when a certain stimulus is present.
- Motor learning – establishment of changes within the motor system
- Relational learning – involves connections between different areas of the association cortex.
- Spatial learning – involves learning about the relations among many stimuli
- Episodic learning – remembering sequences of events that we witness
- Observational learning – learning by watching and imitation other people

### 14.4.1 ES Building Tools on the Market

An expert system tool, or shell, is a software development environment containing the basic components of expert systems. Associated with a shell is a prescribed method for building applications by configuring and instantiating these components. Some of the generic components of a shell are shown in Figure shown and described below. The core components of expert systems are the knowledge base and the reasoning engine.

**Figure 14.2: Basic Components of Expert System Tools**



Notes

- ***Knowledge Base:*** A store of factual and heuristic knowledge. An ES tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both frames (objects) and IF-THEN rules. In Prolog, the knowledge is represented as logical statements.
- ***Reasoning Engine:*** Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base to form a line of reasoning in solving a problem. The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to case-based reasoning.
- ***Knowledge Acquisition Subsystem:*** A subsystem to help experts build knowledge bases. Collecting knowledge needed to solve problems and build the knowledge base continues to be the biggest bottleneck in building expert systems.
- ***Explanation Subsystem:*** A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at to justifying the need for additional data.
- ***User Interface:*** The means of communication with the user. The user interface is generally not a part of the ES technology, and was not given much attention in the past. However, it is now widely accepted that the user interface can make a critical difference in the perceived utility of a system regardless of the system's performance.

### 14.4.2 Market Trends

Tablets, mobile applications and social user experiences are set to dominate the technology landscape in 2012, according to market analyst Gartner. The addition of sensors and intelligence into everyday consumer devices, growing application stores, next-generation analytics, big data, in-memory computing, extreme low-energy servers and cloud computing are also trends that will make their way into the mainstream during 2012 and beyond. Early adopters and avid technology fans have already had their eyes (and fingers) on many of these emerging trends throughout 2011 but Gartner predicts that these ten trends will have a significant impact on the enterprise in the years ahead. "The user interface (IU) paradigm in place for more than 20 years is changing. UIs with windows, icons, menus, and pointers will be replaced by mobile-centric interfaces emphasizing touch, gesture, search, voice and video," says Gartner. Context-aware computing is also set to get a boost - that's great news for consumers as the focus will be placed on them and their needs. "A contextually aware system anticipates the user's needs and proactively serves up the most appropriate and customized content, product or service," explains Gartner. 2012 will be all about "The Internet of Things" – giving users the ability to interact with physical items and to track every element of their lives. This will be achieved by integrating technology such as sensors, Near Field Communication (NFC), artificial intelligence and image recognition into everyday objects and connecting them to the web.

## Self Assessment

State whether the following statements are true or false:

7. The core components of expert systems are the knowledge base and the reasoning engine.
8. The user interface is generally a part of the ES technology.

Notes

## 14.5 Learning by Induction

Inductive learning is essentially learning by example. The process itself ideally implies some method for drawing conclusions about previously unseen examples once learning is complete. More formally, one might state: Given a set of training examples, develop a hypothesis that is as consistent as possible with the provided data. It is worthy of note that this is an imperfect technique. As Chalmers points out, "an inductive inference with true premises [can] lead to false conclusions". The example set may be an incomplete representation of the true population, or correct but inappropriate rules may be derived which apply only to the example set. A simple demonstration of this type of learning is to consider the following set of bit-strings (each digit can only take on the value 0 or 1), each noted as either a positive or negative.

In logic, we often refer to the two broad methods of reasoning as the deductive and inductive approaches. Deductive reasoning works from the more general to the more specific. Sometimes this is informally called a "top-down" approach. We might begin with thinking up a theory about our topic of interest. We then narrow that down into more specific hypotheses that we can test. We narrow down even further when we collect observations to address the hypotheses. This ultimately leads us to be able to test the hypotheses with specific data – a confirmation (or not) of our original theories.

Inductive reasoning works the other way, moving from specific observations to broader generalizations and theories. Informally, we sometimes call this a "bottom up" approach. In inductive reasoning, we begin with specific observations and measures, begin to detect patterns and regularities, formulate some tentative hypotheses that we can explore, and finally end up developing some general conclusions or theories.

*Caution* Learning should be applied after refining the given task.

*Task* Create a learning chart for a car Driving.

### Self Assessment

State whether the following statements are true or false:

9. Deductive reasoning works from the more general to the more specific.

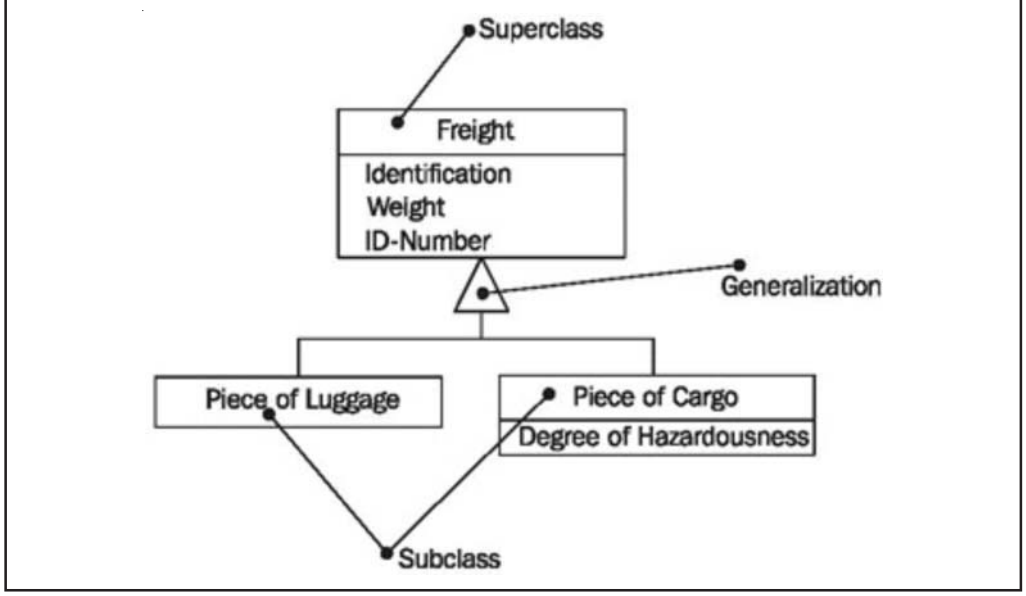
10. Inductive learning is essentially learning by example.

## 14.6 Generalization and Specialization

Terms such as superclass, subclass, or inheritance come to mind when thinking about the object-oriented approach. These concepts are very important when dealing with object-oriented programming languages such as Java, Smalltalk, or C++. For modeling classes that illustrate technical concepts they are secondary. The reason for this is that modeling relevant objects or ideas from the real world gives little opportunity for using inheritance (compare the class diagram of our case study). Nevertheless, we would like to further introduce these terms at this point in Figure 14.3 shown below:

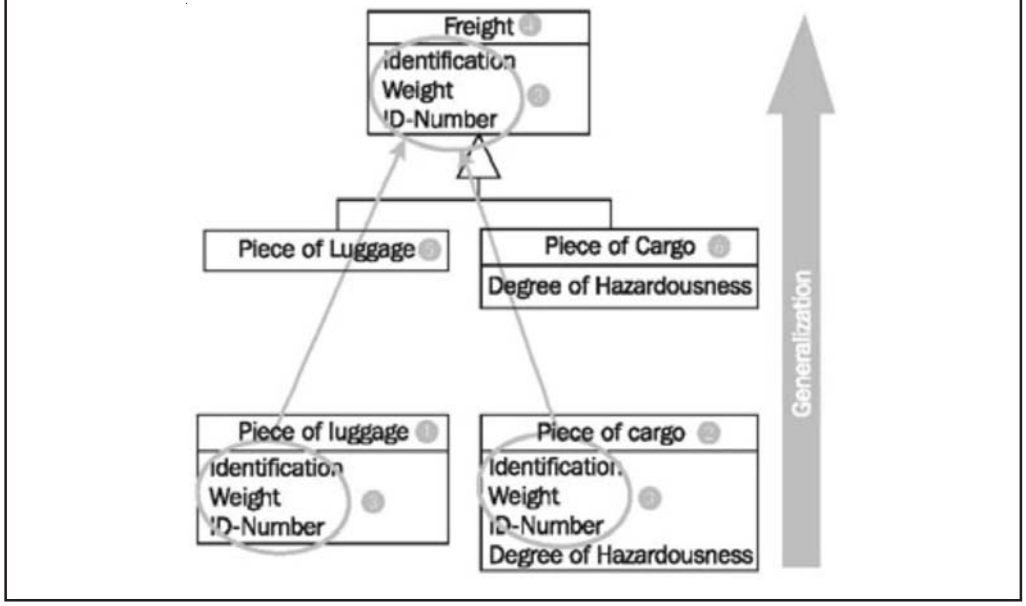
Notes

**Figure 14.3: Notation of Generalization**




*Generalization* is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass. Shared characteristics can be attributes, associations, or methods. In next figure 14.4, the classes *Piece of Luggage* (1) and *Piece of Cargo* (2) partially share the same attributes. From a domain perspective, the two classes are also very similar. During generalization, the shared characteristics (3) are combined and used to create a new superclass *Freight* (4). *Piece of Luggage* (5) and *Piece of Cargo* (6) become subclasses of the class *Freight*.

**Figure 14.4: Example of Generalization**

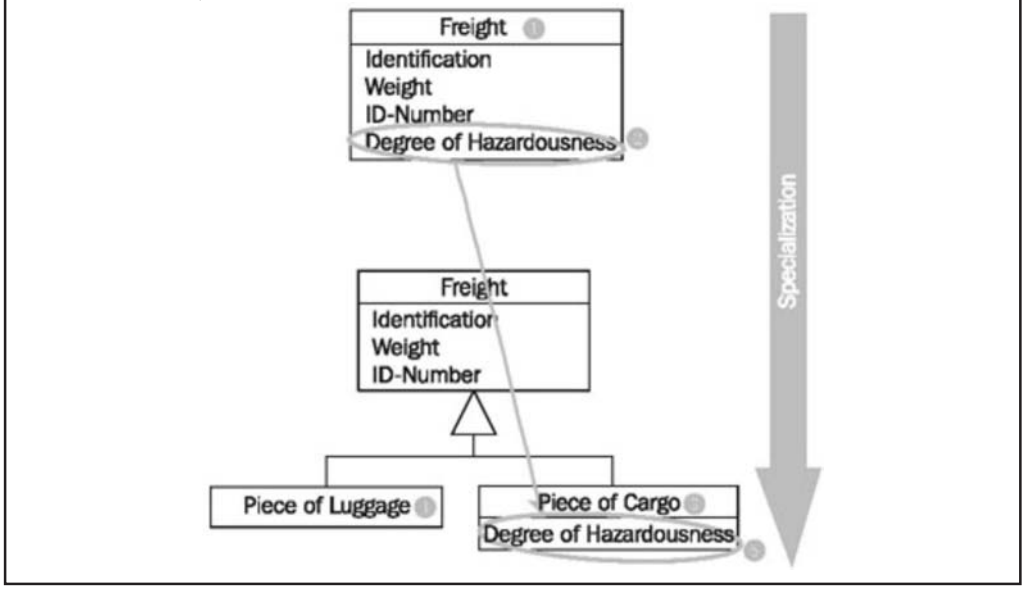



The shared attributes (3) are only listed in the superclass, but also apply to the two subclasses, even though they are not listed there.

Notes

In contrast to generalization, *specialization* means creating new subclasses from an existing class. If it turns out that certain attributes, associations, or methods only apply to some of the objects of the class, a subclass can be created. The most inclusive class in a generalization/specialization is called the superclass and is generally located at the top of the diagram. The more specific classes are called subclasses and are generally placed below the superclass. In figure, the class *Freight* (1) has the attribute *Degree of Hazardousness* (2), which is needed only for cargo, but not for passenger luggage. Additionally, only passenger luggage has a connection to a coupon. Obviously, here two similar but different domain concepts are combined into one class. Through specialization the two special cases of freights are formed: *Piece of Cargo* (3) and *Piece of Luggage* (4). The attribute *Degree of Hazardousness* (5) is placed where it belongs—in *Piece of Cargo*. The attributes of the class *Freight* (1) also apply to the two subclasses *Piece of Cargo* (3) and *Piece of Luggage* (4):

**Figure 14.5: Example of Specialization**




So much for the mechanism, however, the domain meaning of the relationship between superclass and subclass is much more important. These rules apply to this relationship: All statements that are made about a superclass also apply to all subclasses. We say that subclasses "inherit" attributes, associations, and operations from the superclass. For example, if the superclass Freight has an attribute Weight, then the subclass piece of luggage also has an attribute Weight, even though this attribute is not listed in the subclass Piece of Luggage. Anything that can be done with an object of the superclass can also be done with an object of the subclass. For example, if freight can be loaded, pieces of luggage can also be loaded. In the terminology of the system that is being modeled, a subclass has to be a special form of the superclass. For example, a piece of luggage is a special case of freight. The counter-example to this is: A flight is not a special case of a flight number.

## Self Assessment

State whether the following statements are true or false:

11. Specialisation is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass.

12. The shared attributes are not listed in the superclass.

## 14.7 Inductive Bias

Notes

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered (Mitchell, 1980).

In machine learning, one aims to construct algorithms that are able to learn to predict a certain target output. To achieve this, the learning algorithm is presented some training examples that demonstrate the intended relation of input and output values. Then the learner is supposed to approximate the correct output, even for examples that have not been shown during training. Without any additional assumptions, this task cannot be solved exactly since unseen situations might have an arbitrary output value. The kind of necessary assumptions about the nature of the target function are subsumed in the term inductive bias (Mitchell, 1980; desJardins and Gordon, 1995). A classical example of an inductive bias is Occam's Razor, assuming that the simplest consistent hypothesis about the target function is actually the best. Here consistent means that the hypothesis of the learner yields correct outputs for all of the examples that have been given to the algorithm. Approaches to a more formal definition of inductive bias are based on mathematical logic. Here, the inductive bias is a logical formula that, together with the training data, logically entails the hypothesis generated by the learner.

*Caution* Unfortunately, this strict formalism fails in many practical cases, where the inductive bias can only be given as a rough description (e.g. in the case of neural networks), or not at all.

The following is a list of common inductive biases in machine learning algorithms.

- ***Maximum Conditional Independence:*** If the hypothesis can be cast in a Bayesian framework, try to maximize conditional independence. This is the bias used in the Naive Bayes classifier.
- ***Minimum Cross-validation Error:*** When trying to choose among hypotheses, select the hypothesis with the lowest cross-validation error. Although cross-validation may seem to be free of bias, the No Free Lunch theorems show that cross-validation must be biased.
- ***Maximum Margin:*** When drawing a boundary between two classes, attempt to maximize the width of the boundary. This is the bias used in Support Vector Machines. The assumption is that distinct classes tend to be separated by wide boundaries.
- ***Minimum Description Length:*** When forming a hypothesis, attempt to minimize the length of the description of the hypothesis. The assumption is that simpler hypotheses are more likely to be true.
- ***Minimum Features:*** Unless there is good evidence that a feature is useful, it should be deleted. This is the assumption behind feature selection algorithms.
- ***Nearest Neighbors:*** Assume that most of the cases in a small neighborhood in feature space belong to the same class. Given a case for which the class is unknown, guess that it belongs to the same class as the majority in its immediate neighborhood. This is the bias used in the k-nearest neighbor algorithm. The assumption is that cases that are near each other tend to belong to the same class.

Notes

### Self Assessment

State whether the following statements are true or false:

13. If the hypothesis can be cast in a Bayesian framework, try to minimise conditional independence.

14. When forming a hypothesis, attempt to minimize the length of the description of the hypothesis.

## 14.8 Analogical Reasoning and Learning

Analogy plays an important role in learning and instruction. As John Bransford, Jeffrey Franks, Nancy Vye, and Robert Sherwood noted in 1989, analogies can help students make connections between different concepts and transfer knowledge from a well-understood domain to one that is unfamiliar or not directly perceptual.

*Example:* The circulatory system is often explained as being like a plumbing system, with the heart as pump.

Analogical reasoning involves several sub-processes: (1) retrieval of one case given another; (2) mapping between two cases in working memory; (3) evaluating the analogy and its inferences; and, sometimes, (4) abstracting the common structure. The core process in analogical reasoning is mapping. According to structure-mapping theory, developed by Dedre Gentner in 1982, an analogy is a mapping of knowledge from one domain (the base or source) into another (the target) such that a system of relations that holds among the base objects also holds among the target objects. In interpreting an analogy, people seek to put the objects of the base in one-to-one correspondence with the objects of the target so as to obtain the maximal structural match. The corresponding objects in the base and target need not resemble each other; what is important is that they hold like roles in the matching relational structures. Thus, analogy provides a way to focus on relational commonalities independently of the objects in which those relations are embedded. In explanatory analogy, a well-understood base or source situation is mapped to a target situation that is less familiar and/or less concrete. Once the two situations are aligned – that is, once the learner has established correspondences between them – then new inferences are derived by importing connected information from the base to the target. For example, in the analogy between blood circulation and plumbing, students might first align the known facts that the pump causes water to flow through the pipes with the fact that the heart causes blood to flow through the veins. Given this alignment of structure, the learner can carry over additional inferences: for example, that plaque in the veins forces the heart to work harder, just as narrow pipes require a pump to work harder.

### 14.8.1 Makes a Good Analogy

Analogy examples with corresponding meanings are the best way to show the meaning of the word "analogy." The following is a list of some common analogies and an explanation of their meaning.

- The relationship between them began to thaw. This means that the relationship was changing.
- You are as annoying as nails on a chalkboard. You must be pretty annoying for someone to say that.
- I am going to be toast when I get home. This is usually said when someone is in trouble with their significant other.

Notes

- He is like a rock. This means he is steadfast and strong.
- She attended the celebrity roast. The person being roasted is being honored by people making harmless jokes about him or her.
- I feel like a fish out of water. This implies that you are not comfortable in your surroundings.
- She was offended when I said she was as flaky as a snowstorm. That isn't a very nice comparison to make.
- There are plenty of fish in the sea. Unless you really are a fish, this encourages you to move on and find another potential mate.
- She was as quiet as a mouse. It is hard to hear a mouse, so that means she was very quiet.
- Bing Crosby had a velvet voice. Since voices are not made of velvet, this implies that his voice was smooth and soothing.
- Life is like a box of chocolates. This has many meanings and is a great analogy for life.

Analogy plays a significant role in problem solving such as, decision making, perception, memory, creativity, emotion, explanation and communication. It lies behind basic tasks such as the identification of places, objects and people, for example, in face perception and facial recognition systems. It has been argued that analogy is "the core of cognition". Specific analogical language comprises exemplification, comparisons, metaphors, similes, allegories, and parables, but not metonymy. Phrases like and so on, and the like, as if, and the very word like also rely on an analogical understanding by the receiver of a message including them. Analogy is important not only in ordinary language and common sense (where proverbs and idioms give many examples of its application) but also in science, philosophy and the humanities. The concepts of association, comparison, correspondence, mathematical and morphological homology, homomorphism, iconicity, isomorphism, metaphor, resemblance, and similarity are closely related to analogy. In cognitive linguistics, the notion of conceptual metaphor may be equivalent to that of analogy.

Analogy has been studied and discussed since classical antiquity by philosophers, scientists and lawyers. The last few decades have shown a renewed interest in analogy, most notably in cognitive science.

In addition to the above general qualities, several further factors influence the success of an explanatory analogy, including base specificity, transparency, and scope. Base specificity is the degree to which the structure of the base domain is clearly understood. Transparency is the ease with which the correspondences can be seen. Transparency is increased by similarities between corresponding objects and is decreased by similarities between non corresponding objects. For example, in 1986 Gentner and Cecile Toupin found that four-to six-year-old children succeeded in transferring a story to new characters when similar characters occupied similar roles (e.g., squirrel [.arrowright] chipmunk; trout salmon), but they failed when the match was cross-mapped, with similar characters in different roles (e.g., squirrel salmon; trout chipmunk). The same pattern has been found with adults. Transparency also applies to relations. In 2001, Miriam Bassok found that students more easily aligned instances of "increase" when both were continuous (e.g., speed of a car and growth of a population) than when one was discrete (e.g., attendance at an annual event).

### 14.8.2 Methods Used to Investigate Analogical Learning

Much research on analogy in learning has been devoted to the effects of analogies on domain understanding.

Notes

*Example:* In 1987, Brian Ross found that giving learners analogical examples to illustrate a probability principle facilitated their later use of the probability formula to solve other problems. In classroom studies from 1998, Daniel Schwartz and John Bransford found that generating distinctions between contrasting cases improved students' subsequent learning. As reported in 1993, John Clement used a technique of bridging analogies to induce revision of faulty mental models. Learners were given a series of analogs, beginning with a very close match and moving gradually to a situation that exemplified the desired new model.

Another line of inquiry focuses on the spontaneous analogies people use as mental models of the world. This research generally begins with a questionnaire or interview to elicit the person's own analogical models. For example, Willet Kempton in 1986 used interviews to uncover two common analogical models of home heating systems. In the (incorrect) valve model, the thermostat is like a faucet: It controls the rate at which the furnace produces heat. In the (correct) threshold model, the thermostat is like an oven: It simply controls the goal temperature, and the furnace runs at a constant rate. Kempton then examined household thermostat records and found patterns of thermostat settings corresponding to the two analogies. Some families constantly adjusted their thermostats from high to low temperatures, an expensive strategy that follows from the valve model. Others simply set their thermostat twice a day – low at night, higher by day, consistent with the threshold model.

### Retrieval of Analogs: The Inert Knowledge Problem

Learning from cases is often easier than learning principles directly. Despite its usefulness, however, training with examples and cases often fails to lead to transfer, because people fail to retrieve potentially useful analogs. For example, Mary Gick and Holyoak found in 1980 that participants given an insight problem typically failed to solve it, even when they had just read a story with an analogous solution. Yet, when they were told to use the prior example, they were able to do so. This shows that the prior knowledge was not lost from memory; this failure to access prior structurally similar cases is, rather, an instance of "inert knowledge" – knowledge that is not accessed when needed.

One explanation for this failure of transfer is that people often encode cases in a situation-specific manner, so that later remindings occur only for highly similar cases. For example, in 1984, Ross gave people mathematical problems to study and later gave them new problems. Most of their later remindings were to examples that were similar only on the surface, irrespective of whether the principles matched. Experts in a domain are more likely than novices to retrieve structurally similar examples, but even experts retrieve some examples that are similar only on the surface. However, as demonstrated by Laura Novick in 1988, experts reject spurious reminding, more quickly than do novices. Thus, especially for novices, there is an unfortunate dissociation: While accuracy of transfer depends critically on the degree of structural match, memory retrieval depends largely on surface similarity between objects and contexts.

### Analogical Encoding in Learning

In the late 20th century, researchers began exploring a new technique, called analogical encoding, that can help overcome the inert knowledge problem. Instead of studying cases separately, learners are asked to compare analogous cases and describe their similarities. This fosters the formation of a common schema, which in turn facilitates transfer to a further problem.

*Example:* In 1999, Jeffrey Loewenstein, Leigh Thompson, and Gentner found that graduate management students who compared two analogical cases were nearly three times

Notes

more likely to transfer the common strategy into a subsequent negotiation task than were students who analyzed the same two cases separately.

### 14.8.3 Retrieval of Analogy: The Inert Knowledge Problem

Inert knowledge is information which one can express but not use. The process of understanding by learners does not happen to that extent where the knowledge can be used for effective problem-solving in realistic situations. The phenomenon of inert knowledge was first described in 1929 by Alfred North Whitehead:

*"Theoretical ideas should always find important applications within the pupil's curriculum. This is not an easy doctrine to apply, but a very hard one. It contains within itself the problem of keeping knowledge alive, of preventing it from becoming inert, which is the central problem of all education."*

An example for inert knowledge is vocabulary of a foreign knowledge which is available during an exam but not in a real situation of communication.

An explanation for the problem of inert knowledge is that people often encode knowledge to a specific situation, so that later remindings occur only for highly similar situations

In contrast so called conditionalized knowledge is knowledge about something which includes also knowledge as to the contexts in which that certain knowledge will be useful.

### Self Assessment

State whether the following statements are true or false:

15. Analogy plays an important role in learning and instruction.

16. Learning from cases is often difficult than learning principles directly.

## 14.9 Explanation-based Learning (EBL)

Explanation-based learning (EBL) is a form of machine learning that exploits a very strong, or even perfect, domain theory to make generalizations or form concepts from training examples.

An example of EBL using a perfect domain theory is a program that learns to play chess by being shown examples. A specific chess position that contains an important feature, say, "Forced loss of black queen in two moves," includes many irrelevant features, such as the specific scattering of pawns on the board. EBL can take a single training example and determine what are the relevant features in order to form a generalization. A domain theory is perfect or complete if it contains, in principle, all information needed to decide any question about the domain. For example, the domain theory for chess is simply the rules of chess. Knowing the rules, in principle it is possible to deduce the best move in any situation. However, actually making such a deduction is impossible in practice due to combinatoric explosion. EBL uses training examples to make searching for deductive consequences of a domain theory efficient in practice. An EBL system works, by finding a way to deduce each training example, from the system's existing database of domain theory. Having a short proof of the training example extends the domain-theory database, enabling the EBL system to find and classify future examples that are similar to the training example very quickly. The main drawback of the method, the cost of applying the learned proof macros as these become numerous.

*Did u know?* Humans appear to learn quite a lot from one example.

Notes

***Basic Idea:*** Use results from one examples problem solving effort next time around. An EBL accepts 4 kinds of input:

- A training example – what the learning sees in the world.
- A goal concept – a high level description of what the program is supposed to learn.
- An operational criterion - a description of which concepts are usable.
- A domain theory – a set of rules that describe relationships between objects and actions in a domain.
- From this EBL computes a generalization of the training example that is sufficient not only to describe the goal concept but also satisfies the operational criterion. This has two steps:
- Explanation – the domain theory is used to prune away all unimportant aspects of the training example with respect to the goal concept.
- Generalization – the explanation is generalized as far possible while still describing the goal concept.

*Task* Make a difference hart between learning and knowledge acquisition.

### Self Assessment

State whether the following statements are true or false:

17. A domain theory is perfect or complete if it contains, in principle, all information needed to decide any question about the domain.
18. EBL uses training examples to make searching for deductive consequences of a domain theory efficient in practice.

## 14.10 Summary

- Learning is acquiring new knowledge, behaviours, skills, values, preferences or understanding, and may involve synthesizing different types of information.
- Knowledge acquisition is the process of adding new knowledge to a knowledge-base and refining or otherwise improving knowledge that was previously acquired.
- Explanation-based learning (EBL) systems attempt to improve the performance of a problem solver (PS), by first examining how PS solved previous problems, then modifying PS to enable it to solve similar problems better (typically, more efficiently) in the future.
- Inductive biases can be probabilistically correct or probabilistically incorrect, and if they are correct, it is good to have as much of them as possible, and if they are incorrect, we are left worse off than if you had no inductive bias at all.
- 'Learning' is a broad term covering a wide range of processes. We learn (memorize) multiplication tables, learn (discover how) to walk, learn (build up an understanding of, then an ability to synthesize) languages.
- Performance measures are based on data, and tell a story about whether an agency or activity is achieving its objectives and if progress is being made toward attaining policy or organizational goals.

Notes

- An expert system tool, or shell, is a software development environment containing the basic components of expert systems
- Inductive reasoning works the other way, moving from specific observations to broader generalizations and theories.
- The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

## 14.11 Keywords

***Backtracking:*** It is a general algorithm for finding all (or some) solutions to some computational problem that incrementally builds candidates to the solutions.

***Explanation-Based Learning (EBL):*** It is a form of machine learning that exploits a very strong, or even perfect, domain theory to make generalizations or form concepts from training examples.

***Knowledge Acquisition:*** It is a structured way of developing knowledge-based systems (expert systems).

***Knowledge-based Systems:*** Knowledge-based system are artificial intelligent tools working in a narrow domain to provide intelligent decisions with justification.

***User Interface:*** The user interface, in the industrial design field of human – machine interaction, is the space where interaction between humans and machines occurs.

## 14.12 Review Questions

1. Explain the knowledge acquisition difficulties.
2. Explain the general learning model.
3. What are the different measures of performance?
4. Write few knowledge system building tools.
5. What is learning by induction?
6. Explain super class.
7. What is the inductive bias?
8. Explain analogical reasoning and learning.
9. What is explanation based learning?
10. Differentiate generalization and aggregation.

### Answers: Self Assessment

| | | | |
|---|---|---|---|
| 1. | True | 2. | False |
| 3. | False | 4. | True |
| 5. | False | 6. | True |
| 7. | True | 8. | True |
| 9. | True | 10. | True |
| 11. | False | 12. | False |

Notes

13. False
14. True
15. True
16. False
17. True
18. True

## 14.13 Further Readings

*Books*

Deshpande, Neeta (2009), *Artificial Intelligence,* Technical Publications.

Harris, Michael C. (2010), *Artificial Intelligence,* Marshall Cavendish.

Rich, Elaine (2004), *Artificial Intelligence 3E (Sie),* Tata McGraw-Hill Education.

Russell, Stuart (2003), *Artificial Intelligence: A Modern Approach, 2/E,* Pearson Education India.

Whitby, Blay (2009), *Artificial Intelligence,* The Rosen Publishing Group.

*Online links*

http://home.epix.net/~tcannon1/Physioweek10.htm

http://www.chess.com/article/view/learning-by-induction

http://www.claes.sci.eg/Department.aspx?DepId=272&lang=en

http://www.eolss.net/Sample-Chapters/C15/E6-44-03.pdf

http://www.washington.edu/doit/TeamN/types.html

http://www.wtec.org/loyola/kb/c3_s2.htm

**LOVELY PROFESSIONAL UNIVERSITY**
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in