LOVELY PROFESSIONAL UNIVERSITY

ARTIFICIAL INTELLIGENCE

Edited By

Parminder Kaur

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

SYLLABUS

Artificial Intelligence

Objectives: To enable the student to understand technicalities of intelligence, capturing and generating knowledge, knowledge representation methodologies, Natural language processing. Student will also learn Fuzzy Logic with their applications and an Artificial intelligence language Prolog.

Sr. No.	Description
1.	***Introduction and Overview:*** Meaning of AI, The AI Problems, Task Domains, AI Technique, Criteria for Success
2.	***Problems, Problem Spaces & Search:*** Defining The Problem as a State Space Search, Production Systems – BFS, DFS, Heuristic Search, Problem & Production System Characteristics, Issues in the Design of Search Programs, Common AI Problems
3.	***Heuristic Search Techniques:*** Generate & Test, Hill Climbing, Best First Search, Constraint Satisfaction, Means-End Analysis
4.	***Knowledge Representation:*** General Concepts of Knowledge, Approaches of Knowledge Representation, Predicate Logic to Represent Knowledge, Resolution, Unification algorithm
5.	***Knowledge Representation using Rules:*** Procedural vs Declarative Knowledge, Logic Programming, Forward vs Backward Reasoning, Matching & Control Knowledge
6.	***Symbolic Reasoning Under Uncertainty:*** Nonmonotonic Reasoning ***Statistical Reasoning:*** Probability & Bayes Theorem, Certainty Factors and Rule Based Systems, Bayesian N/W, Fuzzy Logic and applications
7.	***Weak Slot and Filler Structures:*** Semantic Nets, Frames ***Strong Slot and Filler Structures:*** Conceptual Dependency, Scripts
8.	***Natural Language Processing:*** Introduction, Steps, Syntactic Processing, Semantic Analysis, Discourse & Pragmatic Processing, Spell Checking
9.	***Learning:*** Meaning, Rote Learning, Learning by taking Advice, Learning from examples, Explanation-Based learning, Expert Systems & Its Architecture, Speech Recognition
10.	***Prolog:*** Introduction, Converting English to Prolog Facts and Rules, Goals, Prolog Terminology, Variables, Control Structures, Arithmetic operators, Matching, Backtracking, Lists, Input/Output and Streams

CONTENT

Unit 1:	Introduction and Overview *Parminder Kaur, Lovely Professional University*	1
Unit 2:	Problems, Problem Spaces and Search *Parminder Kaur, Lovely Professional University*	15
Unit 3:	Common AI Problems *Parminder Kaur, Lovely Professional University*	31
Unit 4:	Heuristic Search Techniques *Manish Kumar, Lovely Professional University*	44
Unit 5:	Knowledge Representation *Parminder Kaur, Lovely Professional University*	61
Unit 6:	Knowledge Representation using Rules *Parminder Kaur, Lovely Professional University*	79
Unit 7:	Symbolic Reasoning under Uncertainty *Parminder Kaur, Lovely Professional University*	90
Unit 8;	Statistical Reasoning *Dinesh Kumar, Lovely Professional University*	100
Unit 9:	Weak Slot and Filler Structures *Dinesh Kumar, Lovely Professional University*	113
Unit 10:	Strong Slot and Filler Structures *Dinesh Kumar, Lovely Professional University*	127
Unit 11:	Natural Language Processing *Dinesh Kumar, Lovely Professional University*	139
Unit 12:	Learning *Parminder Kaur, Lovely Professional University*	158
Unit 13:	Expert Systems and its Architecture *Parminder Kaur, Lovely Professional University*	172
Unit 14:	Prolog *Dinesh Kumar, Lovely Professional University*	193

Notes

Unit 1: Introduction and Overview

CONTENTS

Objectives
Introduction
1.1 Overview
1.1.1 History of AI
1.2 Meaning of Artificial Intelligence (AI)
1.2.1 What is Intelligence?
1.2.2 Some Short Definitions of AI
1.3 AI Problems and Task Domains
1.3.1 AI Problems
1.3.2 Task Domains
1.4 AI Technique
1.5 Criteria for Success
1.6 Summary
1.7 Keywords
1.8 Review Questions
1.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Get the introduction and overview of artificial intelligence
- Discuss the meaning of artificial intelligence
- Recognize the AI problems
- Discuss the task domains and AI technique
- Discuss the criteria for success

Introduction

Artificial Intelligence (AI) is the area of computer science focusing on creating machines that can engage on behaviors that humans consider intelligent. The ability to create intelligent machines has intrigued humans since ancient times, and today with the advent of the computer and 50 years of research into AI programming techniques, the dream of smart machines is becoming a reality. Researchers are creating systems which can mimic human thought, understand speech, beat the best human chess player, and countless other feats never before possible. Find out how the military is applying AI logic to its hi-tech systems, and how in the near future Artificial Intelligence may impact our lives. Look at The History of Artificial Intelligence.

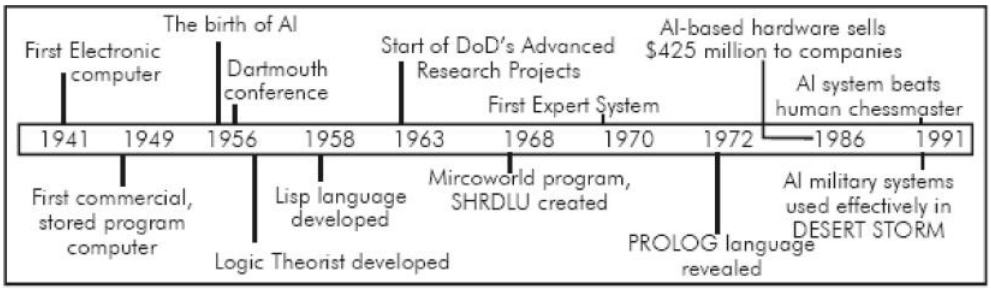
Notes

1.1 Overview

1.1.1 History of AI

Evidence of Artificial Intelligence folklore can be traced back to ancient Egypt, but with the development of the electronic computer in 1941, the technology finally became available to create machine intelligence. The term artificial intelligence was first coined in 1956, at the Dartmouth conference, and since then Artificial Intelligence has expanded because of the theories and principles developed by its dedicated researchers. Through its short modern history, advancement in the fields of AI have been slower than first estimated, progress continues to be made. From its birth 4 decades ago, there have been a variety of AI programs, and they have impacted other technological advancements.

Figure 1.1: History of AI



The Era of the Computer

In 1941 an invention revolutionized every aspect of the storage and processing of information. That invention, developed in both the US and Germany was the electronic computer. The first computers required large, separate air-conditioned rooms, and were a programmer's nightmare, involving the separate configuration of thousands of wires to even get a program running.

The 1949 innovation, the stored program computer, made the job of entering a program easier, and advancements in computer theory lead to computer science, and eventually Artificial intelligence.

Caution With the invention of an electronic means of processing data, came a medium that made AI possible.

Figure 1.2: Old Era of Computer

Notes

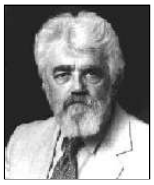
The Beginnings of AI

Norbert Wiener

Although the computer provided the technology necessary for AI, it was not until the early 1950's that the link between human intelligence and machines was really observed. Norbert Wiener was one of the first Americans to make observations on the principle of feedback theory.

Example: The most familiar example of feedback theory is the thermostat. It controls the temperature of an environment by gathering the actual temperature of the house, comparing it to the desired temperature, and responding by turning the heat up or down.

What was so important about his research into feedback loops was that Wiener theorized that all intelligent behavior were the result of feedback mechanisms that could possibly be simulated by machines. This discovery influenced much of early development of AI. In late 1955, Newell and Simon developed *The Logic Theorist,* considered by many to be the first AI program. The program, representing each problem as a tree model, would attempt to solve it by selecting the branch that would most likely result in the correct conclusion. The impact that the logic theorist made on both the public and the field of AI has made it a crucial stepping stone in developing the AI field.


John McCarthy

In 1956 John McCarthy regarded as the father of AI, organized a conference to draw the talent and expertise of others interested in machine intelligence for a month of brainstorming. He invited them to Vermont for "The Dartmouth summer research project on artificial intelligence." From that point on, because of McCarthy, the field would be known as Artificial intelligence. Although not a huge success, the Dartmouth conference did bring together the founders in AI, and served to lay the groundwork for the future of AI research.

Knowledge Expansion

In the seven years after the conference, AI began to pick up momentum. Although the field was still undefined, ideas formed at the conference were re-examined, and built upon. Centers for AI research began forming at Carnegie Mellon and MIT, and a new challenge was faced: further research was placed upon creating systems that could efficiently solve problems, by limiting the search, such as the Logic Theorist. And second, making systems that could learn by themselves.

In 1957, the first version of a new program The General Problem Solver (GPS) was tested. The program developed by the same pair which developed the Logic Theorist. The GPS was an extension of Wiener's feedback principle, and was capable of solving a greater extent of common sense problems. A couple of years after the GPS, IBM contracted a team to research artificial intelligence. Herbert Gelerneter spent three years working on a program for solving geometry theorems. While more programs were being produced, McCarthy was busy developing a major breakthrough in AI history. In 1958 McCarthy announced his new development; the LISP language, which is still used today. LISP stands for LISt Processing, and was soon adopted as the language of choice among most AI developers.

Figure 1.3: Department of Defense Advance Research Project

Notes

In 1963 MIT received a 2.2 million dollar grant from the United States government to be used in researching Machine-Aided Cognition (artificial intelligence). The grant by the Department of Defense's Advanced Research Projects Agency (ARPA), to ensure that the US would stay ahead of the Soviet Union in technological advancements. The project served to increase the pace of development in AI research, by drawing computer scientists from around the world, and continues funding.

Multitude of Programs

The next few years showed a multitude of programs, one notably was SHRDLU. SHRDLU was part of the micro-worlds project, which consisted of research and programming in small worlds (such as with a limited number of geometric shapes). The MIT researchers headed by Marvin Minsky, demonstrated that when confined to a small subject matter, computer programs could solve spatial problems and logic problems. Other programs which appeared during the late 1960's were STUDENT, which could solve algebra story problems, and SIR which could understand simple English sentences. The result of these programs was a refinement in language comprehension and logic.

Another advancement in the 1970's was the advent of the expert system. Expert systems predict the probability of a solution under set conditions.

Example: Because of the large storage capacity of computers at the time, expert systems had the potential to interpret statistics, to formulate rules. And the applications in the market place were extensive, and over the course of ten years, expert systems had been introduced to forecast the stock market, aiding doctors with the ability to diagnose disease, and instruct miners to promising mineral locations. This was made possible because of the systems ability to store conditional rules, and a storage of information.

During the 1970's, many new methods in the development of AI were tested, notably Minsky's frames theory. Also David Marr proposed new theories about machine vision, for instance, how it would be possible to distinguish an image based on the shading of an image, basic information on shapes, color, edges, and texture. With analysis of this information, frames of what an image might be could then be referenced. Another development during this time was the PROLOGUE language. The language was proposed for 1972.

During the 1980's AI was moving at a faster pace, and further into the corporate sector. In 1986, US sales of AI-related hardware and software surged to $425 million. Expert systems in particular demand because of their efficiency. Companies such as Digital Electronics were using XCON, an expert system designed to program the large VAX computers. DuPont, General Motors, and Boeing relied heavily on expert systems Indeed to keep up with the demand for the computer experts, companies such as Teknowledge and Intellicorp specializing in creating software to aid in producing expert systems formed. Other expert systems were designed to find and correct flaws in existing expert systems.

Transition from Lab to Life

The impact of the computer technology, AI included was felt. No longer was the computer technology just part of a select few researchers in laboratories. The personal computer made its debut along with many technological magazines. Such foundations as the American Association for Artificial Intelligence also started. There was also, with the demand for AI development, a push for researchers to join private companies. 150 companies such as DEC which employed its AI research group of 700 personnel, spend $1 billion on internal AI groups. Other fields of AI also made there way into the marketplace during the 1980's. One in particular was the machine

Notes

vision field. The work by Minsky and Marr were now the foundation for the cameras and computers on assembly lines, performing quality control. Although crude, these systems could distinguish differences shapes in objects using black and white differences. By 1985 over a hundred companies offered machine vision systems in the US, and sales totaled $80 million. The 1980's were not totally good for the AI industry. In 1986-87 the demand in AI systems decreased, and the industry lost almost a half of a billion dollars. Companies such as Teknowledge and Intellicorp together lost more than $6 million, about a third of there total earnings. The large losses convinced many research leaders to cut back funding. Another disappointment was the so called "smart truck" financed by the Defense Advanced Research Projects Agency. The projects goal was to develop a robot that could perform many battlefield tasks. In 1989, due to project setbacks and unlikely success, the Pentagon cut funding for the project.

Despite these discouraging events, AI slowly recovered. New technology in Japan was being developed. Fuzzy logic, first pioneered in the US has the unique ability to make decisions under uncertain conditions. Also neural networks were being reconsidered as possible ways of achieving Artificial Intelligence. The 1980's introduced to its place in the corporate marketplace, and showed the technology had real life uses, ensuring it would be a key in the 21st century.

AI Put to the Test

The military put AI based hardware to the test of war during Desert Storm. AI-based technologies were used in missile systems, heads-up-displays, and other advancements. AI has also made the transition to the home. With the popularity of the AI computer growing, the interest of the public has also grown. Applications for the Apple Macintosh and IBM compatible computer, such as voice and character recognition have become available. Also AI technology has made steadying camcorders simple using fuzzy logic. With a greater demand for AI-related technology, new advancements are becoming available. Inevitably Artificial Intelligence has, and will continue to affecting our lives.

Figure 1.4: AI in Test

As a result of research done by Willi Bruns Artificial Intelligence (AI) using computer programs has the ability to solve complex problems of the real world with logical calculus and not with functional analysis. The AI is engaged in the development of computer programs to make computers more "intelligent". Its research has two aims:

1. To make machines and calculating processes more user-friendly and
2. To analyze intelligent behavior.

AI methods try to copy human behavior, that is they attempt to recreate human errors and experiences to produce logical conclusions. Another difference to normal programs is the fact that they do use less functions, compared to the classical numerical mathematics, but much more

Notes

text, symbols and logical calculus. Therefore, AI can be used for solving problems which do not have exact solutions or where solutions can be only gained with an enormous effort, whereby it can happen that false answers are tolerated which happens with human problem solving.

Another fundamental indication of the AI is *the heuristic search*. Generally in AI empiric rules exist, called heuristics, to shorten the search as the number of possible solutions for complex problems can be very high. Another aspect is the fact that traditional computer programs in contrast to AI programs are very difficult to modify, because both the control and the expert knowledge are integrated in the same system. Modifications made in a part of the program have to be carefully examined on their influence and consequences in other parts of the program.

Notes AI programs, expert knowledge is used overlapping, between intelligence and knowledge, where no correlation exists, it must be possible to use the knowledge for the search. In this case it is common to divide the knowledge from the mechanism controlling the search to make sure that modifications of the knowledge only require modifications of the knowledge base.

Task Discuss the various advancements occurred during the evolution of artificial intelligence.

Self Assessment

Fill in the blanks:

1. In 1956 ..... regarded as the father of AI, organized a conference to draw the talent and expertise of others interested in machine intelligence for a month of brainstorming.
2. In 1957, the first version of a new program ..... was tested.
3. Generally in AI empiric rules exist, called ....., to shorten the search as the number of possible solutions for complex problems can be very high.

1.2 Meaning of Artificial Intelligence (AI)

Artificial Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans. AI has had some success in limited, or simplified, domains. However, the five decades since the inception of AI have brought only very slow progress, and early optimism concerning the attainment of human-level intelligence has given way to an appreciation of the profound difficulty of the problem.

1.2.1 What is Intelligence?

Quite simple human behaviour can be intelligent yet quite complex behaviour performed by insects is unintelligent. What is the difference? Consider the behavior of the digger wasp, *Sphex ichneumoneus*. When the female wasp brings food to her burrow, she deposits it on the threshold, goes inside the burrow to check for intruders, and then if the coast is clear carries in the food. The unintelligent nature of the wasp's behavior is revealed if the watching experimenter moves the food a few inches while the wasp is inside the burrow checking. On emerging, the wasp repeats the whole procedure: she carries the food to the threshold once again, goes in to look around,

Notes

and emerges. She can be made to repeat this cycle of behavior upwards of forty times in succession. Intelligence – conspicuously absent in the case of *Sphex* – is the ability to adapt one's behavior to fit new circumstances. Research in AI has focused chiefly on the following components of intelligence: learning, reasoning, problem-solving, perception, and language-understanding.

Caution Mainstream thinking in psychology regards human intelligence not as a single ability or cognitive process but rather as an array of separate components.

1.2.2 Some Short Definitions of AI

There are following definition which describes the term Artificial Intelligence:

- The branch of computer science that deal with writing computer programs that can solve problems creatively.
- Artificial Intelligence (AI) is both the intelligence of machines and the branch of computer science which aims to create it.
- Artificial Intelligence is a series of albums by Warp Records released in the early 1990s to exhibit the capabilities and sounds of electronic music. Warp described the new (post-rave electronic) music as "electronic listening music" to clarify that it was meant more for the mind than the body.
- The history of artificial intelligence begins in antiquity, with myths, stories and rumors of beings built by craftsman and endowed with intelligence and consciousness.
- Intelligence exhibited by an artificial (non-natural, man-made) entity; The branch of computer science dealing with the reproduction or mimicking of human-level thought in computers; The essential quality of a machine which thinks in a manner similar to or on the same general level as a human being
- This refers to an algorithm or set of algorithms that can make decisions in a logical way. For example, the AI routine for a bad guy in a game.
- Occurs when analysis and the search for truth takes precedence over the creative and human activities of a job. People who practice artificial intelligence behave with so much thinking and analysis that the feeling, intuition, and art of making decisions are sacrificed.
- Research into spacecraft autonomy, emerging properties of complex systems, automated system design and in general on the applications of the methods developed by the AI community to problems related to space system.
- Tools that exhibit human intelligence and behavior including self-learning robots, expert systems, and voice recognition, natural and automated translation.
- Information processing by mimicking or simulation of the cerebral, nervous or cognitive processes.
- Applies to a computer system that is able to operate in a manner similar to that of human intelligence; that is, it can understand natural language and is capable of solving problems, learning, adapting, recognizing, classifying, self-improvement, and reasoning.
- The branch of computer science that attempts to program computers to respond as if they were thinking – capable of reasoning, adapting to new situations, and learning new skills.

Notes

- A generic term commonly used to indicate the inclusion in software of some type of automated application of rules, the results of which give the appearance of "intelligence" on the part of the computer.
- The development of computers that 'think' and respond like humans.
- Computational techniques to automate tasks that require human intelligence and the ability to reason.
- A branch of information science aiming at computational models of human cognition, such as expert systems.
- An algorithm by which the computer gives the illusion of thinking like a human. Also, the action of a character in a game as it reacts to other objects in the game.
- The discipline of building special computer systems that can perform complex activities usually only performed by humans.
- The concept of making computers do tasks once considered to require thinking.
- AI makes computers play chess and recognize handwriting and speech. Return to the top.
- A branch of computer science whose goal is the design of machines that have attributes associated with human intelligence, such as learning, reasoning, vision, understanding speech, and, ultimately, consciousness.
- Field of study concerned with producing computer programs capable of learning and processing their own 'thoughts'.
- A growing set of computer problem-solving techniques being developed to imitate human thought or decision making processes.
- Default behavior of units, programmed into the game, to give basic reactions and potentially to simulate a confrontation with another player.
- It is the study and design of intelligent agents, where an intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success.
- An assembler is a program that takes basic computer instructions and converts them into a pattern of bits (binary digits).
- The automation of human cognitive skills through rules and such things as recognizing speech or visual images, solving problems, or making medical diagnoses.
- The ability of a machine system such as the ASAP software to perceive anticipated or unanticipated new conditions, decide what actions must be performed under the conditions, and plan the actions accordingly.
- A set of code or algorithms designed to simulate the actions of an intelligent being – such as a human or animal (Tamagochi)
- A broad term describing the field of developing computer programs to simulate human thought processes and behaviors.

Self Assessment

Fill in the blanks:

4. ..... is usually defined as the science of making computers do things that require intelligence when done by humans.

Notes

5. ..... in AI has focused chiefly on the following components of intelligence: learning, reasoning, problem-solving, perception, and language-understanding.
6. A generic term commonly used to indicate the inclusion in software of some type of automated application of rules, the results of which give the appearance of "....." on the part of the computer.

1.3 AI Problems and Task Domains

1.3.1 AI Problems

When learning the usual range of tasks that we might look forward to an "intelligent entity" to perform, we require to take into account both "commonplace" tasks in addition to expert tasks.

Example: The commonplace tasks comprise:

- Identifying people, objects.
- Communicating (via natural language).
- Finding the way around obstructions on the streets.

These tasks are performed by matter of routinely by people and some other animals. Specialist tasks comprise:

- Medical diagnosis.
- Mathematical problem solving
- Playing games such as chess

Did u know? These tasks cannot be performed by all people, and can only be carried out by skilled experts.

Now, which of these tasks are simple and which ones are tough? Evidently tasks of the first type are simple for humans to carry out, and approximately all are able to perform them. The second range of tasks needs skill development and/or intelligence and only some expert scan execute them well.

Notes Though, when we see what computer systems have been able to attain to date, we observe that their accomplishments comprise performing complicated tasks such as medical diagnosis, carrying out symbolic integration, proving theorems and playing chess.

Alternatively, it has proved to be very firm to make computer systems achieve many routine tasks that all humans and many animals can do.

Example: Examples of such tasks comprise navigating our route without running into things, catching prey and averting predators.

Humans and animals are also competent of interpreting multifaceted sensory information. We are able to identify objects and people from the visual picture that we obtain. We are also able to carry out compound social functions.

Notes

Intelligent Behaviour

This leads us to the question of what represents intelligent behaviour. Some of these tasks and applications are:

- Perception including image recognition and computer vision
- Reasoning
- Learning
- Understanding language including natural language processing, speech processing
- Solving problems
- Robotics.

1.3.2 Task Domains

Task Domains include:

- Formal tasks (matematika, games)
- Mundane task (perception, robotics, natural language, common sense, reasoning)
- Expert tasks (financial analysis, medical diagnostics, engineering, scientific analysis, dll)

Self Assessment

Fill in the blanks:

7. When learning the usual range of tasks that we might look forward to an "intelligent entity" to perform, we require to take into account both "commonplace" tasks in addition to ..... tasks.
8. ..... tasks are performed by matter of routinely by people and some other animals.

1.4 AI Technique

As there are many diverse artificial intelligence techniques that have been generated, with new ways being produced, a few forms of Artificial Intelligence (AI) have turn out to be more and more well-liked. Some of the most general techniques comprise the use of neural networks and the expansion of expert systems. These dissimilar artificial intelligence techniques can be accessed to build up dissimilar forms of AI, typically depending on the amount of "thinking" the program can really do, and these are called as either "strong AI" or "weak AI."

Artificial intelligence techniques are the ways that can be used to generate and produce computer programs generally observed as forms of artificial intelligence. Usually, artificial intelligence points to a program that is able to imitate or re-create the consideration processes illustrated by the human brain. This typically comprises solving problems, making observations or obtaining input for utilization in analysis or problem solving, and the aptitude to classify and identify dissimilar objects and the properties of those objects.

There are many dissimilar artificial intelligence methods that can be used by an AI programmer, though two of the most general are neural networks and expert systems. Neural networks are computer programs intended about the cognitive processes utilized by the human brain. Basically, a neural network includes layers of categorization and ways by which objects can be recognized

Notes

and categorized. This is analogous to the idea of plan in human cognition, which permits people to recognize objects based on properties of those objects. New information offered to the neural network can then be analyzed and recognized based on formerly inputted criteria, permitting the system to "learn" new categories and recognize known or unknown objects.

Expert systems are artificial intelligence techniques constructed around logic and "if/then" statements. This typically includes a great deal of information that is "taught" to the computer system, which then makes the system an expert in a specific field. When new input is commenced in, such as a request for processing financial reports, the expert system can examine the information by means of these if/then statements to restrict the output response.

These different artificial intelligence techniques can be utilized to build up systems that are considered either "strong AI" or "weak AI." Strong AI systems are those that most completely look for to emulate human thought and cognitive potentials through a broad range of functions. Artificial intelligence techniques that build up weak AI systems are narrower in focus, and look for to imitate only a single function or facet of human intelligence.

Did u know? Strong AI systems can examine new information and offer output that potentially goes beyond the limitations of input data.

Task Illustrate the function of neural networks.

Self Assessment

Fill in the blanks:

9. ..... are the ways that can be used to generate and produce computer programs generally observed as forms of artificial intelligence.
10. ..... are computer programs intended about the cognitive processes utilized by the human brain.
11. ..... are artificial intelligence techniques constructed around logic and "if/then" statements.
12. ..... systems are those that most completely look for to emulate human thought and cognitive potentials through a broad range of functions.
13. Artificial intelligence techniques that build up ..... systems are narrower in focus, and look for to imitate only a single function or facet of human intelligence.

1.5 Criteria for Success

Criteria for success includes:

- ***Long-term:*** Turing Test (for Weak AI)
  - As suggested by Alan Turing (1950), if a computer can make people believe it is human (i.e., intelligent) by means of an unobstructed conversation, then it is intelligent.
  - Turing forecasted fully intelligent machines by 2000, not even close
  - Loebner Prize competition, tremendously controversial

Notes

- *Short-term:* More modest success in restricted domains
  - Performance equal or better than humans, game playing (Deep Blue), expert systems (MYCIN)
  - Real-world expediency $$$e.g., expert systems (XCON, Prospector), fuzzy logic (cruise control)

Self Assessment

Fill in the blanks:

14. The long-term criteria include ..... Test performed for Weak AI.
15. In short-term criteria, performance is equal or better than ..... .

1.6 Summary

- Artificial Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans.
- The term artificial intelligence was first coined in 1956, at the Dartmouth conference, and since then Artificial Intelligence has expanded because of the theories and principles developed by its dedicated researchers.
- AI-based technologies were used in missile systems, heads-up-displays, and other advancements.
- Research in AI has focused chiefly on the following components of intelligence: learning, reasoning, problem-solving, perception, and language-understanding.
- Artificial Intelligence (AI) is both the intelligence of machines and the branch of computer science which aims to create it.
- When learning the usual range of tasks that we might look forward to an "intelligent entity" to perform, we require to take into account both "commonplace" tasks in addition to expert tasks.
- Some of the most general techniques comprise the use of neural networks and the expansion of expert systems.
- Artificial intelligence techniques are the ways that can be used to generate and produce computer programs generally observed as forms of artificial intelligence.
- Strong AI systems are those that most completely look for to emulate human thought and cognitive potentials through a broad range of functions.
- Artificial intelligence techniques that build up weak AI systems are narrower in focus, and look for to imitate only a single function or facet of human intelligence.

1.7 Keywords

AI Technique: Artificial intelligence techniques are the ways that can be used to generate and produce computer programs generally observed as forms of artificial intelligence.

Artificial Intelligence (AI): Artificial Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans.

Notes

Strong AI Systems: Strong AI systems are those that most completely look for to emulate human thought and cognitive potentials through a broad range of functions.

Weak AI Systems: Artificial intelligence techniques that build up weak AI systems are narrower in focus, and look for to imitate only a single function or facet of human intelligence.

1.8 Review Questions

1. Elucidate in detail the overview of artificial intelligence.
2. Make distinction between human and computer intelligence.
3. Illustrate the concept of AI Problems.
4. Describe the tasks and applications associated with intelligent behavior.
5. Illustrate the concept of task domain.
6. What are artificial intelligence techniques? How are they used to generate and produce computer programs?
7. Make distinction between "strong AI" and "weak AI".
8. What does long-term and short-term criteria mean?
9. Write a short note on the evolution of AI.
10. When the PROLOGUE language came in existence?

Answers: Self Assessment

1.	John McCarthy	2.	The General Problem Solver (GPS)
3.	heuristics	4.	Artificial Intelligence (AI)
5.	Research	6.	intelligence
7.	expert	8.	Common-place
9.	Artificial intelligence techniques	10.	Neural networks
11.	Expert systems	12.	Strong AI
13.	weak AI	14.	Turing
15.	humans		

1.9 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Notes

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

aima.cs.berkeley.edu/

Notes

Unit 2: Problems, Problem Spaces and Search

CONTENTS

Objectives
Introduction
2.1 Defining the Problem as a State Space Search
2.2 Production System
2.2.1 Classes of Production Systems
2.2.2 Advantages of Production Systems
2.2.3 Disadvantages of Production Systems
2.2.4 Partially Commutative Production System
2.3 Breadth-first Search
2.3.1 How does it Work?
2.3.2 Features
2.3.3 Applications
2.4 Depth-first Search
2.4.1 Vertex Orderings
2.4.2 Applications
2.5 Heuristic Search
2.6 Problem & Production System Characteristics and Design of Search Programs Issues
2.6.1 Problem Characteristics
2.6.2 Production System Characteristics
2.6.3 Issues in the Design of Search Programs
2.7 Summary
2.8 Keywords
2.9 Review Questions
2.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the problem as a state space search
- Illustrate the concept of production systems
- Discuss the Breadth-first search, Depth-first search, and Heuristic search
- Identify the problem & production system characteristics
- Understand the Issues in the design of search programs

Notes

Introduction

Problems have the general form given such-and-such data, find x. A huge variety of types of problem is addressed in AI. Some examples are: finding winning moves in board games; identifying people from their photographs; and planning series of movements that enable a robot to carry out a given task. In this unit, you will understand various concepts of problems, problem spaces & search.

2.1 Defining the Problem as a State Space Search

The steps that are needed to make a system to work out a particular problem are:

1. Problem Definition that must include precise specifications of what the initial situation will be as well as what final situations constitute acceptable solutions to the problem.
2. Problem Analysis, this can have immense impact on the appropriateness of varies possible techniques for solving the problem.
3. Selection of the best technique(s) for solving the particular problem.

The thought of State Space Search is broadly used in Artificial Intelligence. The plan is that a problem can be solved by probing the steps which might be taken for the solution. Every action takes the solver to a novel state.

Example: The typical example is of the Farmer who is required to transport a Chicken, a Fox and some Grain across a river separately. The Fox will eat the Chicken if left unconfirmed. Similarly the Chicken will eat the Grain.

Here, the State is illustrated by the positions of the Farmer, Chicken, Fox and Grain. The solver can move among States by making a legal move (which does not consequence in something being eaten). Non-legal moves are not valued analyzing.

The clarification to such a problem is a record of linked States leading from the Initial State to the Goal State. This may be found either by beginning at the Initial State and functioning towards the Goal state or vice-versa.

The necessary State can be functioned towards by either:

- ***Depth-First Search:*** Discovering each strand of a State Space in turn.
- ***Breadth-First Search:*** Discovering every link encountered, analyzing the state space a level at a time.

These techniques usually use lists of:

- ***Closed States:*** States whose links have all been discovered.
- ***Open States:*** States which have been came across, but have not been fully explored.

Supremely, these lists will also be used to avert endless loops.

Example: Take into account the problem of "Playing Chess". To construct a program that could play chess, we have to identify the beginning position of the chess board, the rules that describe legal moves. And the board position that symbolize a win. The objective of the winning the game, if possible, must be made unambiguous.

The beginning position can be illustrated by an 8 × 8 array square in which every element square (x,y), (x ranging from 1 to 8 & y varying from 1 to 8) illustrates the board position of a suitable

Notes

chess coin, the goal is any board position in which the challenger does not have a legal move and his or her "king" is under attack. The legal moves offer the way of receiving from initial state of final state.

The legal moves can be illustrated as a set of rules including two parts: A left side that provides the current position and the right side that illustrates the change to be made to the board position.

Current Position:

While pawn at square (5, 2), and Square (5, 3) is empty, and Square (5, 4) is empty.

Changing Board Position:

Move pawn from Square (5, 2) to Square (5, 4).

The existing position of a coin on the board is its state and the set of all possible states is state space. One or more states where the problem finishes is final state or goal state.

The state space illustration forms the foundation of most of the AI methods. It permits for a formal definition of the problem as the requirement to exchange some given situation into some preferred situation by means of a set of allowable operations. It allows the problem to be solved with the assistance of known techniques and control approaches to move via the problem space until goal state is located.

Therefore problem as State space search can be recapitulated as:

- Formulate a problem as a state space search by displaying the legal problem states, the legal operators, and the initial and goal states.
- A state is defined by the measurement of the values of all attributes of interest in the world.
- An operator modifies one state into the other; it has a precondition which is the value of certain attributes previous to the application of the operator, and a set of effects, which are the attributes changed by the operator.
- The initial state is where you begin.

Task Make distinction between closed states and open states.

Self Assessment

Fill in the blanks:

1. The thought of ..... Search is that a problem can be solved by probing the steps which might be taken for the solution.
2. ..... are the States whose links have all been discovered.
3. A ..... is defined by the measurement of the values of all attributes of interest in the world.

2.2 Production System

A Knowledge demonstration formalism comprises collections of condition-action rules (Production Rules or Operators), a database which is customized in harmony with the rules, and a Production System Interpreter which manages the operation of the rules i.e. The 'control mechanism' of a Production System, formatting the order in which Production Rules are fired.

Notes

A system that utilizes this form of knowledge representation is known as a production system.

A production system includes rules and factors. Knowledge is programmed in a declarative from which includes a set of rules of the form

Situation — — — — — — action

Situation that implies action.

Example: IF the initial state is a goal state THEN quit.

The main components of an AI production system are:

i. A global database

ii. A set of production rules and

iii. A control system

The objective database is the central data structure utilized by an AI production system. The production rules function on the global database. Every rule has a requirement that is either pleased or not by the database. If the requirement is pleased, the rule can be applied. Application of the rule alters the database.

Notes The control system selects which relevant rule should be applied and ceases calculation when a termination condition on the database is pleased. If numerous rules are to fire simultaneously, the control system resolves the disagreements.

2.2.1 Classes of Production Systems

1. A monotonic production system
2. A non monotonic production system
3. A partially commutative production system
4. A commutative production system.

2.2.2 Advantages of Production Systems

1. Production systems offer an exceptional tool for structuring AI programs.
2. Production Systems are extremely modular since the individual rules can be added, removed or modified independently.
3. The production rules are articulated in a natural form, so the statements enclosed in the knowledge base should the recording of an expert thinking out loud.

2.2.3 Disadvantages of Production Systems

One significant difficulty is the fact that it may be very hard to analyze the flow of control inside a production system since the individual rules don't call each other.

Production systems depict the operations that can be carried out in a search for a solution to the problem.

Notes

Monotonic production system: A system in which the application of a rule never averts the later application of another rule, that could have also been applied at the time the first rule was chosen.

2.2.4 Partially Commutative Production System

A production system in which the application of a specific sequence of rules transforms state X into state Y, then any permutation of those rules that is permissible also converts state X into state Y.

Theorem proving falls under monotonic partially communicative system. Blocks world and 8 puzzle problems such as chemical analysis and synthesis come under monotonic, not partially commutative systems. Playing the game of bridge comes under non monotonic, not partially commutative system.

For any problem, numerous production systems exist. Some will be competent than others. However it may appear that there is no association among kinds of problems and kinds of production systems, in practice there is a definite relationship.

Partially commutative, monotonic production systems are functional for solving ignorable problems. These systems are significant for man accomplishment standpoint since they can be executed without the ability to back off to earlier states, when it is exposed that an incorrect path was followed. Such systems augment the efficiency as it is not essential to keep track of the changes done in the search process.

Monotonic partially commutative systems are functional for problems in which alterations take place but can be upturned and in which the order of operation is not decisive (ex: 8 puzzle problem).

Production systems that are not partially commutative are functional for many problems in which permanent alterations happen, such as chemical analysis.

Did u know? When dealing with partially commutative production system, the order in which operations are performed is very significant and therefore correct decisions have to be made at the first time itself.

Self Assessment

Fill in the blanks:

4. A system that utilizes this form of knowledge representation is known as a ..... .
5. Partially commutative, monotonic production systems are functional for solving ..... problems.

2.3 Breadth-first Search

In graph theory, **breadth-first search** (**BFS**) is a graph search algorithm that starts at the root node and discovers all the adjacent nodes. Then for each of those adjoining nodes, it discovers their unknown neighbor nodes, and so on, until it finds the objective.

2.3.1 How does it Work?

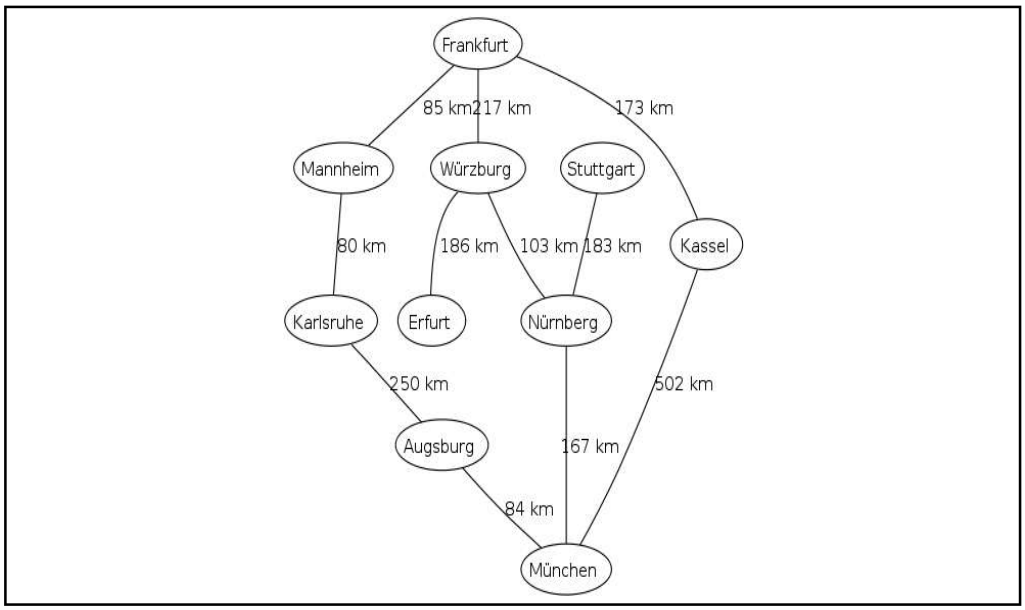
BFS is an uninformed search technique that intends to enlarge and scrutinize all nodes of a graph or amalgamation of sequences by methodically penetrating via every solution. Alternatively, it

Notes

comprehensively looks for the whole graph or sequence without considering the goal until it locates it. It does not make use of a heuristic algorithm.

From the position of the algorithm, all child nodes attained by increasing a node are added to a FIFO (i.e., First In, First Out) queue. In usual implementations, nodes that have not yet been scrutinized for their neighbors are positioned in some container (like a queue or linked list) known as "open" and then once inspected are positioned in the container "closed".

Example: An Example Map of Germany with some Associations among Cities.



Algorithm (Informal)

1. Enqueue the root node.
2. Dequeue a node and examine it.
   (i) If the element wanted is found in this node, suspend the search and return a result.
   (ii) or else enqueue any successors (the direct child nodes) that have not yet been exposed.
3. If the queue is empty, every node on the graph has been scrutinized – quit the search and return "not found".
4. Repeat from Step 2.

Caution Using a stack rather than a queue would turn this algorithm into a depth-first search.

2.3.2 Features

Space Complexity

As all of the nodes of a level must be accumulated until their child nodes in the next level have been produced, the space complexity is comparative to the number of nodes at the deepest level.

Notes

Provided a branching factor b and graph depth d the asymptotic space complexity is the number of nodes at the deepest level, $O(bd)$. When the number of vertices and edges in the graph are recognized ahead of time, the space complexity can also be articulated as $O(|E| + |V|)$ where $|E|$ is the cardinality of the set of edges (the number of edges), and $|V|$ is the cardinality of the set of vertices. In the worst case the graph has a depth of 1 and all vertices must be amassed.

Notes As it is exponential in the depth of the graph, breadth-first search is often unreasonable for large problems on systems with bounded space.

Time Complexity

Since in the worst case breadth-first search has to regard all paths to all probable nodes the time complexity of breadth-first search is $1 + b + b^2 + b^3 + \ldots + b^d$ which is $O(bd)$. The time complexity can also be articulated as $O(|E| + |V|)$ as every vertex and every edge will be discovered in the worst case.

Completeness

Breadth-first search is complete. This shows that if there is a solution, breadth-first search will find it in spite of the type of graph. Though, if the graph is infinite and there is no solution breadth-first hunt will depart.

Proof of Completeness

If the shallowest objective node is at some restricted depth say d, breadth-first search will sooner or later find it after increasing all shallower nodes (given that the branching factor b is finite).

Optimality

For unit-step cost, breadth-first search is best possible. Usually breadth-first search is not best possible as it always returns the consequence with the fewest edges among the start node and the goal node. If the graph is a weighted graph, and as a result has costs connected with each step, a goal next to the beginning does not have to be the cheapest objective obtainable. This problem is solved by enhancing breadth-first search to uniform-cost search which considers the path costs. Nonetheless, if the graph is not weighted, and consequently all step costs are equal, breadth-first search will discover the nearest and the finest solution.

Bias towards Nodes of High Degree

It has been empirically noticed (and analytically exposed for random graphs) that incomplete breadth-first search is prejudiced towards nodes of high degree. This makes a breadth-first search model of a graph very tricky to understand.

Example: For example, a breadth-first sample of 1 million nodes in Facebook (less than 1% of the complete graph) overvalues the average node degree by 240%.

2.3.3 Applications

Breadth-first search can be utilized to explain many problems in graph theory, for instance:

- Locating all nodes within one connected component
- Copying Collection, Cheney's algorithm

Notes

- Locating the shortest path among two nodes u and v
- Testing a graph for bipartiteness
- (Reverse) Cuthill–McKee mesh numbering
- Ford–Fulkerson method for calculating the maximum flow in a flow network
- Serialization/Deserialization of a binary tree vs serialization in sorted order, permits the tree to be reconstructed in a competent manner.

Finding Connected Components

The set of nodes reached by a BFS (breadth-first search) form the associated component enclosing the starting node.

Testing Bipartiteness

BFS can be utilized to test bipartiteness, by beginning the hunt at any vertex and providing alternating labels to the vertices visited all through the search. That is, provide label 0 to the beginning vertex, 1 to all its neighbours, 0 to those neighbours, and so on. If at any step a vertex has (visited) neighbours with the same label as itself, then the graph is not bipartite. If the search ends without such a circumstance occurring, then the graph is bipartite.

Self Assessment

Fill in the blanks:

6. In graph theory, ... is a graph search algorithm that starts at the root node and discovers all the adjacent nodes.
7. From the position of the algorithm, all child nodes attained by increasing a node are added to a ..... queue.
8. BFS can be utilized to test ....., by beginning the hunt at any vertex and providing alternating labels to the vertices visited all through the search.

2.4 Depth-first Search

Depth-first Search (DFS) is an algorithm for navigating or looking for a tree, tree structure, or graph. One begins at the root (choosing some node as the root in the graph case) and discovers as far as possible with each branch before backtracking.

Officially, DFS is an uninformed search that advances by increasing the first child node of the search tree that occurs and therefore going deeper and deeper until a purpose node is found, or until it reaches a node that has no children. Then the search backtracks, recurring to the most recent node it hasn't completed exploring. In a non-recursive accomplishment, all freshly extended nodes are added to a stack for exploration.

The time and space analysis of DFS varies as per its application area. In hypothetical computer science, DFS is usually used to navigate a complete graph, and takes time $O(|V| + |E|)$, linear in the size of the graph. In these applications it also utilizes space $O(|V|)$ in the worst case to accumulate the stack of vertices on the current search path in addition to the set of already-visited vertices. Consequently, in this setting, the time and space bounds are the comparable as for breadth first search and the option of which of these two algorithms to use depends less on their complexity and more on the dissimilar properties of the vertex orderings the two algorithms generate.

Notes

For applications of DFS to search problems in artificial intelligence, though, the graph to be searched is frequently either excessively large to visit in its entirety or even infinite, and DFS may undergo from non-termination when the length of a path in the search tree is infinite. Consequently, the search is only performed to a restricted depth, and because of restricted memory accessibility one naturally does not use data structures that keep track of the set of all formerly visited vertices. Here, the time is still linear in the number of extended vertices and edges (even though this number is not similar as the size of the complete graph since some vertices may be searched more than once and others not at all) but the space complexity of this alternative of DFS is only comparative to the depth limit, much slighter than the space required for searching to the similar depth using breadth-first search. For such applications, DFS also provides itself much better to heuristic methods of selecting a likely-looking branch. When a suitable depth limit is not recognized a priori, iterative deepening depth-first search applies DFS frequently with a sequence of rising limits; in the artificial intelligence mode of analysis, with a branching factor superior than one, iterative deepening enhances the running time by only a constant factor over the case in which the accurate depth limit is recognized due to the geometric growth of the number of nodes for each level.

2.4.1 Vertex Orderings

It is also probable to utilize the depth-first search to linearly order the vertices of the original graph (or tree). There are three general methods of performing this:

- A **preordering** is a list of the vertices so that they were first visited by the depth-first search algorithm. This is a compact and usual manner of illustrating the development of the search. A preordering of an expression tree is the expression in Polish notation.
- A **postordering** is a list of the vertices so that they were *last* visited by the algorithm. A postordering of an expression tree is the expression in reverse Polish notation.
- A **reverse postordering** is the reverse of a postordering, which is defined a list of the vertices in the conflicting order of their previous visit. When probing a tree, reverse postordering is similar as preordering, but generally they are dissimilar when looking for a graph.

Task Make distinction between preordering and postordering.

2.4.2 Applications

Algorithms that utilize depth-first search as a building block comprise:

- Locating connected components.
- Topological sorting.
- Finding 2-(edge or vertex)-connected components.
- Locating strongly connected components.
- Planarity Testing
- Solving puzzles with only one solution, like mazes. (DFS can be adapted to discover all solutions to a maze by only involving nodes on the present path in the visited set.)
- bi connectivity.

Notes

Self Assessment

Fill in the blanks:

9. ..... is an algorithm for navigating or looking for a tree, tree structure, or graph.
10. A ..... is a list of the vertices so that they were first visited by the depth-first search algorithm.
11. A reverse postordering is the reverse of a postordering, which is defined a list of the vertices in the ..... order of their previous visit.

2.5 Heuristic Search

Due to the momentary situation changes and new possibilities arising for further decisions during the problem solving, new ramifications are needed. For this reason, problem solving the AI is best represented with the help of a tree. It starts with a beginning condition and after each decision follows a ramification. The final number of ramifications at the end of a tree can be very high, if many steps are necessary for solving the problem. Therefore, empiric rules, so called heuristics, were developed to facilitate the search for the most likely ramifications and also to limit them.

Heuristic Search resolve multifaceted problems competently, it is essential to negotiate the needs of the movability and systematically. A control structure has to be constructed that no longer assures the best solution, but that will almost always discover a very superior answer. Such a method is said to be heuristic (rule of thumb). A heuristic search enhances competently the search process, but sacrifices the claims of wholeness. But they perk up the class of the paths that are discovered.

Caution Using good heuristics we can obtain good solutions to tough problems, like the traveling salesman problem.

Self Assessment

Fill in the blank:

12. A ..... search enhances competently the search process, but sacrifices the claims of wholeness.

2.6 Problem & Production System Characteristics and Design of Search Programs Issues

2.6.1 Problem Characteristics

Heuristic search is an extremely common method pertinent to a large class of problem. It involves numerous techniques. In order to select a appropriate method, it is essential to examine the problem concerning the following deliberations.

1. ***Is the problem decomposable?***

   A very great and compound problem can be simply solved if it can be wrecked into smaller problems and recursion could be used. Presume that we want to solve.

Notes

+" x2 + 3x+sin2x cos 2x dx

This can be performed by dividing it into three lesser problems and solving each by applying particular rules. Adding the results the whole solution is attained.

2. ***Can solution steps be ignored or undone?***

   Problem occurs under three classes ignorable, recoverable and irrecoverable. This categorization is pertaining to the steps of the solution to a problem.

Example: Consider theorem proving. We may later discover that it is of no aid. We can still continue further, as nothing is lost by this outmoded step. This is an example of ignorable solutions steps.

Now consider the 8 puzzle problem tray and arranged in particular order. While moving from the begin state towards objective state, we may make some dull move and consider theorem proving. We may carry on by first proving lemma. But we may backtrack and untie the unnecessary move. This only includes additional steps and the explanation steps are recoverable.

Finally consider the game of chess. If a wrong move is done, it can neither be ignored nor be recovered. The thing to do is to make the best utilization of present situation and proceed. This is an example of an irrecoverable solution steps.

1. Ignorable problems ***Ex:*** theorem proving
   (i) In which solution steps can be ignored.
2. Recoverable problems ***Ex:*** 8 puzzle
   (i) In which solution steps can be undone
3. Irrecoverable problems ***Ex:*** Chess
   (i) In which solution steps can't be undone

   A knowledge of these will assist in identifying the control structure.

3. ***Is the Universal Predictable?***

   Problems can be categorized into those with firm result (eight puzzle and water jug problems) and those with unsure result (playing cards) in certain outcome problems, planning could be made to produce a sequence of operators that assures to lead to a solution. Planning assists to evade unwanted solution steps. For unsure result problems, planning can at best produce a sequence of operators that has a good likelihood of approaching to a solution. The uncertain result problems do not assures a solution and it is frequently very expensive as the number of solution and it is frequently very expensive as the number of solution paths to be discovered enhances exponentially with the number of points at which the result can not be predicted. Therefore one of the hardest types of problems to resolve is the irrecoverable, uncertain outcome problems (***Ex:*** Playing cards).

4. ***Is good solution absolute or relative ? (Is the solution a state or a path?)***

   There are two groups of problems. In one, like the water jug and 8 puzzle problems, we are content with the solution, unaware of the solution path taken, while in the other group not just any solution is suitable. We want the finest, like that of traveling sales man problem, where it is the shortest path. In any path problems, by heuristic methods we get hold of a solution and we do not discover alternatives.

Notes

Did u know? For the best-path problems all probable paths are discovered by means of a comprehensive search until the best path is attained.

5. ***The knowledge base consistent?***

   In some problems the knowledge base is reliable and in some it is not.

Example: Consider the case when a Boolean expression is assessed. The knowledge base now includes theorems and laws of Boolean Algebra which are forever true. In contrast consider a knowledge base that includes facts regarding production and cost. These keep ranging with time.

Thus many reasoning schemes that function well in consistent domains are not suitable in conflicting domains.

Example: Boolean expression evaluation.

6. ***What is the role of Knowledge?***

   However one could have limitless computing power, the size of the knowledge base obtainable for solving the problem does matter in reaching at a good solution.

   Take for example the game of playing chess, just the rues for identifying legal moves and some easy control mechanism is sufficient to arrive at a solution. But additional knowledge about good strategy and tactics could aid to restrain the search and accelerate the implementation of the program. The solution would then be realistic.

   Consider the case of guessing the political trend. This would need a massive amount of knowledge even to be able to identify a solution, leave alone the best.

Example: Playing chess 2. News paper understanding

7. ***Does the task requires interaction with the person?***

   The problems can again be classified under two heads.

   (i) Solitary in which the computer will be specified a problem explanation and will create an answer, with no in-between communication and with he demand for a clarification of the reasoning process. Simple theorem proving occurs under this group provided the basic rules and laws, the theorem could be proved, if one occurs.

Example: Theorem proving (give basic rules & laws to computer)

   (ii) Conversational, in which there will be in-between communication among a person and the computer, wither to give additional aid to the computer or to give additional informed information to the user, or both problems like medical diagnosis comes under this group, where people will be unwilling to recognize the decision of the program, if they can not follow its reasoning.

Example: Problems such as medical diagnosis.

8. ***Problem Classification:*** Definite problems are inspected from the point of view, the task here is scrutinize an input and decide which of a set of recognized classes.

Notes

Example: Problems like medical diagnosis, engineering design.

2.6.2 Production System Characteristics

1. Can production systems, such as problems, be illustrated by a set of traits that shed some light on how they can simply be implemented?
2. If so, what relationships are there among problem types and the kinds of production systems best matched to solving the problems?
   (i) *Classes of Production systems:*
      (a) Monotonic Production System: The application of a rule by no means averts the later application of another rule that could also have been functional at the time the first rule was chosen.
      (b) Non-monotonic Production system
      (c) Partially commutative Production system: Property that if application of a specific sequence of rules converts state x to state y, then permutation of those rules permissible, also converts state x into state y.
      (d) Commutative Production system

2.6.3 Issues in the Design of Search Programs

1. The direction in which to carry out the search (forward versus backward reasoning)
2. How to choose applicable rules (Matching)
3. How to symbolize every node of the hunt process (knowledge demonstration problem)

Self Assessment

Fill in the blanks:

13. A very great and compound problem can be simply solved if it can be wrecked into smaller problems and ..... could be used.
14. ..... is the application of a rule by no means averts the later application of another rule that could also have been functional at the time the first rule was chosen.
15. The problem is said to be ..... when the computer will be specified a problem explanation and will create an answer, with no in-between communication and with he demand for a clarification of the reasoning process.

2.7 Summary

- Problems have the general form given such-and-such data, find x.
- The thought of State Space Search is broadly used in Artificial Intelligence. The plan is that a problem can be solved by probing the steps which might be taken for the solution.
- The state space illustration permits for a formal definition of the problem as the requirement to exchange some given situation into some preferred situation by means of a set of allowable operations.

Notes

- A system that utilizes this form of knowledge representation is known as a production system.
- A production system includes rules and factors. Knowledge is programmed in a declarative from which includes a set of rules of the form.
- A production system in which the application of a specific sequence of rules transforms state X into state Y, then any permutation of those rules that is permissible also converts state x into state Y.
- In graph theory, breadth-first search (BFS) is a graph search algorithm that starts at the root node and discovers all the adjacent nodes.
- Depth-first search (DFS) is an algorithm for navigating or looking for a tree, tree structure, or graph.
- Heuristic Search resolve multifaceted problems competently, it is essential to negotiate the needs of the movability and systematically.

2.8 Keywords

Breadth-first Search (BFS): In graph theory, breadth-first search (BFS) is a graph search algorithm that starts at the root node and discovers all the adjacent nodes.

Depth-first Search (DFS): It is an algorithm for navigating or looking for a tree, tree structure, or graph.

Heuristic Search: It resolve multifaceted problems competently, it is essential to negotiate the needs of the movability and systematically.

Production System: A system that utilizes this form of knowledge representation is known as a production system.

State Space Search: In this, the plan is that a problem can be solved by probing the steps which might be taken for the solution.

2.9 Review Questions

1. How do you define a problem as a State space search? Illustrate.
2. Explain the concept of production system with example.
3. Elucidate various classes of production systems.
4. What are the advantages and disadvantages of production system?
5. Describe the working of Breadth-first search with example.
6. Explain the features and applications of Breadth-first search.
7. What is Depth First Search? Illustrate its working with example.
8. How do you linearly order the vertices of the original graph (or tree)? Discuss the methods.
9. Illustrate how does heuristic search resolves multifaceted problems.
10. Enlighten the various characteristics of Problem & Production system.

Notes

Answers: Self Assessment

1. State Space
2. Closed State
3. state
4. production system
5. ignorable
6. breadth-first search (BFS)
7. FIFO (i.e., First In, First Out)
8. bipartiteness
9. Depth-first search (DFS)
10. preordering
11. conflicting
12. heuristic
13. recursion
14. Monotonic Production System
15. Solitary

2.10 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench.* Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Notes

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

intelligence.worldofcomputing.net/ai.../ai-search-techniques.html

Notes

Unit 3: Common AI Problems

CONTENTS

Objectives

Introduction

3.1 Water Jug Problem

3.2 8 Puzzle Problem

3.3 Frame Problem

3.3.1 Problems Related to the Frame Problem

3.4 Epistemological Problems

3.5 Summary

3.6 Keywords

3.7 Review Questions

3.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of water jug problem
- Illustrate the concept of 8 puzzle problem
- Discuss the frame problem and problems relate to frame problem
- Understand the epistemological problems

Introduction

There are various problems concerning artificial intelligence. In this unit, we will discuss Water Jug problem, 8 puzzle problem, frame problem, and Epistemological problems. You will also understand the problems related to frame problem.

3.1 Water Jug Problem

Statement: We are provided with 2 jugs, a 4-litre one and a 3-litre one. Neither contains any gauging markers on it. There is a pump that can be accessed to fill the jugs with water. How can we obtain precisely 2-litre of water in to the 4-litre jugs?

Solution:

The state space for this problem can be defined as

$$\{(i, j)\ i = 0, 1, 2, 3, 4\ j = 0, 1, 2, 3\}$$

'i' shows the number of litres of water in the 4-litre jug and 'j' shows the number of litres of water in the 3-litre jug. The preliminary state is (0, 0) that is no water on every jug. The objective state is to obtain (2, n) for any value of 'n'.

Notes

To resolve this we have to build some suppositions not stated in the problem. They are:

1. We can fill a jug from the pump.
2. We can pour water out of a jug to the ground.
3. We can pour water from one jug to another.
4. There is no measuring device available.

The different operators (Production Rules) that are obtainable to solve this problem may be declared as given in the following Table.

Table 3.1

Rule No.	Production Rule	Action
1.	(i, j) — (4, j) if i < 4.	Fill the 4-liter jug, if 4-liter jug is not full.
2.	(i, j) — (i, 3) if j < 3.	Fill the 3-liter jug, if 3-liter jug is not full.
3.	(i, j) — (I – s, j) if i > 0.	Pour some water out of the ground, if 4-liter jug is not empty.
4.	(i, j) — (i, j – s) if j > 0.	Pour some water out the 3-liter jug, if 3-liter jug is not empty.
5.	(i, j) — (o, j) if j > 0.	Empty the 4-liter jug on the ground, if 4-liter jug is not empty.
6.	(i, j) — (i, o) if j > 0.	Empty the 3-liter jug on the ground, if 3-liter jug is not empty.
7.	(i, j) — (4, j – (4 – I)) if (i + j) < = 4 & j < 0.	Pour water from the 30-liter jug into the 4-liter jug until the 4-liter jug is full, if the combined content is > = 4 and 3-liter jug is not empty.
8.	(i, j) — (i,(3 – j), 3) if (i + j) > = 3 & i > o.	Pour water from the 4-liter into the 3-liter jug until the 3-liter jug is full, if the combined content is > = 3 and 4-liter jug is not empty.
9.	(i, j) — (i + j, 0) if (i + j) > = 4 and i > 0.	Pour all the water from the 3-liter jug into the 4-liter jug if the jug, combined content is < = 4 and 3-liter jug is not empty.
10.	(i, j) — (0, i + j) if (i + j) < = 3 and i > 0.	Pour all the water from the 4-liter jug into the 3-liter jug, if the combined content is < = 3 and 4-liter jug is not empty.

For the water jug problem, there are several sequence of operators that will solve the problem, let us see of them.

Solution 1:

Table 3.2

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
4	0	1
1	3	8
1	0	6
0	1	10
4	1	1
2	3	8

Notes

Solution 2:

Table 3.3

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5
2	0	9

Solution 3:

Table 3.4

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
4	0	1
1	3	8
0	3	5
3	0	9
3	3	2
4	2	7
0	2	5
2	0	9

Figures gives comparative study of the above 3 different solutions.

In the comparison of 3 solutions, we see that, when there is no limit for water prevails then solution 1 is the most efficient. When water is limited then solution 2 is the best suited. In no way, solution 3 is good, Because it requires 8 steps to solution and wastes 5 liters of water.

Self Assessment

Fill in the blanks:

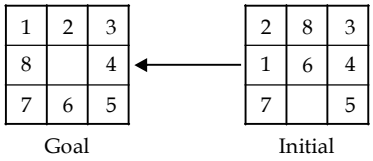
1. In ..... problem, neither jug contains any gauging markers on it.
2. There is a ..... that can be accessed to fill the jugs with water.

3.2 8 Puzzle Problem

The 8 puzzle comprises eight numbered, changeable tiles set in a 3 × 3 frame. One cell of the frame is at all times empty therefore making it probable to move a nearby numbered tile into the unfilled cell. Such a puzzle is demonstrated in following diagram.

Notes

Figure 3.1



The program is to modify the initial configuration into the objective configuration. A solution to the problem is a suitable sequence of moves, like "move tiles 5 to the right, move tile 7 to the left, move tile 6 to the down, etc".

Caution To solve a problem by means of a production system, we must state the global database the rules, and the control strategy.

For the 8 puzzle problem that communicate to these three components. These rudiments are the problem states, moves and goal. Here each tile configuration is a state. The set of all configuration in the space of problem declares or the problem space, there are only 3,62,880 different configurations of the 8 tiles and blank space. Once the problem states have been theoretically identified, we must build a computer representation, or description of them . This description is then accessed as the database of a production system. For the 8-puzzle, a straight forward description is a 3 × 3 array of matrix of numbers. The preliminary global database is this explanation of the initial problem state.

Did u know? Practically any type of data structure can be accessed to describe states.

A move converts one problem state into another state. The 8-puzzle is suitably interpreted as having the following for moves. Move bare space (blank) to the left, move blank up, move blank to the right and move blank down,. These moves are represented by production rules that function on the state descriptions in the suitable way.

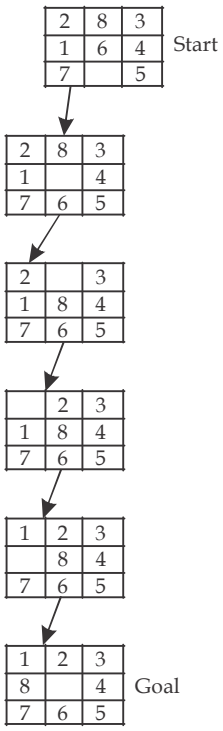
The rules each have prerequisites that must be contented by a state description in order for them to be valid to that state description. Therefore the precondition for the rule connected with "move blank up" is derived from the necessity that the blank space must not already be in the top row.

The problem goal condition generates the basis for the termination condition of the production system. The control strategy repeatedly use rules to affirm descriptions until a explanation of a goal state is produced. It also keep track of rules that have been applied so that it can compose them into sequence displaying the problem solution. A solution to the 8-puzzle problem is specified in the Figure 3.2.

Example: Depth-First Search traversal for 8-puzzle problem is displayed in Figure 3.2.

Notes

Figure 3.2



Self Assessment

Fill in the blanks:

3. The ..... comprises eight numbered, changeable tiles set in a 3 × 3 frame.
4. The program is to modify the initial configuration into the ..... configuration.
5. To solve a problem by means of a ..... system, we must state the global database the rules, and the control strategy.
6. The problem goal condition generates the basis for the ..... condition of the production system.

3.3 Frame Problem

In the restricted planet of a robot, surroundings are not static. Many changeable forces or actions can cause alterations or modifications to it. The problem of forcing a robot to acclimatize to these changes is the foundation of the frame problem in artificial intelligence. Information in the knowledge base and the robot's conclusions unite to form the input for what the robot's succeeding action should be. A good assortment from its details can be made by disposing or ignoring inappropriate facts and ridding of results that could have negative side effects.

A robot must bring in facts that are pertinent to a particular moment. Specifically, a robot will inspect its current situation, and then observe the facts that will be beneficial to selecting its subsequent action. The robot should also investigate for any changeable facts. It then inspects these facts to verify if any of them have been altered during a preceding examination.

Notes

There are two fundamental types of change:

- ***Relevant Change***: examine the changes made by an action
- ***Irrelevant Change***: do not examine facts that are not related to the task at hand

Facts may be examined utilizing two levels:

- ***Semantic Level***: This level interprets what type of information is being inspected. Solutions should become understandable by the suppositions of how an object should behave. There are believers in a wholly semantic approach who consider that accurate information can be reached by means of meaning. However, this hypothesis has yet to be confirmed.
- ***Syntactic Level***: This level just decides in which format the information should be examined. That is, it generates solutions depending on the structure and patterns of facts.

When examining the facts, numerous problems can happen:

- At times an implication can be missed.
- Considering all facts and all their succeeding side effects is time-consuming.
- Some facts are needlessly inspected when they are unneeded.

Task Make distinction between semantic level and syntactic level.

3.3.1 Problems Related to the Frame Problem

The Qualification Problem

The Qualification Problem was initiated by John McCarthy. It recommends that one is never totally positive if a particular rule will work. It also proposes that the robot does not essentially know which rules to disregard in a specified situation. Modifications in the environment can "confuse" the robot as certain rules will turn out to be obsolete and new rules will be essential before they exist.

The Representational Problem

The Representational Problem is the complexity of developing truths concerning the current environment. For instance, how can one program the thoughts of up and down? These are relative to each other, and can not be just described by direction. To incompletely correct this problem, successor-state axioms are accessed. These axioms demonstrate all the true and false potentials of a rule.

The Inferential Problem

Difficulty with inspecting the methods by which the world is reviewed is the Inferential Problem. There are two types of purposes. The General Purpose is to check the complete world of things that are changeable. The particular Purpose is to only examine actions that can amend over a little area of environments.

The Ramification Problem

This problem illustrates how an action can cause deviations inside its environment.

Notes

Example: A robotic arm has been provided the task of picking up a brick and positioning it on its side in a dissimilar location. If the brick has been knocked over, what can the robot perform to correct the problem? Will it still recognize which side should be facing up without the aptitude of human sight? Should these deviations be inspected individually every time an action has taken place?

The Predictive Problem

The Predictive Problem concerns with the advantages of predictions. Specifically, it is uncertain if a specified prediction will cause a positive change in the surroundings.

Did u know? If the variation will not be positive, "either the laws or explanation of the provided condition must be imperfect."

Task Make distinction between the Representational Problem and the Inferential Problem.

Self Assessment

Fill in the blanks:

7. The problem of forcing a robot to acclimatize to these changes is the foundation of the ..... problem in artificial intelligence.
8. ..... level interprets what type of information is being inspected.
9. ..... level just decides in which format the information should be examined.
10. The ..... problem recommends that one is never totally positive if a particular rule will work.
11. The General Purpose of ..... problem is to check the complete world of things that are changeable.
12. The ..... problem illustrates how an action can cause deviations inside its environment.

3.4 Epistemological Problems

The epistemological portion of AI studies what types of facts regarding the world are accessible to a viewer with specified opportunities to scrutinize, how these facts can be symbolized in the memory of a computer, and what rules allow legitimate conclusions to be drawn from these facts. It leaves away the heuristic problems of how to search spaces of probabilities and how to match patterns.

Considering epistemological problems independently has the following benefits:

1. The same problems of what information is obtainable to an spectator and what conclusions can be taken out from information arise in association with a variety of problem solving tasks.
2. A single solution of the epistemological problems can assist a wide variety of heuristic strategies to a problem.

Notes

3. AI is a very complicated scientific problem, so there are huge advantages in locating parts of the problem that can be separated out and separately attacked.

4. It is quite hard to formalize the facts of general knowledge. Current programs that influence facts in some of the domains are restricted to special cases and don't concern the difficulties that must be conquer to attain very intelligent behavior.

We will converse what facts a person or robot must consider in order to attain a goal by some approach of action. We will ignore the question of how these facts are displayed, e.g., whether they are demonstrated by program. We begin with great generality, so there are many problems. We get successively simpler problems by presuming that the difficulties we have acknowledged don't take place until we obtain to a class of problems we believe we can solve.

1. We start by enquiring whether solving the problem needs the cooperation of other people or overcoming their opposition. If either is true, there are two subcases. In the first subcase, the other people's wishes and goals must be considered, and the actions they will take in specified conditions predicted on the supposition that they will attempt to attain their goals, which may have to be exposed. The problem is even more hard if bargaining is concerned, because then the problems and indeterminacies of game theory are pertinent. Even if bargaining is not concerned, the robot still must "put himself in the position of the other people with whom he communicates".

   Caution Facts such as a person wanting a thing or a person hating another must be illustrated.

   The second subcase builds the assumption that the other people can be considered as machines with recognized input-output behavior. This is frequently a good assumption, e.g., one considers that a clerk in a store will sell the goods in exchange for their price and that a professor will allocate a grade in accordance with the quality of the work completed. Neither the goals of the clerk or the professor need be considered; either might well regard an effort to access them to optimize the communication as an incursion of privacy.

   In such conditions, man generally prefers to be treated as a machine. Let us now presume that either other people are not concerned in the problem or that the information obtainable regarding their actions takes the form of input-output relations and does not entail understanding their goals.

2. The second question is whether the strategy includes the acquisition of knowledge. Even if we can regard other people as machines, we still may have to reason regarding what they know. Therefore an airline clerk knows what airplanes fly from here to there and when, even though he will tell you when asked without your having to motivate him. One must also take into account information in books and in tables. The latter information is illustrated by other information.

3. The second subcase of knowledge is according to whether the information obtained can be simply plugged into a program or whether it enters in a more complex manner. Therefore, if the robot must telephone someone, its program can just dial the number received, but it might have to ask a question, *"How can I get in touch with Mike?"* and reason concerning how to access the resulting information in combination with other information. The common distinction may be according to whether new sentences are produced or whether values are just allocated to variables.

Example: An example valued considering is that a sophisticated air traveler rarely enquires how he will obtain from the arriving flight to the departing flight at an airport where he must

alter planes. He is certain that the information will be obtainable in a form he can appreciate at the time he will require it.

Notes If the strategy is personified in a program that branches on an environmental state or reads a numerical parameter from the environment, we can observe it as acquiring knowledge, but this is perceptibly an easier case than those we have conversed.

4. A problem is more hard if it includes concurrent events and actions. To me this appears to be the most hard unsolved epistemological problem for AI—how to articulate rules that provide the effects of actions and events when they take place concurrently. We may compare this with the sequential case regarded in (McCarthy and Hayes 1969). In the sequential case we can write $S' = result\ (e, s)$ (1) where s' is the circumstance that results when event e appears in situation s. The effects of e can be illustrated by sentences relating s', e and s. One can effort a similar formalism providing a *partial situation* that results from an event in another incomplete situation, but it is hard to view how to apply this to cases in which other events may influence with the incidence.

   When events are synchronized, it is generally essential to consider time as continuous. We have events such as *raining until the reservoir overflows* and questions like *Where was his train when we wanted to call him?*

   Computer science has lately begun to formalize parallel procedures so that it is at times possible to confirm that a system of parallel processes will fulfill its specifications. Though, the knowledge obtainable to a robot of the other processes going on in the world will not often take the form of a Petri net or any of the other formalisms accessed in engineering or computer science.

   Actually, anyone who desires to prove correct an airline reservation system or an air traffic control system must access information regarding the behavior of the external world that is less particular than a program. Nonetheless, the formalisms for expressing details regarding parallel and indeterminate programs offer a start for axiomatizing concurrent action.

5. A robot must be able to state knowledge regarding space, and the locations, shapes and layouts of objects in space. Present programs considers only very special cases. Typically locations are discrete—block A may be on block B but the formalisms do not permit anything to be said regarding where\ on block B it is, and what shape space is left on block B for positioning other blocks or whether block A could be shifted to project out a bit in order to place another block. Some are more sophisticated, but the objects must have simple geometric shapes. A formalism competent of representing the geometric information people get from seeing and managing objects has not, to my knowledge, been approached.

   The complexity in expressing such facts is symbolized by the limitations of English in articulating human visual knowledge. We can portray usual geometric shapes exactly in English (fortified by mathematics), but the information we access for identifying another person's face cannot normally be transmitted in words. We can respond to more questions in the occurrence of a scene than we can from memory.

6. The relation among three dimensional objects and their two dimensional retinal or camera images is typically untreated. Dissimilar to some philosophical positions, the three dimensional object is regarded by our minds as different from its appearances. People

Notes

blind from birth can still converse in the similar language as sighted people concerning three dimensional objects.

Notes We require a formalism that regards three dimensional objects as instances of patterns and their two dimensional appearances as projections of these patterns altered by lighting and occlusion.

7. Objects can be prepared by shaping materials and by merging other objects. They can also be taken distant, cut apart or destroyed in various manners. What people recognize about the relations among materials and objects remains to be illustrated.

8. Modal concepts such as *event e1 caused event e2* and *person e can do action a* are required. (McCarthy and Hayes 1969) considers ability as a function of a person's position in a fundamental system and not at all as a function of his interior structure. This still appears correct, but that action is only metaphysically adequate, since it doesn't give for expressing the information about capability that people really have.

9. Assume now that the problem can be formalized in terms of a single state that is altered by events. In appealing cases, the set of components of the state relies on the problem, but general knowledge is generally expressed in terms of the effect of an action on one or a few components of the state. Though, it cannot always be presumed that the other components are unaltered, especially since the state can be illustrated in a variety of co-ordinate systems and the meaning of changing a single co-ordinate relies on the co-ordinate system. The problem of showing information regarding what remains unchanged by an event was known as *the frame problem* in (McCarthy and Hayes 1969). Minsky consequently confused matters by means of the word "frame" for patterns into which situations may fit. (His hypothesis appears to have been that almost all conditions encountered in human problem solving fit into a small number of previously recognized patterns of situation and goal. I consider this as unlikely in difficult problems).

10. The *frame problem* may be a sub case of what we consider the *qualification problem*, and a good solution of the qualification problem may solve the frame problem also. In the *missionaries and cannibals* problem, a boat holding two people is declared to be obtainable. In the declaration of the problem, nothing is said regarding how boats are used to cross rivers, so obviously this information must come from general knowledge, and a computer program capable of solving the problem from an English description or from a conversion of this description into logic must have the necessary common knowledge. The simplest statement regarding the use of boats says something like, "*If a boat is at one point on the shore of a body of water, and a set of things enter the boat, and the boat is propelled to the another point on the shore, and the things exit the boat, then they will be at the second point on the shore*". Though, this statement is too rigid to be true, since anyone will admit that if the boat is a rowboat and has a leak or no oars, the action may not attain its desired result. One might try altering the familiar knowledge statement regarding boats, but this encounters difficulties when a critic asks for a qualification that the vertical exhaust stack of a diesel boat must not be struck square by a cow turd dropped by a passing hawk or some other event that no-one has earlier thought of. We require to be able to say that the boat can be accessed as a vehicle for crossing a body of water unless something averts it. Though, as we are not willing to set the limits of in advance possible situations that may avert the use of the boat, there is still a problem of confirming or at least conjecturing that nothing averts the use of the boat. The decline of the frame problem to the qualification problem has not been fully executed, however.

Notes

Self Assessment

Fill in the blanks:

13. The ..... portion of AI studies what types of facts regarding the world are accessible to a viewer with specified opportunities to scrutinize.
14. Epistemological problems leave away the ..... problems of how to search spaces of probabilities and how to match patterns.
15. The problem of showing information regarding what remains unchanged by an ..... was known as the frame problem.

3.5 Summary

- In water jug problems, neither jug contains any gauging markers on it. There is a pump that can be accessed to fill the jugs with water
- The 8 puzzle comprises eight numbered, changeable tiles set in a 3 × 3 frame.
- One cell of the frame is at all times empty therefore making it probable to move a nearby numbered tile into the unfilled cell.
- To solve 8 puzzle problem by means of a production system, we must state the global database the rules, and the control strategy.
- Many changeable forces or actions can cause alterations or modifications to it. The problem of forcing a robot to acclimatize to these changes is the foundation of the frame problem in artificial intelligence.
- Semantic Level interprets what type of information is being inspected.
- Syntactic Level just decides in which format the information should be examined.
- The epistemological portion of AI studies what types of facts regarding the world are accessible to a viewer with specified opportunities to scrutinize, how these facts can be symbolized in the memory of a computer, and what rules allow legitimate conclusions to be drawn from these facts.

3.6 Keywords

Semantic Level: Semantic Level interprets what type of information is being inspected.

Syntactic Level: Syntactic Level just decides in which format the information should be examined.

Water Jug Problem: In water jug problems, neither jug contains any gauging markers on it. There is a pump that can be accessed to fill the jugs with water.

3.7 Review Questions

1. What is 8 puzzle problem? Illustrate with example.
2. You have a two-gallon jug and a one-gallon jug; neither have any measuring marks on them at all. Initially both are empty. How do you get exactly one gallon into the two-gallon jug?
3. Suppose that you are given 'n' red and 'n' blue water jugs, all of different shapes and sizes. All red jugs hold different amounts of water, as do the blue ones. For every red jug, there

Notes

is a blue jug that holds the same amount of water and vice versa. How can you find the grouping of the jugs into pairs of red and blue jugs that hold the same amount of water, in the minimum number of comparisons?

4. Two friends who have an eight-quart jug of water wish to share it evenly. They also have two empty jars, one holding five quarts, the other three. How can they each measure exactly 4 quarts of water?
5. Show that the 8-puzzle states are divided into two disjoint sets, such that no state in one set can be transformed into a state in the other set by any number of moves. Devise a procedure that will tell you which class a given state is in, and explain why this is a good thing to have for generating random states.
6. What is frame problem? Illustrate the concept.
7. Illustrate the fundamental types of changes that occur in frame problem.
8. Discuss the various problems that can take place in case of frame problem.
9. What are Epistemological problems in artificial intelligence? Explain.
10. Explain the facts a person or robot must consider in order to attain a goal by some approach of action.

Answers: Self Assessment

1.	water jug	2.	pump
3.	8 puzzle	4.	objective
5.	production	6.	termination
7.	frame	8.	Semantic
9.	Syntactic	10.	Qualification
11.	inferential	12.	Ramification
13.	epistemological	14.	heuristic
15.	event		

3.8 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Notes

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.cogsci.ecs.soton.ac.uk/cgi/psyc/ptopic?topic=frame-problem

Notes

Unit 4: Heuristic Search Techniques

CONTENTS

Objectives

Introduction

4.1 Heuristic Search Techniques

4.2 Generate and Test

4.3 Hill Climbing

4.4 Best First Search

4.4.1 Best First Search Algorithm

4.4.2 The A* Algorithm

4.4.3 Greedy Best-first Search (GBFS)

4.5 Constraint Satisfaction

4.6 Means-ends Analysis

4.7 Summary

4.8 Keywords

4.9 Review Questions

4.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the various heuristic search techniques
- Discuss the generate and test, hill climbing, best first search
- Discuss the constraint satisfaction and mean-end analysis

Introduction

Most of the problems are too multifaceted to be solvable by straight techniques. They have to be solved only by appropriate heuristic search techniques. However the heuristic techniques can be illustrated separately, they are domain particular. They are known as " Weak Methods", as they are susceptible to combinatorial explosion. Nonetheless, they give the frame work into which domain specific knowledge can be positioned.

Each search process can be observed as a traversal of a directed graph, in which the nodes symbolize problem states and the arcs stand for relationships among states. The search process must locate a path through this graph, beginning at an initial state and ending in one or more final states. The issues discussed in the unit have to be considered before performing a search.

4.1 Heuristic Search Techniques

Heuristic techniques are known as weak methods, as they are susceptible to combinatorial explosion. Still these techniques persist to offer framework into which domain specific knowledge

Notes

can be positioned, either by hand or as a consequence of learning. The following are some common purpose control approaches (frequently known as weak methods).

- Generate-and-test
- Hill climbing
- Best First Search (A* search)
- Constraint satisfaction
- Means-ends analysis

A heuristic process, or heuristic, is defined as enclosing the following properties:

1. It will typically locate good, although not essential optimum solutions.
2. It is quicker and easier to execute than any recognized exact algorithm (one which guarantees an optimum solution).

Usually, heuristic search perk up the excellence of the path that are exported. By means of good heuristics we can expect to acquire good solutions to tough problems like the traveling salesman problem in less than exponential time. There are a number of good common purpose heuristics that are valuable in a broad variety of problems. It is also probable to create special purpose heuristics to resolve specific problems.

Self Assessment

Fill in the blanks:

1. Heuristic techniques are known as ..... methods, as they are susceptible to combinatorial explosion.
2. ..... techniques give the frame work into which domain specific knowledge can be positioned.

4.2 Generate and Test

This is the easiest search strategy. It comprises the following steps:

1. Producing a possible solution for some problems; this means generating a particular point in the problem space. For others it may be creating a path from a start state.
2. Test to observe if this is really a solution by comparing the chosen point at the end point of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit or else return to step 1.

The generate and Test algorithm is a depth first search practice since complete possible solutions are produced before test. This shows that executed states are likely to emerge frequently in a tree; it can be implemented on a search graph instead of a tree.

Self Assessment

Fill in the blanks:

3. The ..... algorithm is a depth first search practice since complete possible solutions are produced before test.

Notes

4. Producing a possible solution for some problems; this means generating a particular point in the ..... .

4.3 Hill Climbing

This is a selection of depth-first (generate and test) search. A feedback is utilized here to decide on the course of movement in the search space. In the depth-first search, the test function will just accept or reject a solution. But in hill climbing the test function is offered with a heuristic function which offers an estimate of how close a known state is to goal state. The hill climbing test process is as follows:

1. Generally he first proposed solution as performed in depth-first procedure. Observe if it is a solution. If so quit, else continue.
2. From this solution produce new set of solutions use, some application rules
3. For every element of this set
   (i) Apply test function. It is a solution quit.
   (ii) Moreover observe whether it is closer to the goal state than the solution already produced. If yes, keep in mind it else discard it.
4. Take the best element so far produced and use it as the next proposed solution.

   This step matches up to move through the problem space in the direction towards the objective state.
5. Go back to step 2.

At times this procedure may approach to a position, which is not a solution, but from which there is no move that enhances things. This will occur if we have reached one of the following three states:

(a) A "local maximum" which is a state improved than all its neighbors, but is not better than some other states farther away. Local maxim sometimes appears with in sight of a solution. In such cases they are known as "Foothills".

(b) A "plateau'' which is a flat area of the search space, in which adjacent states have the similar value. On a plateau, it is not probable to verify the best direction in which to move by making local comparisons.

(c) A "ridge" which is an area in the search that is superior than the surrounding areas, but can not be looked in a simple move.

To conquer these problems we can:

(a) Back track to some previous nodes and try a different direction. This is a fine manner of dealing with local maxim.

(b) Make a big jump in some course to a new area in the search. This can be produced by applying two more rules of the similar rule several times, before testing. This is a good approach is dealing with plate and ridges.

Example: A search algorithm that tries to locate a route that diminishes the number of connections utilize the heuristic that the longer the span of the flight, the greater the probability that it takes the traveler nearer to the target. Thus, the number of connections is diminished. This is an example of hill climbing in the language of Artificial Intelligence.

Notes

Notes Hill climbing turns out to be incompetent in large problem spaces, and when combinatorial explosion appears. But it is a functional when combined with other methods.

Task Illustrate the problems that occur in hill climbing technique.

Self Assessment

Fill in the blanks:

5. In ..... the test function is offered with a heuristic function which offers an estimate of how close a known state is to goal state.
6. A "..... '' which is a flat area of the search space, in which adjacent states have the similar value.
7. A "..... " which is an area in the search that is superior than the surrounding areas, but can not be looked in a simple move.

4.4 Best First Search

The general strategy of heuristic search is best-first search (BeFS)

- a node is chosen for expansion based on an evaluation function, f(n).
- expand the node with the lowest "evaluation" – the one that "appears" to be best as per the evaluation function
- algorithm uses a heuristic function, h(n):

  h(n) = estimated cost of cheapest path from node n to a goal node.

Caution Evaluation gauges the distance to the goal.

Best First Search is a amalgamation of depth first and breadth first searches.

Depth first is good since a solution can be located without calculating all nodes and breadth first is good since it does not get trapped in dead ends. The best first search permits us to switch among paths thus gaining the advantage of both approaches. At every step the most promising node is selected. If one of the nodes selected produces nodes that are less promising it is probable to select another at the similar level and in effect the search alters from depth to breadth. If on analysis these are no better then this beforehand unexpanded node and branch is not forgotten and the search technique reverts to the descendants of the first option and proceeds, backtracking as it were.

This process is very alike to steepest ascent, but in hill climbing once a move is selected and the others are rejected the others are not at all reconsidered even as in best first they are saved to allow revisits if an impasse appears on the evident best path. Also the best obtainable state is chosen in best first even its value is inferior than the value of the node just discovered while in hill climbing the progress stops if there are no healthier successor nodes. The best first search algorithm will engross an OR graph which averts the problem of node duplication and presumes that every node has a parent link to provide the best node from which it came and a link to all

Notes

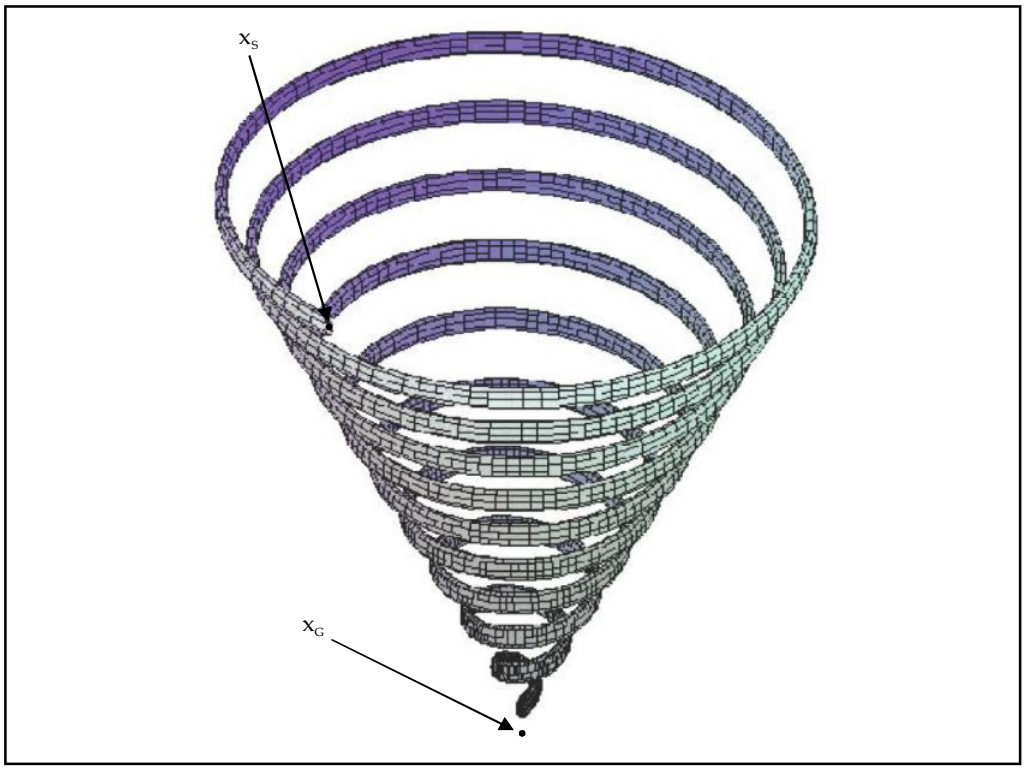
its successors. In this method if a better node is located this path can be propagated down to the successors. This method of using an OR graph needs 2 lists of nodes

OPEN is a precedence queue of nodes that have been evaluated by the heuristic function but which have not yet been extended into successors. The most capable nodes are at the front. CLOSED are nodes that have already been produced and these nodes must be amassed since a graph is being used in partiality to a tree.

Heuristics so as to discover the most promising nodes a heuristic function is required known as f′ where f′ is an rough calculation to find is made up of two parts g and h′ where g is the cost of going from the first state to the current node; g is considered merely in this context to be the number of arcs navigated each of which is considered as being of unit weight. h′ is an estimate of the initial cost of obtaining from the current node to the goal state. The function f′ is the estimated value or estimate of obtaining from the initial state to the objective state. Both g and h′ are positive valued variables. Best First The Best First algorithm is an easy form of the A* algorithm. From A* we note down that f′ = g+h′ where g is assess of the time taken to go from the initial node to the current node and h′ is an estimate of the time taken to solution from the current node. Therefore f′ is an estimate of how long it takes to go from the original node to the solution.

Did u know? As a support we take the time to go from one node to the subsequent one to be a constant at 1.

Example: Here is an example for best-first search in the figure given below. Visualize trying to arrive at a state that is shown below the spiral tube. If the initial state begins inside of the opening at the top of the tube, the search will move in the region of the spiral rather than leaving the tube and looking directly for the goal.

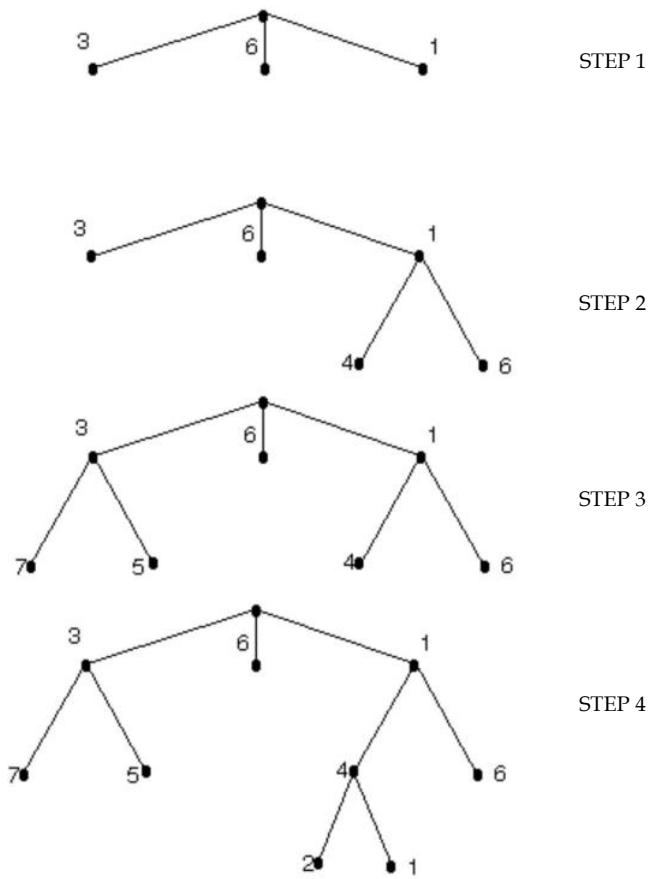


Notes

4.4.1 Best First Search Algorithm

1. Begin with OPEN holding the initial state
2. Choose the best node on OPEN
3. Produce its successors
4. For each successor Do
   (i) If it has not been produced before evaluate it add it to OPEN and record its parent
   (ii) If it has been produced before change the parent if this new path is better and in that case update the cost of obtaining to any successor nodes
5. If a goal is located or no more nodes left in OPEN, quit, else return to 2.

Figure 4.1



All figures indicate "cost" of move

4.4.2 The A* Algorithm

Best first search is a simplified A*.

1. Begin with OPEN holding the initial nodes.
2. Choose the BEST node on OPEN such that f = g + h′ is minimal.

Notes

3. If BEST is goal node quit and return the path from initial to BEST Otherwise
4. Eradicate BEST from OPEN and all of BEST's children, labelling each with its path from initial node.

A* Search

- Minimizes whole estimated solution cost
- Measures nodes by combining g(n) – the cost to reach node n -, and h(n) – the cost to get from node n to the goal, thus

  f(n) = g(n) + h(n) = estimated cost of the cheapest solution through n.

A* is most constructive if h(n) is an admissible heuristic – that is, h(n) never overrates the cost to reach the goal.

Disadvantages of A*

- Although usually better than the uninformed searches, the computation time of A* is too large
- As it keeps all generated nodes in memory, A* usually runs out of space
- Not practical for numerous large-scale problems.

Example: ***Eight puzzle Example:*** The heuristic of number of tiles out-of-position is definitely less than the definite number of moves to the goal state. So this heuristic (united with best-first search) is a permissible algorithm. So is the heuristic sum of the distances of out-of position tiles, since it too undervalues the definite number of moves needed to arrive at a goal state.

Task Discuss how *A* Search is performed.*

4.4.3 Greedy Best-first Search (GBFS)

- Develop node closest to the goal; choose path with lowest h(n)
- Evaluate nodes by using heuristic function – that is, f(n) = h(n).

Example: Make use of straight-line distance heuristic, hSLD.

In the example, the path A-S-F-B (found by GBFS) is longer than the path

A-S-R-P-B by 32km!!

In going from I to F, the heuristic proposes that N be expanded first, but it is a dead end!!

Worst-case time and space complexity is O(bm), where m is the maximum depth of the search space.

The option of the heuristic is important.

Notes

Notes

1. Like DFS, GBFS follows a single path to the objective, but "backtracks" when it hits a dead end.
2. Like DFS it is not most favorable and it is incomplete.

Self Assessment

Fill in the blanks:

8. ..... is a amalgamation of depth first and breadth first searches.
9. ..... is a precedence queue of nodes that have been evaluated by the heuristic function but which have not yet been extended into successors.
10. ..... are nodes that have already been produced and these nodes must be amassed since a graph is being used in partiality to a tree.
11. The Best First algorithm is an easy form of the ..... algorithm.

4.5 Constraint Satisfaction

Many troubles in AI can be regarded as problems of constraint satisfaction, where the goal state pleases a specified set of constraint.

Caution Constraint satisfaction problems can be solved by means of any of the search approaches.

The common form of the constraint satisfaction procedure is as follows:

Until a whole solution is located or until all paths have led to lead ends, do

1. Choose an unexpanded node of the search graph.
2. Apply the constraint inference rules to the chosen node to generate all possible new constraints.
3. If the set of constraints encloses a contradiction, then report that this path is a dead end.
4. If the set of constraints illustrates a total solution then report success.
5. If neither a constraint nor a complete solution has been located then apply the rules to produce new partial solutions. Insert these partial solutions into the search graph.

Example: Consider the crypt arithmetic problems.

```
  SEND
+ MORE
-----
MONEY
-----
```

Notes

Allocate decimal digit to all the letters in such a manner that the answer to the problem is correct to the same letter appears more than once, it must be allocated the same digit each time. No two different letters may be allocated the same digit. Consider the below given crypt arithmetic problem.

```
  SEND
+ MORE
-----------
MONEY
-----------
```

Constraints:

1. No two digit can be allocated to same letter.
2. Only single digit number can be allocate to a letter.
3. No two letters can be allocated same digit.
4. Assumption can be prepared at different levels such that they do not oppose each other.
5. The problem can be divided into secured constraints. A constraint satisfaction strategy may be used.
6. Any of search techniques may be utilized.
7. Backtracking may be performed as applicable us applied search techniques.
8. Rule of arithmetic may be followed.

Initial state of problem.

D = ?

E = ?

Y = ?

N = ?

R = ?

O = ?

S = ?

M = ?

C1 = ?

C2 = ?

C1, C2, C3 stands for the carry variables respectively.

Goal State: The digits to the letters must be allocated in such a manner so that the sum is satisfied.

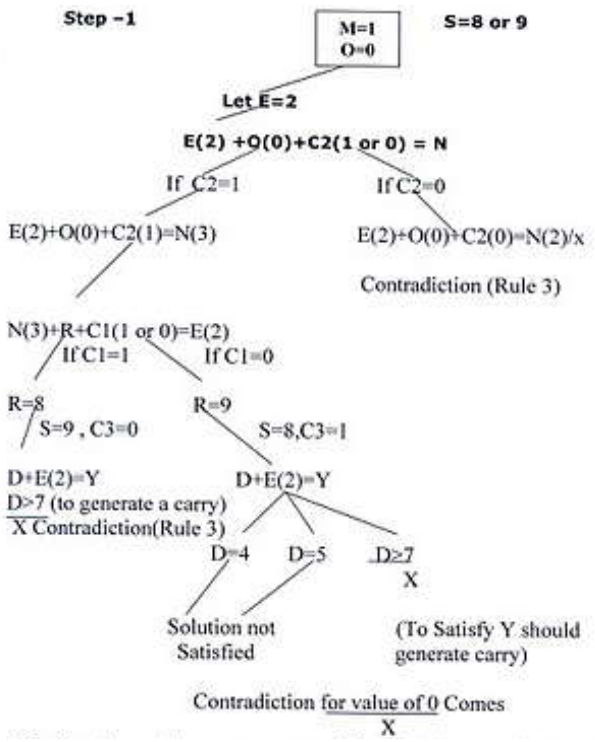
Solution Process:

We are following the depth-first technique to solve the problem.

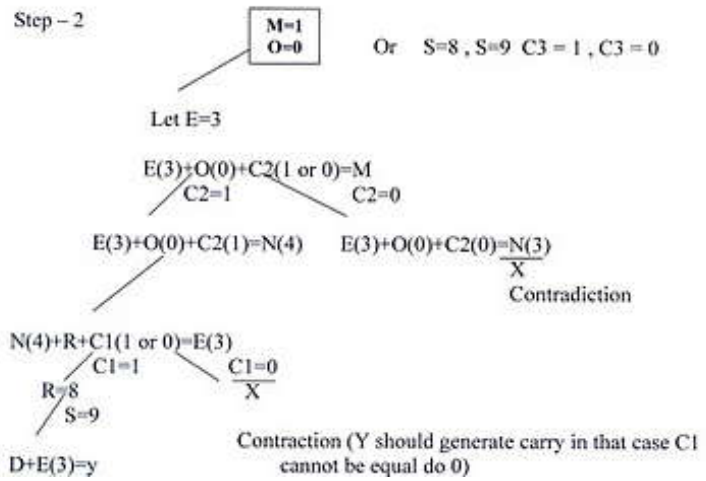
1. Initial guess m=1 since the sum of two single digits can generate at most a carry '1'.
2. When n=1 o=0 or 1 since the largest single digit number added to m=1 can produce the sum of either 0 or 1 depend on the carry obtained from the carry sum. By this we say that o=0 because m is already 1 hence we cannot assign same digit another letter (rule no.)
3. We have m=1 and o=0 to get o=0 we have s=8 or 9, again based on the carry obtained from the earlier sum.

Notes

The similar process can be repeated further. The problem has to be composed into different constraints. And each constraints is to be satisfied by guessing the likely digits that the letters can be assumed that the initial supposition has been already made . Rest of the process is being revealed in the form of a tree, by means of depth-first search for the clear understandability of the solution process.



After Step 1 we derive are more conclusion that Y contradiction should generate a Carry. That is D + 2 > 9

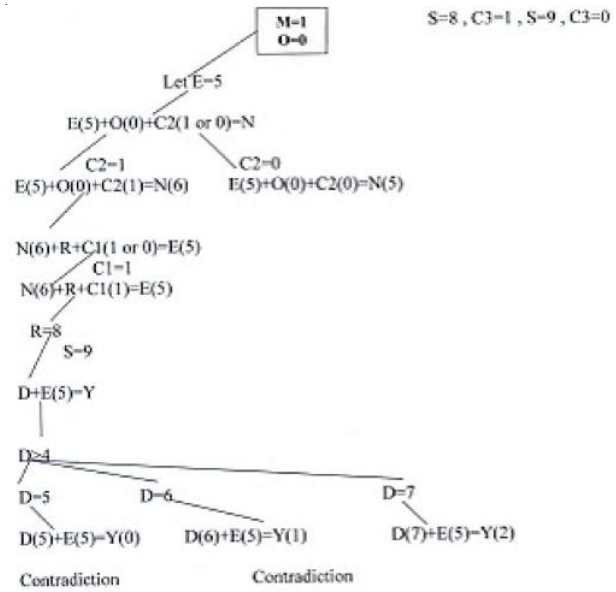


D > 6 (Controduction)

After Step 2, we found that C1 cannot be Zero, Since Y has to a carry to satisfy goal state. From this step onwards, no need to branch for C1 = 0.

Notes

Step – 3



We have assigned digit to every letter in accordance with the constraints and production rules.

Now by backtracking, we find the different digits assigned to different letters and hence reach the solution state.

Solution State:

Y = 2

D = 7

S = 9

R = 8

N = 6

E = 5

O = 0

M = 1

C1 = 1

C2 = 0

C3 = 0

C3(0)	C2(1)	C1(1)		
S(9)	E(5)	N(6)	D(7)	
+ M(1)	O(0)	R(8)	E(5)	
M(1)	O(0)	N(6)	E(5)	Y(2)

Notes

Example: Given Problem:

CROSS

+ROADS

– – – – – –

DANGER

– – – – – –

Constraints & Production Rules:

Same rules as used for the previous problem.

Initial State of Problem:

C = ?

R = ?

D = ?

A = ?

O = ?

N = ?

G = ?

S = ?

E = ?

C1 = ?

C2 = ?

C3 = ?

C4 = ?

Goal State:

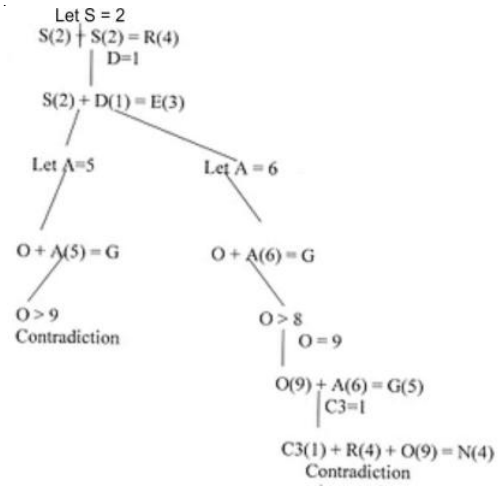
The Digits to the Letters should be allocated in such a way. So that, the sum is satisfied.

Solution Process:

Again, I am following depth-first search technique to solve the problem.

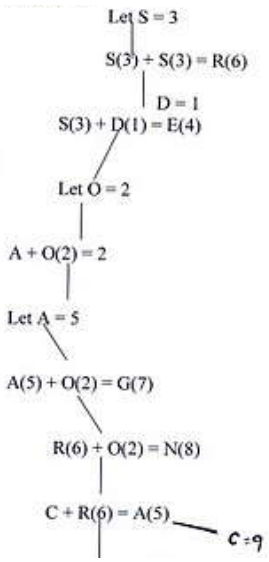
1. Initial Guess, D=1. Since the sum of two single digits can, at most, generate a carry '1'.
2. Now, we begin guessing the digits and try to solve the problem. The supposing and the consequent effect of it is exposed in the form of a tree below.

Notes



After the Step 1, w see that the initial guess is not correct. Because all the guesses following that have been checked, which has lead to the contradiction. Thus we move to second step with the change in the initial guess.

Step – 2



At Step (2) we have allocated a single digit to every letter in accordance with the constraints and production rules.

Now by backtracking, we discover the different digits allocated to dissimilar letters and therefore reach the solution state.

Solution State:

C = 9

N = 8

G = 7

A = 5

O = 2

Notes

E = 4

D = 1

R = 6

S = 3

C1 = 0

C2 = 0

C3 = 0

C4 = 0

C4(0) C3 C2(0) C1(0)

C9 R(6) O(2) S(3) S(3)

+ R6 O(2) A(5) D(1) S(3)

D(1) A(5) N(8) G(7) E(4) R(6)

Self Assessment

Fill in the blanks:

12. Many troubles in AI can be regarded as problems of ..... where the goal state pleases a specified set of constraint.
13. Constraint satisfaction problems can be solved by means of any of the ..... approaches.

4.6 Means-ends Analysis

Many of the search strategies either reason forward of backward however, frequently a mixture o the two directions is suitable. Such assorted strategy would make it probable to solve the major parts of problem first and solve the lesser problems occur when combining them together. Such a technique is known as "Means-ends Analysis".

The problem space of means-ends analysis has an early state and one or more objective state, a set of operate with a set of preconditions their application and difference functions that calculates the difference among two state a(i) and s(j).

Did u know? The means-ends analysis process centers about locating the difference among current state and goal state.

A problem is solved by means of means-ends analysis by:

1. Calculating the current state s1 to a target state s2 and computing their difference D12.
2. Satisfy the preconditions for some suggested operator op is selected, then to reduce the difference D12.
3. The operator OP is applied if probable. If not the present state is solved a goal is formed and means- ends analysis is applied recursively to decrease the sub goal.
4. If the sub goal is solved state is reinstated and work resumed on the original problem. (The first AI program to use means-ends analysis was the GPS General problem solver)

Means-ends analysis is functional for many human planning activities.

Notes

Example: Take the example of planning for an office worker. Let we have a different table of three rules:

1. If in out current state we are starving, and in our objective state we are not starving, then either the "visit hotel" or "visit Canteen " operator is suggested.
2. If our current state we do not have money, and if in your goal state we have money, then the "Visit our bank" operator or the "Visit secretary" operator is suggested.
3. If our current state we do not identify where something is needed in our target state we do recognize, then either the "visit office enquiry", "visit secretary" or "visit co worker " operator is suggested.

Self Assessment

Fill in the blanks:

14. The ..... process centers about locating the difference among current state and goal state.
15. Means- ends analysis is functional for many ..... activities.

4.7 Summary

- Heuristic techniques are known as weak methods, as they are susceptible to combinatorial explosion.
- The generate and Test algorithm is a depth first search practice since complete possible solutions are produced before test.
- In hill climbing technique, a feedback is utilized here to decide on the course of movement in the search space. In the depth-first search, the test function will just accept or reject a solution.
- Best First Search is a amalgamation of depth first and breadth first searches.
- Greedy best-first Search (GBFS) develop node closest to the goal; choose path with lowest h(n) and evaluate nodes by using heuristic function – that is, f(n) = h(n).
- Many troubles in AI can be regarded as problems of constraint satisfaction, where the goal state pleases a specified set of constraint.
- The means-ends analysis process centers about locating the difference among current state and goal state.
- Means-ends analysis is functional for many human planning activities.

4.8 Keywords

Best First Search: Best First Search is a amalgamation of depth first and breadth first searches.

Constraint Satisfaction: Many troubles in AI can be regarded as problems of constraint satisfaction, where the goal state pleases a specified set of constraint.

Generate and Test Algorithm: It is a depth first search practice since complete possible solutions are produced before test.

Notes

Greedy Best-first Search (GBFS): Develop node closest to the goal; choose path with lowest h(n) and evaluate nodes by using heuristic function – that is, f(n) = h(n).

Heuristic Techniques: Heuristic techniques are known as weak methods, as they are susceptible to combinatorial explosion.

Hill Climbing Technique: In this, a feedback is utilized here to decide on the course of movement in the search space. In the depth-first search, the test function will just accept or reject a solution.

Means-ends Analysis: This process centers about locating the difference among current state and goal state.

4.9 Review Questions

1. What are Heuristic techniques? Illustrate why the Heuristic techniques are considered as weak methods.
2. Enlighten various properties of heuristic process.
3. Explain the steps used in Generate and Test technique.
4. Depict the working of Hill Climbing technique.
5. Best First Search is a amalgamation of depth first and breadth first searches. Comment.
6. Make distinction between A* Algorithm and Greedy Best-first Search (GBFS) algorithm.
7. Illustrate the process of solving constraint satisfaction problems.
8. Explain the functioning of Means-ends Analysis technique with examples.
9. Discuss the steps used in solving Means-ends Analysis problem.
10. Explain the advantages and disadvantage of A* algorithm.

Answers: Self Assessment

1.	weak	2.	Heuristic
3.	generate and Test	4.	problem space
5.	hill climbing	6.	plateau
7.	ridge	8.	Best First Search
9.	OPEN	10.	CLOSED
11.	A*	12.	constraint satisfaction
13.	search	14.	means-ends analysis
15.	human planning		

4.10 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Notes

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

krchowdhary.com/ai/heuristic-serch.pdf

Notes

Unit 5: Knowledge Representation

CONTENTS

Objectives
Introduction
5.1 General Concepts of Knowledge
5.1.1 Explicit Knowledge
5.1.2 Tactic Knowledge
5.1.3 Knowledge Representation
5.1.4 Trends in Knowledge Engineering
5.1.5 Knowledge Acquisition: State-of-the-Art
5.2 Approaches of Knowledge Representation
5.2.1 Frames
5.2.2 Rules
5.2.3 Semantics
5.3 Predicate Logic to Represent Knowledge
5.3.1 Isa and Instance Relationships
5.4 Resolution
5.4.1 Normal Forms
5.4.2 Resolution Rule of Inference
5.4.3 Applying Resolution
5.5 Unification Algorithm
5.5.1 Diversity of Commonsense Thinking
5.6 Summary
5.7 Keywords
5.8 Review Questions
5.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the general concepts of knowledge
- Discuss the approaches of knowledge representation
- Understand the use of predicate logic to represent knowledge
- Discuss the resolution
- Illustrate the unification algorithm

Notes

Introduction

In today's new age economy, knowledge and information are the most important factors in the long-term success of both an individual and an organization. Knowledge is information extracted, filtered or formatted in some way. Knowledge and knowledge management have emerged as a vital component for many organizations. In fact, knowledge may soon be the only source of competitive advantage for an organization. All too often one part of an organization repeats the work of another part simply because it is impossible to keep track of, and makes use of knowledge in other parts.

5.1 General Concepts of Knowledge

Knowledge management is a critical component of an organizations success. Knowledge assets are the knowledge that an organization owns or needs to own, to achieve its goals.

Every company's knowledge requirements are a unique combination of knowledge strategy, tools and technologies, processes and procedures. Knowledge management technologies capture this intangible element in an organization and make it universally available. This approach has come to be known as knowledge management: the practice of capturing and organizing information to make it more accessible and valuable to those who need it.

With the impact of globalization, the Internet, and the rapid evolution of technology, managing knowledge for competitive advantage has become more important than ever. Knowledge management is therefore an "essential ingredient of success" for all organizations.

Knowledge can be divided into two types, Tacit knowledge and Explicit knowledge. Tacit knowledge is implicit, whereas Explicit knowledge is rule-based knowledge that is used to match actions to situations by invoking appropriate rules. An organization promotes the learning of Tacit knowledge to increase the skills and creative capacities of its employees and takes advantage of Explicit knowledge to maximize efficiency.

Caution To manage knowledge it must first of all be captured or acquired in some useful form.

5.1.1 Explicit Knowledge

Knowledge that can be more easily attained and is often expressed or documented in a formal, systematic manner - frequently in words and numbers.

Example: Include Management Directives, Executive Orders, policy manuals, and reference guides.

- Explicit knowledge is used in the design of routines, standard operation procedures, and the structure of data records. These forms of knowledge can be found in any organization.
- It allows an organization to enjoy a certain level of operational efficiency and control.
- Explicit knowledge promotes equable, consistent organizational responses.

Notes

5.1.2 Tactic Knowledge

Knowledge that can also be attained, but is not as easily transferred. Tacit knowledge can be attained through dialogue, job shadowing, storytelling, and sharing of best practices and lessons learned. It usually is rooted in an individual's experiences, intuition, insight, judgment, and knowledge of organizational values. Individuals with tacit knowledge are usually considered to be experts within their organizations and frequently sought out for guidance and input.

- Tacit knowledge includes hands-on skills, best practices, special know-how, and intuitions. Personal knowledge that is difficult to articulate.
- Tacit knowledge in an organization ensures task effectiveness. It also provides for a kind of creative vitality – intuition and spontaneous insight can often tackle tough problems that would otherwise be difficult to solve.
- Traditionally the transfer of Tacit knowledge is through shared experience, through apprenticeship and job training.
- Tacit knowledge is cultivated in an organizational culture that motivates through shared vision and common purpose.

An organization must adopt a holistic approach to knowledge management that successfully combines Tacit and Explicit knowledge at all levels of the organization.

Notes Personal knowledge is leveraged with Explicit knowledge for the design and development of innovative products, services and processes.

5.1.3 Knowledge Representation

This part depicts the way to show knowledge, for easy understanding we will take the help of knowledge acquisition.

Knowledge Acquisition in Context: Knowledge-based System Technologies

Knowledge acquisition can be considered as one further technology contributing to the development of knowledge-based Systems (KBS).

Knowledge Support System

At the top of the hierarchy are experimental systems integrating knowledge acquisition and performance tools in systems designed to support knowledge base updating and extension as part of ongoing applications.

Knowledge Acquisition Tools

At the next level are the tools for automating knowledge engineering for KBS, through automatic interview procedures, modeling expert behavior, and analysis of knowledge in textual form.

Notes

Knowledge-based System Support Environment

At the third level of the hierarchy is the equivalent of the Application Programming Support Environment (APSE) in conventional systems, with facilities for editing, displaying, debugging, and validating the knowledge base.

Knowledge-based System Shell

At the fourth level of the hierarchy is the knowledge-based system shell as a run-time environment that elicits problem-specific information from the user, provides advice based on its knowledge base, and explains that advice in as much detail as required.

Shell Development Language

At the fifth level of the hierarchy is the language in which the knowledge-based system shell is written, generally a special-purpose environment for coping with knowledge representation and inference.

Implementation Language

At the sixth level of the hierarchy is the implementation language which actually interfaces to the computer. This tended to be Lisp in the early days of KBS, but as speed and space efficiency have become significant and knowledge representation has become better understood, other languages that support dynamic data structures such as C and Pascal have become widely used.

Operating System

At the seventh level of the hierarchy is the operating system within which the implementation runs. This needs to provide good interfaces to other programs, large databases and communications.

Machine Architecture

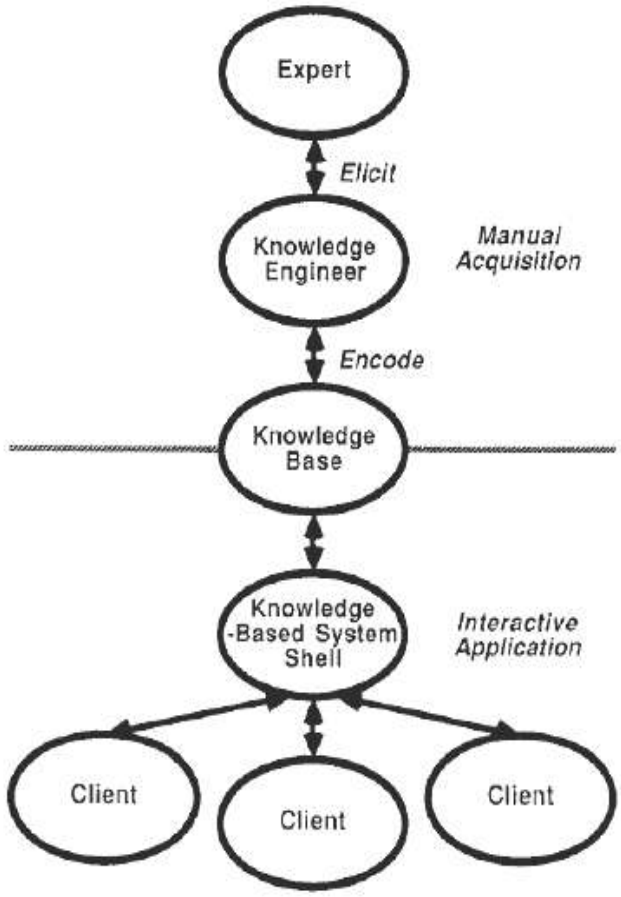
At the lowest level of the hierarchy is the machine on which the KBS runs. In theory, system developers should not need to know about the lower levels of the hierarchy machine architectures, operating systems, and implementation languages are remote from knowledge processing. In practice, these lower levels are the foundations on which systems are built, and any defects in them can undermine the functionality of the upper levels.

5.1.4 Trends in Knowledge Engineering

The basic model for knowledge engineering has been that the knowledge engineer mediates between the expert and knowledge base, eliciting knowledge from the expert, modeling and encoding it for the knowledge base, and refining it in collaboration with the expert to achieve acceptable performance. Figure 5.1 shows this basic model with manual acquisition of knowledge from an expert followed by interactive application of the knowledge with multiple clients through an expert system shell.

Notes

Figure 5.1: Basic Model of Manual Knowledge Acquisition



5.1.5 Knowledge Acquisition: State-of-the-Art

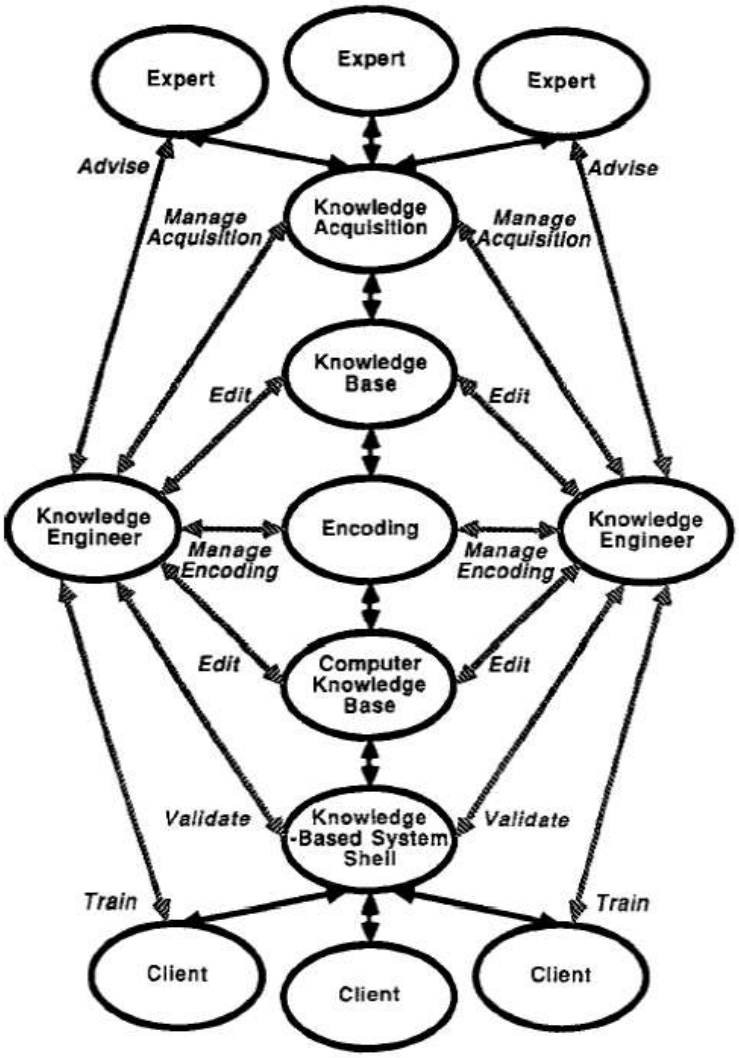
This basic model has been greatly extended by the introduction of interactive knowledge acquisition and encoding tools allowing the expert to enter knowledge directly to the system without an intermediary. Such tools can greatly reduce the need for the knowledge engineer to act as an intermediary, but, in most applications, they leave a substantial role for the knowledge engineer.

As shown in Figure 5.2, knowledge engineers may:

- Advise the expert on the process of interactive knowledge elicitation.
- Manage the interactive knowledge acquisition tools, setting them up appropriately.
- Edit the partially encoded knowledge base in collaboration with the expert.
- Manage the knowledge encoding tools, setting them up appropriately.
- Edit the encoded knowledge base in collaboration with the expert.
- Validate the application of the knowledge base in collaboration with the expert.
- Train the clients in the effective use of the knowledge base in collaboration with the expert by developing operational and training procedures.

Notes

Figure 5.2: Knowledge Engineers' Roles in Interactive Computer-based Knowledge Acquisition



This use of interactive computer-based elicitation can be combined with manual elicitation and with the use of the interactive tools by the knowledge engineer rather than, or in addition to, the expert. The knowledge engineer can:

- Directly elicit knowledge from the expert.
- Use the interactive elicitation tools to enter knowledge into the knowledge base.

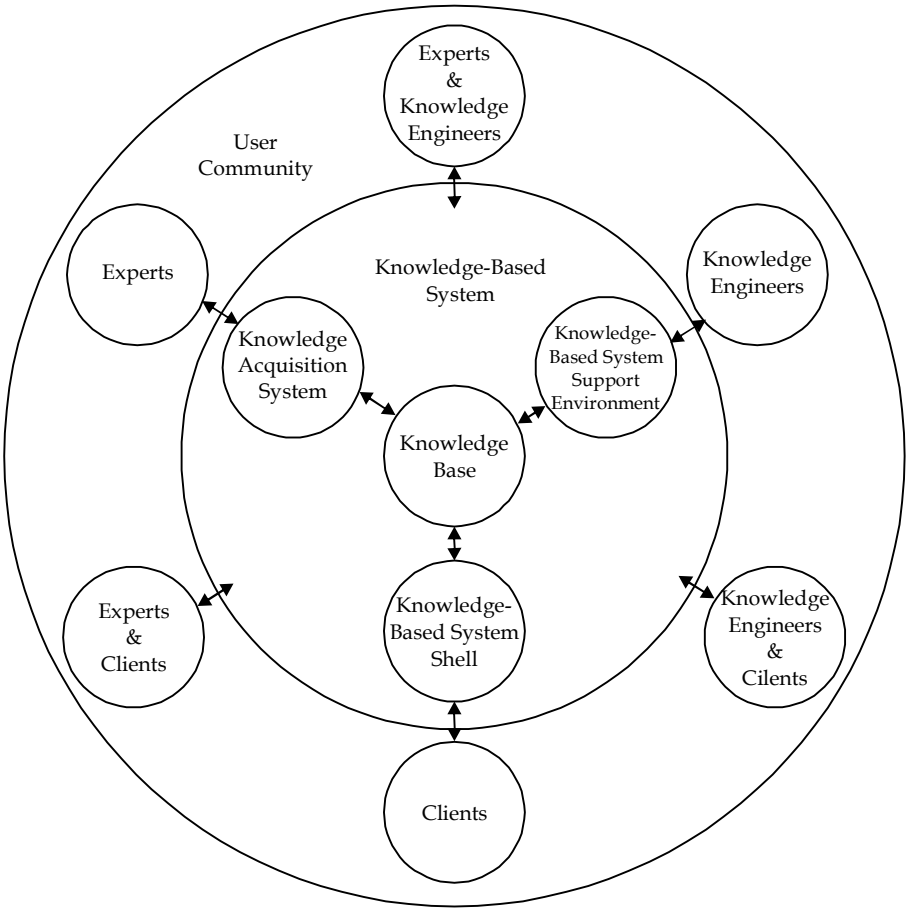
Figure 5.2 shows multiple knowledge engineers since the tasks above may require the effort of more than one person, and some specialization may be appropriate. Multiple experts are also shown since it is rare for one person to have all the knowledge required, and, even if this were so, comparative elicitation form multiple experts is itself a valuable knowledge elicitation technique.

Figure 5.2 also shows the complexity of the knowledge engineer's role and some of the support tools required. Figure 5.3 groups the support tools for editing, display, encoding, and validation of the knowledge bases into a Knowledge-Based System Support Environment and combines the various forms of knowledge bases together. It shows the overall structure of a

Notes

knowledge-based system as a central knowledge base interacting through the knowledge acquisition tools, the expert system support environment, and expert system shell, with a user community of experts, knowledge engineers, and clients.

Figure 5.3: Major Components of a Knowledge-based System



Task Discuss the concept of knowledge acquisition.

Self Assessment

Fill in the blanks:

1. ..... knowledge is rule-based knowledge that is used to match actions to situations by invoking appropriate rules.
2. ..... knowledge can be attained through dialogue, job shadowing, storytelling, and sharing of best practices and lessons learned.
3. Knowledge acquisition can be considered as one of the technology contributing to the development of ..... .

Notes

5.2 Approaches of Knowledge Representation

Knowledge representation is an area in artificial intelligence that is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse" – that which can be talked about, along with functions that may or may not be within the domain of discourse that allow inference (formalized reasoning) about the objects within the domain of discourse to occur. Generally speaking, some kind of logic is used both to supply a formal semantics of how reasoning functions apply to symbols in the domain of discourse, as well as to supply (depending on the particulars of the logic), operators such as quantifiers, modal operators, etc. that, along with an interpretation theory, give meaning to the sentences in the logic.

When we design a knowledge representation (and a knowledge representation system to interpret sentences in the logic in order to derive inferences from them) we have to make trades across a number of design spaces, described in the following sections. The single most important decision to be made, however is the expressivity of the KR. The more expressive, the easier (and more compact) it is to "say something". However, more expressive languages are harder to automatically derive inferences from.

Example: An example of a less expressive KR would be propositional logic. An example of a more expressive KR would be autoepistemic temporal modal logic.

Less expressive KRs may be both complete and consistent (formally less expressive than set theory). More expressive KRs may be neither complete nor consistent.

The key problem is to find a KR (and a supporting reasoning system) that can make the inferences your application needs in time, that is, within the resource constraints appropriate to the problem at hand. This tension between the kinds of inferences an application "needs" and what counts as "in time" along with the cost to generate the representation itself makes knowledge representation engineering interesting. There are representation techniques such as frames, rules and semantic networks which have originated from theories of human information processing.

Did u know? The fundamental goal of knowledge representation is to represent knowledge in a manner as to facilitate inferencing (i.e. drawing conclusions) from knowledge.

5.2.1 Frames

Frames were proposed by Marvin Minsky in his 1974 article "A Framework for Representing Knowledge." A frame is an artificial intelligence data structure used to divide knowledge into substructures by representing "stereotyped situations." Frames are connected together to form a complete idea.

Frame Structure

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.

Did u know? Different frames may share the same terminals.

Notes

A frame's terminals are already filled with default values, which is based on how the human mind works.

Example: When a person is told "a boy kicks a ball," most people will be able to visualize a particular ball (such as a familiar soccer ball) rather than imagining some abstract ball with no attributes.

5.2.2 Rules

If-then rules, which are arguably the most common form of knowledge representation in Artificial Intelligence, are ambiguous. They can be interpreted both as logic programs having the form if conditions then conclusions and as production rules having the form if conditions then do actions. The relationship between these different kinds of rules has received little attention in the AI literature; and, when it has, different authors have reached entirely different conclusions.

Some authors, such as Russell and Norvig in their textbook Introduction to Artificial Intelligence, view production rules as just logical implications used to reason forward, while Herbert Simon in the MIT Encyclopedia of Cognitive Science views the logic programming language Prolog as one of many production system languages. On the other hand, Thagard in his Introduction to Cognitive Science denies any relationship between logic and production rules at all.

5.2.3 Semantics

A semantic network is a network which represents semantic relations between the concepts. This is often used as a form of knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges.

Example: An example of a semantic network is WordNet, a lexical database of English. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets.

Some of the most common semantic relations defined are meronymy (A is part of B, i.e. B has A as a part of itself), holonymy (B is part of A, i.e. A has B as a part of itself), hyponymy (or troponymy) (A is subordinate of B; A is kind of B), hypernymy (A is superordinate of B), synonymy (A denotes the same as B) and antonymy (A denotes the opposite of B). WordNet properties have been studied from a network theory perspective and compared to other semantic networks created from Roget's Thesaurus and word association tasks respectively yielding the three of them a small world structure.

It is also possible to represent logical descriptions using semantic networks such as the existential Graphs of Charles Sanders Peirce or the related Conceptual Graphs of John F. Sowa. These have expressive power equal to or exceeding standard first-order predicate logic. Unlike WordNet or other lexical or browsing networks, semantic networks using these can be used for reliable automated logical deduction. Some automated reasoners exploit the graph-theoretic features of the networks during processing.

Self Assessment

Fill in the blanks:

4. ..... is an area in artificial intelligence that is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse"

Notes

5. A ..... is an artificial intelligence data structure used to divide knowledge into substructures by representing "stereotyped situations."
6. The frame contains ..... on how to use the frame, what to expect next, and what to do when these expectations are not met.
7. ..... rules, which are arguably the most common form of knowledge representation in Artificial Intelligence, are ambiguous.
8. A ..... network is a network which represents semantic relations between the concepts. This is often used as a form of knowledge representation.

5.3 Predicate Logic to Represent Knowledge

Here we will emphasize main ethics enclosed in knowledge representation. Particularly predicate logic will be met in other knowledge representation systems and analysis ways.

The following standard logic symbols are used generally:

For all	$\forall$
There exists	$\exists$
Implies	$\rightarrow$
Not	$\neg$
Or	$\vee$
And	$\wedge$

Now we provide an example of how predicate logic is accessed to represent knowledge. There are other methods but this form is well-liked.

Example: Consider the following:

- Sachin is a mega star.
- Mega stars are rich.
- Rich people have speedy cars.
- Fast cars take a lot of petrol.

and strive to sketch the conclusion: *Sachin's car takes a lot of petrol*.

Thus we can convert *Sachin is a mega star* into: *mega_star(sachin)* and *Mega stars are rich* into: $\forall$ *m: mega_star(m)* $\rightarrow$ *rich(m)*

Rich people contain fast cars, the third axiom is more complicated:

- Is *cars* a relation and so *car(c,m)* says that case *c* is *m's* car. OR
- Is *cars* a function? Thus we may have *car_of(m)*.

Consider that *cars* is a relation then axiom 3 may be written: $\forall$ *c,m: car(c,m)* $\wedge$ *rich(m) fast(c).*

The fourth axiom is a common statement regarding *fast cars*. Suppose *consume(c)* signify that car *c* takes a lot of petrol. So we may write: $\forall$ *c:* [*fast(c)* $\wedge\exists$ *m:car(c,m)* $\rightarrow$ *consume(c)* .

Is this enough? no! — Does sachin have a car? We want the *car_of* function after all (and addition to *car*): $\forall$ *c:car(car_of(m),m)*. The effect of applying *car_of*to *m* is *m*'s car. The concluding set of

Notes

predicates is: *mega_star(sachin)* $\forall$ *m: mega_star(m)* $\rightarrow$ *rich(m)* $\forall$ *c:car(car_of(m),m)*. $\forall$ *c,m: car(c,m)* $\wedge$ *rich(m)* $\rightarrow$ *fast(c)*. $\forall$ *c:* [*fast(c)* $\wedge\exists$ *m:car(c,m)* $\rightarrow$ *consume(c)*]. Provided this we could conclude: *consume(car_of(sachin))*.

5.3.1 Isa and Instance Relationships

Two traits isa and instance play an imperative role in many facets of knowledge representation. The cause for this is that they maintain property inheritance.

isa

– used to demonstrate class inclusion, such as isa(mega_star,rich).

instance

– used to demonstrate class membership, such as instance (prince,mega_star).

Thus, now it should be easy to observe how to represent these in predicate logic.

Isa is used to demonstrate class inclusion, such as isa(mega_star,rich).

Instance is used to demonstrate class membership, such as instance (prince,mega_star).

Self Assessment

Fill in the blanks:

9. Predicate logic contains various standard logic symbols to represent ..... .
10. Isa is used to demonstrate class ..... .
11. Instance is used to demonstrate class ..... .

5.4 Resolution

This is another type of proof system based on refutation. Better suited to computer implementation than systems of axioms and rules (can give correct answers). Generalizes to first order logic. This is the basis of Prologs inference method.

Caution To apply resolution, all formulae in the knowledge base and the query must be in clausal form (c.f. Prolog clauses).

5.4.1 Normal Forms

A literal is a propositional letter or the negation of a propositional letter and a clause is a disjunction of literals.

Conjunctive Normal Form (CNF): a conjunction of clauses, e.g., $(P \vee Q \vee \neg R) \wedge (\neg S \vee \neg R)$.

Disjunctive Normal Form (DNF): a disjunction of conjunctions of literals, e.g., $(\neg P \wedge \neg Q \wedge \neg R) \vee (\neg S \wedge \neg R)$.

Every propositional logic formula can be converted to CNF and DNF.

Conversion to Conjunctive Normal Form

Eliminate $\leftrightarrow\leftrightarrow$ rewriting $P \leftrightarrow\leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$

Notes

Eliminate → rewriting P →→ Q as ¬ P ∨ Q

Use DeMorgans laws to push ¬ inwards:

rewrite ¬ (P ∧ Q) as ¬P ∨ Q

rewrite ¬ (P ∧ Q) as P ∨ ¬Q

Eliminate double negations: rewrite ¬¬ P as P

Use the distributive laws to get CNF:

rewrite (P ∧ Q) ∨ R as (P ∨ R) ∧ (Q ∨ R)

rewrite (P ∨ Q) ∧ R as (P ∧ R) ∨ (Q ∧ R)

Example:

¬ (P → (Q ∧ R))

¬ (¬ P ∨ (Q ∧ R))

¬ ¬ P ∧ ¬ (Q ∨ R)

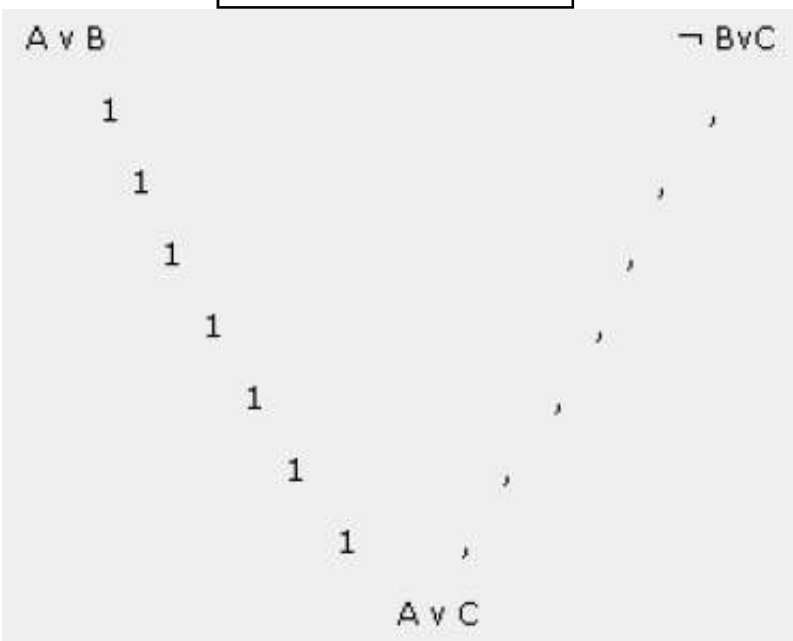
¬ ¬ P ∧ (¬ Q ∧ ¬ R)

P ∧ (¬ Q ∨ ¬ R)

Two clauses: P, ¬ Q ∨ ¬ R

5.4.2 Resolution Rule of Inference

Figure 5.4: Resolution Rule



where *B* is a propositional letter and *A* and *C* are clauses (possibly empty) *A("C is the resolvent of the two clauses.*

Notes

Resolution Rule: Key Idea

1. Consider $A \vee B$ and $\neg B \vee C$
   (a) if B is True, $\neg B$ is False and truth of second formula depends only on C
   (b) if B is False, truth of first formula depends only on A
2. Only one of B, $\neg B$ is True, so if both $A \vee B$ and $\neg B \vee C$ are True, either A or C is True, i.e., $A \vee C$ is True

   Hence the resolution rule is sound.

Task Make distinction between Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF).

5.4.3 Applying Resolution

1. The resolution rule is sound (resolving entailed by two parent clauses)
2. How can we use the resolution rule?
   (a) Convert knowledge base into clausal form
   (b) Repeatedly apply resolution rule to the resulting clauses
   (c) A query A follows from the knowledge base if and only if each of the clauses in the CNF of A can be derived using resolution.

Self Assessment

Fill in the blanks:

12. A ..... is a propositional letter or the negation of a propositional letter.
13. Resolution is considered as a basis of ..... method.

5.5 Unification Algorithm

In Artificial Intelligence there is the question whether we should pursue "unified architectures of cognition". Those of us who build AI systems know that some amount of unification happens automatically as a consequence of trying to simplify what we are building by making the components general and reusable through building libraries, languages, and architectures. And clearly that there has to be something better than Lisp!

Our feeling is that a total unification is not possible, but there is a strong reason why people believe it is. To put it simply, everyone has a different idea of what it means to be intelligent, that is, they have different beliefs about what kinds of problems are interesting or hard. This set of "interesting problems" is usually far smaller than the set of problems people actually solve, and so as a consequence unification may be possible for that small set. But when applied to problems outside that set, the unification begins to break down and a great deal of additional machinery is needed, defeating the original simplification.

Notes

Notes In other words, trying to unify AI is fruitless because every problem and domain has enough of its own character that any AI system that purports to be general in fact requires a significant amount of additional code and modification to get it to actually work in that domain. Anyone who doesn't believe this should try working in another domain!

5.5.1 Diversity of Commonsense Thinking

When we finally build a human-level commonsense thinking machine, it will be composed of at least dozens and possibly hundreds of distinct subsystems with rather different architectures, because it will be the union of a large number of architectures written by a large number of people, each with a different idea of what it means to be intelligent. So perhaps the question we should be asking is not

How do you unify all of AI into one cognitive architecture?

But rather

How do you get several cognitive architectures to work together?

How can we make different architectures work together without unifying them?

Perhaps you can do this without much trouble, if the individual architectures are skilled at figuring out how to use new kinds of resources. The brain's walking agency might exploit its vision agency to see if there are slippery things on the ground, and the vision agency might exploit the walking agency to get the robot around an obstacle to see its goal more clearly. But is this so different from the walking agency needing to figure out how to servo its joints to help it go forward, or the vision agency to figure out how to apply its filters and visual routines to segment an object? The skills that any smart AI system needs to manage and figure itself out may well be enough to let it exploit outside resources as well. Over time, a very tight integration may be achieved, where the vision agency may move the body and head for virtually every task it performs, and vice-versa. In order to achieve this level of cooperation and coordination, what is needed is less a common substrate on which to communicate, and more a large ensemble of "social", "team" and "exploitation" skills that allow specific sets of agents to cooperate with one another or exploit one another without slowing things down or drawing too many resources. There is a general mechanism for this – except maybe for a few features like those in Soar, such as the B-brain facility that detects different kinds of impasses. But even there, the facility is useless without the programmer providing some ways to deal with those impasses.

We should have diversity at the highest level, at the level of the cognitive architecture itself. As a practical matter, this is an alternative to the Soar approach, where you are expected to write all of the different problem solvers you need in the Soar language. Soar or any such substrate is well suited for every different kind of problem. That is why the robot people use consumption-like languages, the vision people have their probabilistic frameworks, etc. It doesn't mean that those methods are sufficient, but it shows how hard it is to get people to agree on a language or representation. And in fact, software engineers have given up on unification. These days, large programs are written in many different languages, and linked using "interface definition languages", which amount to abstract descriptions of the data structures each language uses. Because every language uses integers, floats, arrays, and records, they can talk. And because any AI system will need to use situations, events, objects, properties, relations, and other basic representations, they will be able to talk as well. Chances are, half a billion years of evolution didn't produce a brain that could be effectively expressed as a small set of simple agencies built

on a simple substrate. Perhaps we could invent such a system, but maybe we should wait until we've got something that works before we try to simplify it that much.

Self Assessment

Fill in the blanks:

14. When applied to problems outside the specified set, the ..... begins to break down and a great deal of additional machinery is needed, defeating the original simplification.
15. These days, large programs are written in many different languages, and linked using "..... languages".

5.6 Summary

- Knowledge management is a critical component of an organizations success. Knowledge assets are the knowledge that an organization owns or needs to own, to achieve its goals.
- Explicit knowledge is used in the design of routines, standard operation procedures, and the structure of data records.
- Tacit knowledge can be attained through dialogue, job shadowing, storytelling, and sharing of best practices and lessons learned.
- Knowledge representation is an area in artificial intelligence that is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse".
- Knowledge acquisition can be considered as one of the technologies contributing to the development of knowledge-based Systems (KBS).
- A semantic network is a network which represents semantic relations between the concepts. This is often used as a form of knowledge representation.
- Resolution is another type of proof system based on refutation which is better suited to computer implementation than systems of axioms and rules.
- Trying to unify AI is fruitless because every problem and domain has enough of its own character that any AI system that purports to be general in fact requires a significant amount of additional code and modification to get it to actually work in that domain.

5.7 Keywords

Explicit Knowledge: Knowledge that can be more easily attained and is often expressed or documented in a formal, systematic manner.

Frame: A frame is an artificial intelligence data structure used to divide knowledge into substructures by representing "stereotyped situations."

Knowledge Acquisition: Knowledge acquisition can be considered as one of the technologies contributing to the development of knowledge-based Systems (KBS).

Knowledge Representation: It is an area in artificial intelligence that is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse".

Knowledge: Knowledge is information extracted, filtered or formatted in some way.

Notes

Semantic Network: A semantic network is a network which represents semantic relations between the concepts. This is often used as a form of knowledge representation.

Tactic Knowledge: Knowledge that can also be attained, but is not as easily transferred.

5.8 Review Questions

1. Make distinction between explicit knowledge and tactic knowledge.
2. Describe the meaning of knowledge representation in clausal form.
3. Explain the approaches of knowledge representation.
4. Explain the concept of semantic network with examples.
5. Illustrate how do you use Predicate Logic to represent knowledge.
6. What is knowledge acquisition? Discus how Knowledge acquisition is considered as one of the technology contributing to the development of knowledge-based Systems (KBS).
7. Using the vocabulary (i.e. predicate symbols, constants, and function symbols) express the following sentences in clausal form:
   (a) X is a mother of Y if X is female and X is a parent of Y.
   (b) X is a father of Y if X is a male parent of Y.
   (c) X is human if Y is a parent of X and Y is human.
   (d) An individual is human if his or her mother is human and his or her father is human.
   (e) If a person is human then his (or her) mother is human or his (or her) father is human.
   (f) No one is his or her own parent.
8. Assume the following facts:
   (a) Robin only likes easy courses.
   (b) Computing courses are hard.
   (c) All courses in Sociology are easy.
   (d) "Society is evil" is a sociology course.

   Represent these facts in predicate logic and answer the question. What course would Robin like?
9. What is resolution? Illustrate the concept of applying resolution.
10. Explain the working of unification algorithm.

Answers: Self Assessment

1.	Explicit	2.	Tacit
3.	knowledge-based Systems (KBS)	4.	Knowledge Representation
5.	Frame	6.	Information
7.	If-then	8.	Semantic
9.	Knowledge	10.	Inclusion

Notes

11. Membership
12. Literal
13. Prologs Inference
14. Unification
15. Interface Definition

5.9 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

Notes

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.dave-reed.com/csc550.F04/Lectures/representation.pdf

Unit 6: Knowledge Representation using Rules

Notes

CONTENTS

Objectives
Introduction
6.1 Procedural vs Declarative Knowledge
6.1.1 Representing How to Use Knowledge
6.2 Logic Programming
6.3 Forward vs Backward Reasoning
6.4 Matching and Control Knowledge
6.4.1 Matching
6.4.2 Control Knowledge
6.5 Summary
6.6 Keywords
6.7 Review Questions
6.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the procedural and declarative knowledge
- Illustrate the logic programming
- Compare the forward and backward reasoning
- Understand the matching and control knowledge

Introduction

In this unit, you will understand various concepts of knowledge representation using rules. You will discuss procedural knowledge and declarative knowledge. Logic programming contain three major classes of application which are discussed in the unit. You will illustrate the comparison between Forward and Backward Reasoning. The concept of matching control knowledge is also discussed.

6.1 Procedural vs Declarative Knowledge

Declarative Knowledge Representation

In Declarative knowledge representation, knowledge is signified as static collection of details which are influenced by common procedures. Here the details are required to be accumulated only one and they can be utilized in any number of manners. Details can be easily added to declarative systems without altering the common procedures.

Notes

- ***Static representation*** — knowledge regarding objects, events, *etc.,* and their relationships and states specified.
- Needs a program to recognize what to do with knowledge and how to perform it.

Procedural Representation

In Procedural representation, knowledge is signified as procedures.

Example: Default reasoning and probabilistic reasoning are examples of procedural techniques.

Here, heuristic knowledge of "How to do things efficiently "can be simply signified.

- Control information essential to use the knowledge is entrenched in the knowledge itself. For example, how to discover relevant facts, make inferences etc.
- Needs an interpreter to follow instructions provided in knowledge.

Example: Let us assume what knowledge an alphabetical sorter would want:

- Implicit knowledge that *A* occurs before *B etc.*
- This is simple really integer comparison of (ASCII) codes for A, B.
  - All programs enclose procedural knowledge of this sort.
- The procedural information here is that knowledge of *how to alphabetise* is signified explicitly in the alphabetization procedure.
  - A declarative system might have to have explicit details such as *A* occurs before *B*, *B* comes before *C* etc.

6.1.1 Representing How to Use Knowledge

We discuss below the need to represent *how to control* the processing:

Direction

- Specify the direction an allegation could be used. For example, to verify something can fly show it is a bird. $fly(x) \rightarrow bird(x)$.

Knowledge to Achieve Goal

- Identify what knowledge might be required to attain a specific goal.

Example: To prove something is a bird attempting using two facts *has_wings* and *has_feathers* to demonstrate it.

Actually most of the knowledge representations utilize an amalgamation of both. Most of the knowledge representation structures have been generated to manage programs that control natural language input. There are numerous types of schemas that have proved functional in AI programs. They comprise:

(i) *Frames:* Used to explain a compilation of attributes that a specified object possesses (e.g.: depiction of a chair).

(ii) *Scripts:* Used to illustrate general sequence of events (e.g.:- a restaurant scene).

Notes

(iii) *Stereotypes:* Used to depict traits of people.

(iv) *Rule models:* Used to illustrate general features shared between a set of rules in a production system.

Frames and scripts are used very broadly in various AI programs. Before choosing any particular knowledge representation structure, the following concerns have to be considered.

(i) The basis properties of objects, if any, which are general to each problem domain must be identified and managed suitably.

(ii) The whole knowledge should be signified as a good set of primitives.

(iii) Mechanisms must be developed to access appropriate parts in a huge knowledge base.

Task Illustrate the use of frames and scripts.

Self Assessment

Fill in the blanks:

1. In ..... representation, knowledge is signified as static collection of details which are influenced by common procedures.
2. Details can be easily added to ..... systems without altering the common procedures.
3. In Procedural representation, knowledge is signified as ..... .
4. Most of the knowledge representation structures have been generated to manage programs that control ..... .
5. ..... are used to illustrate general sequence of events.
6. ..... models are used to illustrate general features shared between a set of rules in a production system.

6.2 Logic Programming

Logic programming contain three major classes of application: as a general-purpose programming language, a database language, and a knowledge representation language in AI. As a programming language, it can signify and calculate any assessable function. As a database language, it simplifies relational databases, to comprise general clauses as well as facts. And as a knowledge representation language it is a non-monotonic logic, which can be utilized for default reasoning. Its most renowned execution is the programming language Prolog, which mingles pure logic programming with a number of contaminated traits.

Besides the utilization of logic programming as a normative model of problem solving (Kowalski, 1974/79), Stenning and van Lambalgen (2004, 2005, 2008) have examined its use as a descriptive model of human reasoning.

Logic programs (also known as normal logic programs) are sets of conditionals of the form:

If B1 and … and Bn then H

Notes

where the conclusion H is an atomic formula and the conditions Bi are literals, which are either atomic formulas or the negations of atomic formulas. All variables are implicitly generally quantified in front of the conditional. Conditionals in logic programs are also known as clauses. Horn clauses are the particular case where all of the conditions are atomic formulae. Details are the special case where n = 0 (there are no conditions) and there are no variables. At times clauses that are not facts are also known as rules, inviting confusion with production rules.

Goals (or queries) are combination of literals, syntactically just like the conditions of clauses.

Though, all variables are unreservedly existentially quantified, and the purpose of the goal is to discover an instantiation of the variables that makes the objective hold.

Example: Consider the three sentences:

If you contain the bus fare and you catch a bus and not something goes wrong with the bus voyage, then you will go home for the weekend. 'If you have the bus fare and you catch a bus', 'then you will go home for the weekend'; 'You have the bus fare' are a clause, a Horn clause, and a fact correspondingly.

Observe that the second sentence can be considered as an imprecise version of the first sentence. Observe too that the first two clauses both express "strong" domain-specific knowledge, instead of the kind of weak knowledge that would be essential for general-purpose planning.

The sentence you will go home for the weekend is a simple, atomic goal.

Backward reasoning (from conclusions to conditions) considers conditionals as objective-reduction procedures: to show/solve H, show/solve B1 …. Bn.

For instance, backward reasoning turns the conditionals:

If you study late in the library then you will finish the essay.

If you have the bus fare and you catch a bus, then you will go home for the weekend into the procedures:

To complete the essay, study late in the library.

To go home for the weekend, verify that you have the bus fare, and catch a bus.

Since conditionals in normal logic programming are utilized only backwards, they are usually written backwards:

H if B1 …. Bn

So that backward reasoning is correspondent to "forward chaining" in the direction in which the conditional is written. The Prolog syntax for clauses:

H :- B1 ,… , Bn

is intentionally ambiguous, so that clauses can be read either declaratively as conditionals written backwards or procedurally as goal-reduction procedures implemented forwards.

While positive, atomic goals and sub-goals are resolved by backward reasoning, negative goals and sub-goals of the form not G, where G is an atomic sentence, are resolved by negation as failure: not G succeeds if and only if backward reasoning with the sub-goal G does not succeed.

Negation as failure makes logic programming a non-monotonic logic.

Example: Given only the clauses:

An object is red if the object appears red and not the object is illuminated by a red light.

Notes

The apple appears red. Then the consequence:

The apple is red follows as a goal, since there is no clause whose conclusion matches the sub-goal the apple is enlightened by a red light, and thus the two conditions for the only clause that can be utilized in solving the goal both hold. Though, provided the additional clause the apple is illuminated by a red light, the sub-goal now accomplish something and the top-level goal now fails, non-monotonically withdrawing the consequence the apple is red. Though, the consequence not the apple is red now succeeds instead.

Goals and conditions of clauses can be generalized from conjunctions of literals to arbitrary formulae of first-order logic. The simplest manner to perform so is to use auxiliary predicates and clauses (Lloyd and Topor, 1984).

Example: The goal:

Demonstrate that for all exams, if the exam is a final exam, then you can revise for the exam in the library can be converted into the normal logic programming form:

Demonstrate that not the library is useless; the library is futile if the exam is a final exam and not you can study for the exam in the library.

Caution The conversion applies in logic programming only when the conditional is inferred as a goal, and not when it is inferred as a clause.

The computational benefit of the conversion is that it decreases the problem of identifying whether a random sentence of first-order logic holds with respect to a specified logic program to the two simple inference rules of backward reasoning and negation as breakdown alone.

Notes Horn clauses are named after the logician Alfred Horn, who studied some of their model-theoretic properties.

Comparison between Backward and Forward reasoning is discussed below.

Self Assessment

Fill in the blanks:

7. ..... contains three major classes of application: as a general-purpose programming language, a database language, and a knowledge representation language in AI.
8. As a database language, logic programming simplifies ..... databases, to comprise general clauses as well as facts.
9. Conditionals in logic programs are also known as ..... .
10. ..... are the particular case where all of the conditions are atomic formulae.

6.3 Forward vs Backward Reasoning

A search procedure must locate a path among initial and goal states. There are two directions in which a search process could progress.

Notes

1. ***Reason forward from the initial states:*** Being formed the root of the search tree. General the next level of the tree by locating all the rules whose left sides go with the root node, and use their right sides to produce the siblings. Repeat the process until a configuration that goes with the goal state is produced.
2. ***Reason forward from the goal state(s):*** Start building a search tree starting with the goal configuration(s) at the root. Produce the next level of the tree by locating all the rules whose right sides go with the root node. Use the left sides of the rules to produce the new nodes. Persist until a node that goes with the begin state is produced.

Did u know? The method of chaining backward from the preferred final state is known as goal directed reasoning or back tracing.

Assortment of forward reasoning or backward reasoning is based on which direction provides less branching factor and justifies its reasoning process to the consumer. Most of the search methods can be utilized to search either forward or backward.

Notes One exception is the means-ends analysis technique which continues by decreasing differences among current and goal states, at times reasoning forward and at times backward.

The following are the factors which find out the option of direction for a specific problem:

1. Are there more possible start states on goal states? We would like to shift from the smaller set of states to the bigger set of states.
2. In which direction is the branching factor (that is, their average number of nodes that can be reached directly from a single node) greater ? we would like to carry on in the direction with the lower branching factor.
3. Will the program be inquired to validate its reasoning process to a user? If so, it is imperative to continue in the direction that matches more closely with the way the user will think.
4. What type of event is going to generate a problem-solving episode? If it is the arrival of a new factor, forward reasoning makes sends. If it is a query to which a response is preferred, backward reasoning is more normal.

Self Assessment

Fill in the blanks:

11. A search procedure must locate a path among initial and ..... states.
12. The method of chaining backward from the preferred final state is known as .....

6.4 Matching and Control Knowledge

6.4.1 Matching

How to take out from the whole collection of rules that can be applied at a specified point?

Notes

Matching among current state and the precondition of the rules is discussed as below.

- Indexing
  - A large number of rules ⇒ too sluggish to locate a rule
  - Indexing: Use the existing state as an index into rules and choose the matching ones instantly
  - Only functions when preconditions of rules match accurately
  - Only functions when preconditions of rules match exact board configuration
  - It's not forever apparent whether a rule's preconditions are pleased by a specific state
  - There's a swapping among the ease of writing rules (high-level descriptions) and the ease of the matching process
- Matching with variables
  - *Generality in the statements of the rules:* Require a search process to determine a match among a particular state and the preconditions of a specified rule.
- Backward-chaining systems
  - One-one matching algorithm
    - Unification procedure + Depth-first backtracking to select individual rules
    - Forward-chaining systems
  - Many-many matching algorithm: RETE
- Approximate matching
  - Rules should be applied if their preconditions roughly match the present situation

Example: A speech-understanding program

- ***Rules:*** An explanation of a physical waveform to phones (a, e, …)
- ***Physical signal:*** differences in the way individuals speak, result of background noise, etc.
- ***Conflict resolution:***
  - Preferences dependent on rules:
    - Specificity of rules
    - Physical order of rules
  - Preferences dependent on objects:
    - Importance of objects
    - Position of objects
  - Preferences dependent on states:
    - Assessment of states

Notes

6.4.2 Control Knowledge

An algorithm comprises: logic component, that mentions the knowledge to be utilized in solving problems, and manage component, that identifies the problem-solving approaches by means of which that knowledge is utilized.

Therefore, Algorithm = Logic + Control. The logic component identifies the meaning of the algorithm while the control component only affects its competence. An algorithm may be formulated in dissimilar manners, generating similar behavior. One formulation, may have a apparent statement in logic component but utilize a complicated problem solving strategy in the control component. The other formulation may have a complex logic component but utilize a simple problem-solving approach. The competence of an algorithm can frequently be improved by enhancing the control component without altering the logic of the algorithm and thus without altering the meaning of the algorithm.

The approach in databases is towards the division of logic and control. The programming languages these days do not differentiate between them.

Computer programs will be more frequently accurate, more simply enhanced, and more readily adapted to new troubles when programming languages divide logic and control, and when execution mechanisms offer more powerful problem-solving amenities of the type given by intelligent theorem-proving systems.

Did u know? The programmer mentions both logic and control in a single language.

Caution The execution mechanism utilizes only the most rudimentary problem-solving capabilities.

Task Discuss how to improve the competence of an algorithm.

Self Assessment

Fill in the blanks:

13. The ..... component identifies the meaning of the algorithm while the control component only affects its competence.
14. The ..... of an algorithm can frequently be improved by enhancing the control component without altering the logic of the algorithm and thus without altering the meaning of the algorithm.
15. The approach in ..... is towards the division of logic and control.

6.5 Summary

- In Declarative knowledge representation, knowledge is signified as static collection of details which are influenced by common procedures.

Notes

- Details can be easily added to declarative systems without altering the common procedures.
- In Procedural representation, knowledge is signified as procedures. Default reasoning and probabilistic reasoning are examples of procedural techniques.
- In Procedural representation, heuristic knowledge of "How to do things efficiently "can be simply signified.
- Most of the knowledge representation structures have been generated to manage programs that control natural language input.
- Logic programming contain three major classes of application: as a general-purpose programming language, a database language, and a knowledge representation language in AI.
- Assortment of forward reasoning or backward reasoning is based on which direction provides less branching factor and justifies its reasoning process to the consumer.
- The logic component identifies the meaning of the algorithm while the control component only affects its competence.

6.6 Keywords

Declarative Knowledge Representation: In Declarative knowledge representation, knowledge is signified as static collection of details which are influenced by common procedures.

Frames: Used to explain a compilation of attributes that a specified object possesses (For example, depiction of a chair).

Procedural Representation: In Procedural representation, knowledge is signified as procedures.

Scripts: Scripts are used to illustrate general sequence of events (For example, a restaurant scene).

Stereotypes: Stereotypes are used to depict traits of people.

6.7 Review Questions

1. Make distinction between Declarative knowledge representation and procedural knowledge representation.
2. What does static representation signify? Discuss.
3. Illustrate the concept of Procedural and Declarative Knowledge with the help of example.
4. Explicate the process of using knowledge.
5. Elucidate the different types of schemas that have proved functional in AI programs.
6. Describe the major classes of application used in logic programming.
7. Illustrate the concept of logic programing by means of an example.
8. Make distinction between Forward vs Backward Reasoning. Give examples.
9. Illustrate the factors which find out the option of direction for a specific problem.
10. Illustrate the use of logic component that is used in an algorithm.

Notes

Answers: Self Assessment

1.	Declarative Knowledge	2.	Declarative
3.	Procedures	4.	Natural Language Input
5.	Scripts	6.	Rule
7.	Logic Programming	8.	Relational
9.	Clauses	10.	Horn clauses
11.	Goal	12.	Back tracing
13.	Logic	14.	Competence
15.	Databases		

6.8 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Notes

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.dave-reed.com/csc550.F04/Lectures/representation.pdf

Notes

Unit 7: Symbolic Reasoning under Uncertainty

CONTENTS

Objectives

Introduction

7.1 How can we reason?

7.2 Uncertain Reasoning

7.3 Non-Monotonic Reasoning

7.4 Default Reasoning

7.5 Circumscription

7.5.1 Implementations: Truth Maintenance Systems

7.6 Summary

7.7 Keywords

7.8 Review Questions

7.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concepts of non-monotonic reasoning
- Discuss the default reasoning
- Illustrate the concept of circumscription

Introduction

When we need any knowledge system to accomplish something it has not been clearly informed how to do, it must *reason*. The system must work out what it requires to recognize from what it already be familiar with. We have observed easy instance of reasoning or sketching inferences already. For example, if we are familiar with: *Robins are birds*. *All birds have wings*. Then if we inquire: *Do robins have wings?* Some *reasoning* has to go on respond to the question.

7.1 How can we Reason?

To some extent this will be based on the knowledge representation selected. Although a good knowledge representation system has to permit easy, normal and believable reasoning. We have discussed below very wide techniques of how we may reason.

1. ***Formal reasoning***: It implies to basic rules of inference along with logic knowledge representations.
2. ***Procedural reasoning:*** Procedural reasoning uses procedures that state *how to possibly solve* (sub) problems.

Notes

3. ***Reasoning by analogy:*** Humans are good at this, more complicated for AI systems. E.g. If we are inquired *Can robins fly?* The system may reason that *robins are like sparrows* and it knows *sparrows can fly* .
4. ***Generalization and abstraction:*** Again humans are efficient at this. This is fundamentally obtaining towards *learning* and *understanding* techniques.
5. ***Meta-level reasoning:*** Once again utilizes knowledge regarding what you know and possibly ordering it in some type of significance.

Task Illustrate the concept of reasoning by analogy.

Self Assessment

Fill in the blanks:

1. ..... reasoning implies to basic rules of inference along with logic knowledge representations.
2. ..... reasoning uses procedures that state *how to possibly solve* (sub) problems.

7.2 Uncertain Reasoning

Regrettably the world is an uncertain place. Any AI system that looks for a model and reasoning in such a world must be able to deal with this.

Particularly it must be able to contract with:

- Incompleteness — compensate for be deficiency of knowledge.
- Inconsistencies — resolve indistinctness and contradictions.
- Change — it must be able to modernize its *world knowledge base* over time.

Notes Obviously so as to deal with this, some decision that is made are more possible to be true (or false) than others and we must bring in techniques that can manage with this *uncertainty*.

There are three fundamental methods that can do this:

- Symbolic methods.
- Statistical methods.
- Fuzzy logic methods.

Self Assessment

Fill in the blank:

3. ..... lead to compensate for deficiency of knowledge.

Notes

7.3 Non-Monotonic Reasoning

In monotonic reasoning if we expand at set of axioms we cannot withdraw any present declarations or axioms.

Example: Predicate logic and the conclusions we execute on it is an example of *monotonic* reasoning.

Humans do not hold to this monotonic structure when reasoning:

- We are required to jump to conclusions so as to plan and, more fundamentally, survive.
  - We cannot expect all potential results of our plan.

Caution We must make suppositions regarding things we do not purposely recognize about.

7.4 Default Reasoning

This is a very general from of non-monotonic reasoning. Here *we want to sketch conclusion based on what is most probable to be correct.*

We will converse about two strategies to do this:

- Non-Monotonic logic.
- Default logic.

Caution Do not get puzzled regarding the label *Non-Monotonic* and *Default* being useful to reasoning and a specific logic.

Did u know? Non-Monotonic reasoning is common depiction of a class of reasoning.

Non-Monotonic logic is a particular theory. The same is considered for Default reasoning and Default logic.

Non-Monotonic Logic

This is fundamentally an extension of first-order predicate logic to comprise a *modal* operator, *M*. The reason of this is to permit for steadiness.

Example:

$\forall x$: plays_instrument(x) $\wedge$M improvises(x) $\rightarrow$ jazz_musician(x)

specifies that for all x, x plays an instrument and if the truth that x can manage is reliable with all other knowledge then we can conclude that x is a jazz musician.

Now we will define consistency:

Notes

One general solution (consistent with PROLOG notation) is to demonstrate that fact P is accurate effort to prove $\neg P$. If we do not succeed we may say that P is consistent (because $\neg P$ is false).

However assume the well-known set of assertions connecting to President Nixon.

$\forall x$: Republican(x) $\wedge M\neg$ Pacifist(x) $\rightarrow$ –Pacifist(x)

$\forall x$: Quaker(x) $\wedge M$ Pacifist(x) Pacifist(x)

Now this specifies that Quakers is apt to be pacifists and Republicans tend not to be.

But Nixon was both a Quaker and a Republican and thus we could assert:

Quaker(Nixon)

Republican(Nixon)

This now leads to our total knowledge turning out to be inconsistent.

Default Logic

Default logic brings in a new inference rule:

$$\frac{A \cdot B}{C}$$

which specifies if A is deducible and it is consistent to presume B then conclude C.

Notes Now this is alike to Non-monotonic logic but there are a number of distinctions:

- New inference rules are utilized for calculating the set of probable extensions. So, in the Nixon example above Default logic can assist both assertions as is does not say anything regarding how to select between them — it will rely on the inference being made.
- In Default logic any non-monotonic expressions are rules of inference rather than expressions.

Self Assessment

Fill in the blanks:

4. In ..... reasoning if we expand at set of axioms we cannot withdraw any present declarations or axioms.
5. In ..... reasoning, we want to sketch conclusion based on what is most probable to be correct.
6. Non-Monotonic reasoning is common depiction of a class of ..... .
7. ..... is fundamentally an extension of first-order predicate logic to comprise a modal operator, M.
8. Default logic brings in a new inference rule: ..... which specifies if A is deducible and it is consistent to presume B then conclude C.
9. ..... and the conclusions we execute on it is an example of monotonic reasoning.

Notes

7.5 Circumscription

Circumscription is a rule of speculation that permits you to come to the conclusion that the objects you can demonstrate that posses some property, *p*, are actually all the objects that posses that property.

Circumscription can also deal with default reasoning.

Assume we know: bird(tweety)

$\forall x$: penguin(x) bird(x)

$\forall x$: penguin(x) flies(x)

and we desire to insert the truth that *naturally, birds fly*.

In circumscription this phrase would be specified as:

A bird will fly if it is not abnormal

and can therefore be represented by:

$\forall x$: bird(x) $\wedge\neg$ abnormal(x) $\rightarrow$ flies(x).

Though, this is not enough, we cannot conclude

flies(tweety)

as we cannot prove

abnormal(tweety).

This is where we pertain circumscription and, in this case, *we will suppose that those things that are exposed to be abnormal are the only things to be abnormal*

So we can rewrite our *default rule* as:

$\forall x$: bird(x) $\wedge\neg$ flies(x) $\rightarrow$ abnormal(x)

and add the following

$\forall x$: abnormal(x)

as there is nothing that cannot be exposed to be abnormal.

If we now insert the truth:

penguin(tweety)

Evidently we can prove

abnormal(tweety).

If we circumscribe abnormal now we would adjoin the sentence,

a penguin (tweety) is the abnormal thing:

$\forall x$: abnormal(x) $\rightarrow$ penguin(x).

Did u know? The difference between Default logic and circumscription:

Defaults are sentences in language itself, not supplementary inference rules.

Notes

7.5.1 Implementations: Truth Maintenance Systems

A multiplicity of *Truth Maintenance Systems* (TMS) have been generated as a means of executing Non-Monotonic Reasoning Systems.

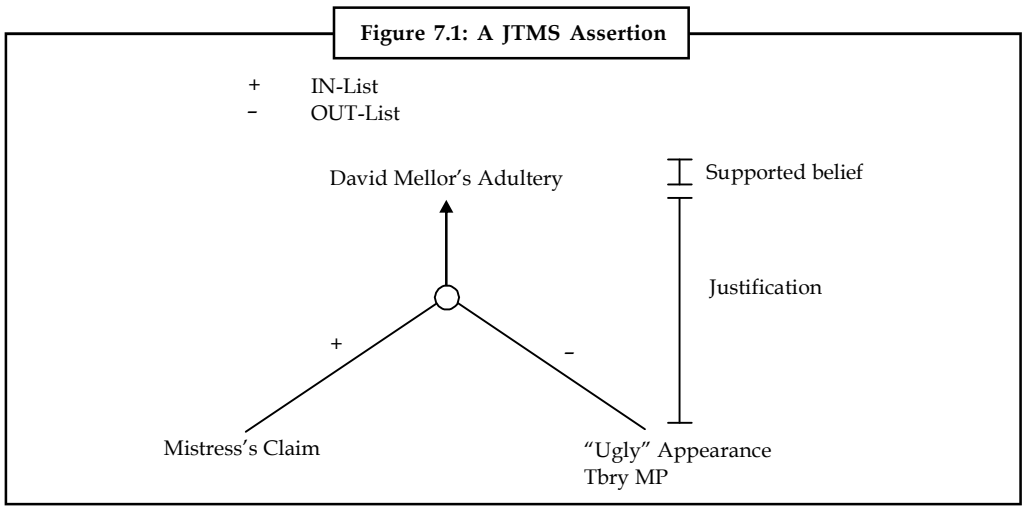
Fundamentally Truth Maintenance Systems (TMSs):

- All do some kind of dependency directed backtracking
- Assertions are connected through a network of dependency.

Justification-based Truth Maintenance Systems (JTMS)

- This is a simple TMS such that it does not recognize anything regarding the structure of the assertions themselves.
- Every supported belief (assertion) in has a explanation.
- Each justification consists of two parts:
  - An *IN-List* — which supports beliefs held.
  - An *OUT-List* — which supports beliefs *not* held.
- An assertion is associated to its justification via an arrow.
- One assertion can *feed* a different explanation thus building the network.
- Assertions may be labelled with a *belief status*.
- An assertion is *valid* if every declaration in the IN-List is supposed and none in the OUT-List are believed.
- An declaration is non-monotonic is the OUT-List is not empty or if any declaration in the IN-List is non-monotonic.

Figure 7.1: A JTMS Assertion



Logic-based Truth Maintenance Systems (LTMS)

Logic-based Truth Maintenance Systems (LTMS) is similar to JTMS except:

- Nodes (assertions) presume no relationships between them except ones overtly specified in justifications.

Notes

- JTMS can symbolize P and P concurrently. An LTMS would throw a challenge here.
- If this happens network has to be reconstructed.

Assumption-based Truth Maintenance Systems (ATMS)

- JTMS and LTMS follow a single line of reasoning at a time and backtrack (dependency-directed) when required — *depth first search*.
- ATMS sustain alternative paths in parallel — *breadth-first search*
- Backtracking is averted at the cost of sustaining numerous contexts.
- On the other hand as reasoning continues, disagreements occur and the ATMS can be *pruned*
  - Just find assertion with no suitable justification.

Task Make distinction between Justification-based Truth Maintenance Systems (JTMS) and Logic-based Truth Maintenance Systems (LTMS).

Self Assessment

Fill in the blanks:

10. ..... is a rule of speculation that permits you to come to the conclusion that the objects you can demonstrate that posses some property, *p*, are actually all the objects that posses that property.
11. In ....., assertions are connected through a network of dependency.
12. ..... is a simple TMS such that it does not recognize anything regarding the structure of the assertions themselves.
13. ..... is similar to JTMS except nodes (assertions) presume no relationships between them except ones overtly specified in justifications.
14. JTMS and LTMS follow a ..... line of reasoning at a time and backtrack (dependency-directed) when required — *depth first search*.
15. ..... sustain alternative paths in parallel — breadth-first search.

7.6 Summary

- When we need any knowledge system to accomplish something it has not been clearly informed how to do, it must reason.
- In monotonic reasoning if we expand at set of axioms we cannot withdraw any present declarations or axioms.
- Default reasoning is a very general from of non-monotonic reasoning where we want to sketch conclusion based on what is most probable to be correct.
- Non-Monotonic reasoning is common depiction of a class of reasoning.
- Non-Monotonic Logic is fundamentally an extension of first-order predicate logic to comprise a *modal* operator, *M*.

Notes

- In Default logic any non-monotonic expressions are rules of inference rather than expressions.
- Circumscription is a rule of speculation that permits you to come to the conclusion that the objects you can demonstrate that posses some property, *p*, are actually all the objects that posses that property.
- A multiplicity of *Truth Maintenance Systems* (TMS) have been generated as a means of executing Non-Monotonic Reasoning Systems.

7.7 Keywords

Circumscription: Circumscription is a rule of speculation that permits you to come to the conclusion that the objects you can demonstrate that posses some property, *p*, are actually all the objects that posses that property.

Default Logic: In Default logic any non-monotonic expressions are rules of inference rather than expressions.

Default Reasoning: Default reasoning is a very general from of non-monotonic reasoning where we want to sketch conclusion based on what is most probable to be correct.

Monotonic Reasoning: In monotonic reasoning if we expand at set of axioms we cannot withdraw any present declarations or axioms.

Non-Monotonic Logic: Non-Monotonic Logic is fundamentally an extension of first-order predicate logic to comprise a modal operator, *M*.

7.8 Review Questions

1. Make distinction between formal reasoning and procedural reasoning.
2. Discuss the factors with which Uncertain Reasoning deals with.
3. What is Non-Monotonic Reasoning? Give examples.
4. Identify the approaches used to perform default reasoning.
5. What is default logic?
6. What does Circumscription signify? Illustrate.
7. Illustrate how Circumscription deals with default reasoning.
8. What are Truth Maintenance Systems (TMSs)? Explicate its various types.
9. Consider the problem of deciding which clothes to wear using knowledge such as:
   (a) Wear casual clothes unless they are not clean or important meeting occurs today.
   (b) Wear a Sweater if it is cold.
   (c) The winter is usually cold.
   (d) Wear shorts if it's warm.
   (e) The summer is usually warm.
      (i) Construct a JTMS network to symbolize these facts.
      (ii) Try to solve the problem *In winter do I wear shorts?*
      (iii) Answer the question *What shall I wear today* (You may assume that the system knows the time of year).

Notes

10. Create as JTMS and ATMS to represent the following:
    (a) If you have spots and a temperature you have measles.
    (b) If you have a liquid nose then unless it is hay fever season you have a cold.

Answers: Self Assessment

1.	Formal	2.	Procedural
3.	Incompleteness	4.	monotonic
5.	Default	6.	reasoning
7.	Non-Monotonic Logic	8.	$\frac{A \cdot B}{C}$
9.	Predicate logic	10.	Circumscription

11. Truth Maintenance Systems (TMSs)
12. Justification-Based Truth Maintenance Systems (JTMS)
13. Logic-Based Truth Maintenance Systems (LTMS)
14. Single
15. Assumption-Based Truth Maintenance Systems (ATMS)

7.9 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS: *A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Notes

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.springer.com

Notes

Unit 8: Statistical Reasoning

CONTENTS

Objectives

Introduction

8.1 Probability and Bayes Theorem

8.1.1 Probability

8.1.2 Bayes Theorem

8.2 Certainty Factors and Rule Based Systems

8.2.1 Reasoning with Certainty Factors

8.2.2 Overcoming the Bayes Rule Shortcomings

8.3 Bayesian Networks

8.3.1 Implementation

8.3.2 Reasoning in Bayesian Nets

8.4 Fuzzy Logic

8.4.1 Fuzzy Set Theory

8.5 Summary

8.6 Keywords

8.7 Review Questions

8.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the probability & Bayes theorem
- Discuss the certainty factors and rule based systems
- Illustrate the Bayesian network
- Understand the fuzzy logic and applications

Introduction

The (Symbolic) methods fundamentally symbolize uncertainty principle as being True, False, or Neither True nor False. Some methods also had problems with Incomplete Knowledge and Contradictions in the knowledge.

Statistical methods give a method for showing principles that are not certain (or uncertain) but for which there may be some assisting (or contradictory) confirmation. Statistical methods propose benefits in two wide scenarios: The first one is **Genuine Randomness where** card games are a good instance. We may not be competent to forecast any outcomes with certainty but we have knowledge regarding the possibility of certain items (such as like being dealt an ace) and we can exploit this. The second one is **Exceptions.** Symbolic methods can symbolize this. However,

Notes

if the number of exceptions is huge such system is apt to break down many common sense and expert reasoning tasks for instance. Statistical techniques can summarize huge exceptions without resorting enumeration.

8.1 Probability and Bayes Theorem

8.1.1 Probability

The fundamental approach statistical methods approve to deal with uncertainty is through the axioms of probability:

- Probabilities are (real) numbers in the range 0 to 1.
- A probability of $P(A) = 0$ symbolizes total uncertainty in A, $P(A) = 1$ total certainty and values in among some degree of (un)certainty.
- Probabilities can be computed in a number of manners.

Probability = (number of desired outcomes)/(total number of outcomes)

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

If you have a option of number of items k from a set of items n then the $C_k^n = \frac{n!}{k!(n-k)}$ formula is applied to discover the number of methods of making this option. (! = factorial).

Thus the possibility of winning the national lottery (choosing 6 from 49) is $\frac{49!}{6!43!}$ = 13,983,816 to 1.

Notes Conditional probability, $P(A \mid B)$, signifies the probability of event A specified that we know event B has appeared.

Task Illustrate the concept of probability.

8.1.2 Bayes Theorem

This specifies:

$$P(H_i / E) = \frac{P(E \mid H_i)P(H_i)}{\sum_{k=1}^{n} P(E \mid H_k)P(H_k)}$$

This signifies that specified some evidence E then probability that hypothesis H_i is true is equal to the proportion of the probability that E will be true specified H_i times the *a priori* evidence on the probability of H_i and the sum of the probability of E over the set of all hypotheses times the probability of these hypotheses.

Notes

Therefore to find if we scrutinize medical evidence to analyze an illness. We must know all the preceding probabilities to locate symptom and also the probability of having an illness depending on certain symptoms being observed.

Caution The set of all hypotheses must be mutually exclusive and comprehensive.

Did u know? Bayesian statistics occurs at the heart of many statistical reasoning systems.

How is Bayes theorem exploited?

- The key is to invent problem properly:

  $P(A \mid B)$ specifies the probability of A specified only B's evidence. *If* there is other relevant evidence then it **must** also be taken into account.

Herein occurs a problem:

- All events must be *mutually exclusive*. However in actual world problems events are not normally unrelated.

Example: In detecting measles, the indications of spots and a fever are associated. This signifies that computing the conditional probabilities gets multifaceted.

Usually if a prior evidence, p and some new inspection, N then computing

$$P(H \mid N_1 p) = P(H \mid N)\frac{P(p \mid N_1 H)}{P(p \mid N)}$$

increases exponentially for huge sets of p

- All events must be *exhaustive*. This signifies that to work out all probabilities the set of possible events must be closed.

Caution If new information occurs the set must be formed afresh and *all* probabilities recalculated.

So Simple Bayes rule-based systems are not appropriate for uncertain reasoning.

- Knowledge acquisition is very rigid.
- Too many probabilities required — too large a storage space.
- Calculation time is too large.
- Updating new information is hard and time consuming.
- Exceptions such as "none of the above" cannot be represented.
- Humans are not very good probability estimators.

However, Bayesian statistics still offer the core to reasoning in most of the uncertain reasoning systems with appropriate enhancement to conquer the above problems.

Notes

We will gaze at wide categories:

- Certainty factors,
- Bayesian networks.

Task Discuss the problems occurring in bayes theorem.

Self Assessment

Fill in the blanks:

1. ..... methods give a method for showing principles that are not certain (or uncertain) but for which there may be some assisting (or contradictory) confirmation.
2. ..... are (real) numbers in the range 0 to 1.
3. ..... probability, $P(A \mid B)$, signifies the probability of event A specified that we know event B has appeared.
4. The set of all ..... must be mutually exclusive and comprehensive.
5. P(.....) specifies the probability of A specified only B's evidence.
6. Probability = (number of desired outcomes) / (.....)

8.2 Certainty Factors and Rule Based Systems

This strategy has been recommended by Shortliffe and Buchanan and utilized in their famous medical diagnosis MYCIN system.

MYCIN is fundamentally and *expert system*. Here we only focus on the probabilistic reasoning aspects of MYCIN.

- MYCIN signifies knowledge as a set of rules.
- Related with each rule is a *certainty factor*
- A certainty factor depends on measures of belief B and disbelief D of an hypothesis H_i given evidence E as below:

$$B(H_i \mid E) = \begin{cases} \frac{1}{\max[P(H_i|E), P(H_i)]} & \text{if } P(H_i) = 1 \\ \text{otherwise} \\ \frac{(1 - P(H_i))P(H_i \mid E)}{} \end{cases}$$

$$D(H_i \mid E) = \begin{cases} \frac{1}{P(H_i) - \min[P(H_i|E), P(H_i)]} & \text{if } P(H_i) = 0 \\ \text{otherwise} \\ \frac{P(H_i))P(H_i \mid E)}{} \end{cases}$$

where $P(H_i)$ is the standard probability.

- The certainty factor C of some hypothesis H_i given evidence E is defined as:

$$C(H_i \mid E) = B(H_i \mid E) - D(H_i \mid E)$$

Notes

8.2.1 Reasoning with Certainty Factors

- Rules articulated as if *evidence list* E_1, E_2 then there is indicative evidence with probability, *p* for *symptom* H_i.
- MYCIN accesses rules to reason backward to clinical data evidence from its aim of forecasting a disease-causing organism.
- Certainty factors primarily supplied by experts changed as per the previous formulae.
- How do we perform reasoning when numerous rules are *chained* together?

  Measures of belief and disbelief given numerous observations are computed as follows:

$$B(H_i \mid E_1, E_2) = \begin{cases} 0 & \text{if } D(H_i \mid E_2) = 1 \text{ otherwise} \\ B(H_i|E_1) + B(H_i|E_2)(1 - B(H_i|E_1)) & \end{cases}$$

$$D(H_i \mid E) = \begin{cases} 0 & \text{if } B(H_1 \mid E_2) = 1 \text{ otherwise} \\ D(H_i|E_1) + D(H_i|E_2) + D(H_i|E_1)) & \end{cases}$$

- How about our belief regarding several hypotheses taken jointly? Measures of belief given numerous hypotheses and to be shared logically are computed as follows:

$$B(H_1 \wedge H_2 \mid E) = \min(B(H_1 \mid E)_1 B(H_1 \mid E)$$
$$B(H_1 \vee H_2 \mid E) = \max(B(H_1 \mid E)_1 B(H_1 \mid E)$$

  Disbelief is computed similarly.

8.2.2 Overcoming the Bayes Rule Shortcomings

Certainty Factors do hold on to the rules of Bayesian statistics, but it can symbolize tractable knowledge systems:

- Individual rules give belief in an hypotheses — fundamentally a conditional probability.
- The formulae for amalgamation of evidence/hypotheses fundamentally assume that all rules are sovereign ruling out the requirement for joint probabilities.
- The load of assuring independence is positioned on the rule writer.

Self Assessment

Fill in the blanks:

7. MYCIN signifies knowledge as a set of ..... .
8. MYCIN accesses rules to reason backward to clinical data evidence from its aim of forecasting a ..... causing organism.
9. The load of assuring ..... is positioned on the rule writer.

8.3 Bayesian Networks

These are also known as *Belief Networks* or *Probabilistic Inference Networks* primarily generated by Pearl (1988).

The fundamental idea is:

- Knowledge in the world is *modular* — many events are conditionally sovereign of most other events.

Notes

- Adopt a model that can utilize a more local depiction to permit interactions among events that *only* affect each other.
- Some events may only be *unidirectional* others may be *bidirectional* — make a difference between these in model.
- Events may be fundamental and so get chained jointly in a network.

8.3.1 Implementation

A Bayesian Network is define as a directed acyclic graph:

- A graph where the directions are links which specify dependencies that appears between nodes.
- Nodes symbolize propositions regarding events or events themselves.
- Conditional probabilities measure the power of dependencies.

Example: Let us see the following example:

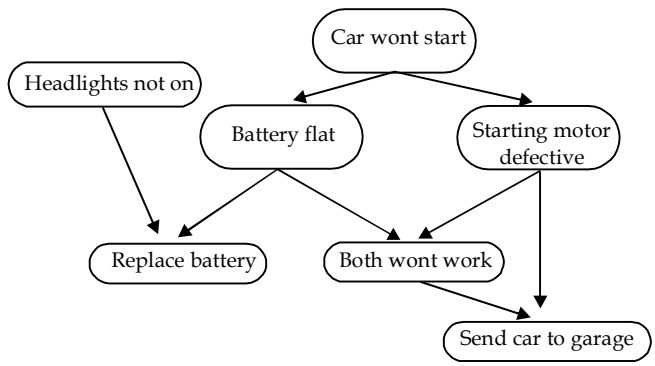
- The probability, $P(\delta_1)$ that my car won't begin.
- If my car won't begin then it is possible that
  - The battery is flat or
  - The staring motor is broken.

To decide whether to fix the car myself or send it to the garage the following decision is made:

- If the headlights do not function then the battery is apt to be flat so i fix it myself.
- If the beginning motor is faulty then send car to garage.
- If battery and beginning motor both gone send car to garage.

The network to symbolize this is as follows:

Figure 8.1: A Simple Bayesian Network



8.3.2 Reasoning in Bayesian Nets

- Probabilities in links follow typical conditional probability axioms.
- Thus follow links in attaining hypothesis and update beliefs consequently.

Notes

- A few broad classes of algorithms have bee used to assist with this:
  - Pearls's message passing method.
  - Clique triangulation.
  - Stochastic methods.
  - Basically they all take advantage of *clusters* in the network and use their limits on the influence to constrain the search through net.
  - They also ensure that probabilities are updated correctly.

Notes As information is *local* information can be willingly added and removed with minimum result on the entire network. **ONLY** affected nodes require updating.

Example: Here we portray a practical example from research based here in Cardiff.

We have utilized Bayesian Nets in a Computer Vision application. Here we try to portray the Bayesian reasoning behind the process.

The *objective* is to execute a task known as *data fusion* to attain a *segmentation* — an explanation of an object (observed from a set of images) detailing its surface properties. In the instance given here we contract with a simple cube. So the final explanation will hopefully list its edges and its faces and how they are associated together.

The input to the *fusion process* is three *preprocessing stages* that have removed out edge information and planar surface information from 2D grey scale (monochrome) images and 3D range data.

So from these three pre-processes we have a record of all lines, curved or straight, a list of all line intersections (two or three line intersections) and a record of all the surface equations extracted from both image types. We can now construct the network from these lists of features. As discussed above, we hypothesize regarding extracted surfaces intersecting. For us to assess these hypotheses we require to have evidence to sustain or contradict them. The evidence that we use is :

- Straight lines extracted from light image.
- Curves extracted from light image.
- 'Areas of uncertainty' extracted from depth map.

The two lines lists are produced as discussed above. The areas of uncertainty are found when we are trying to locate the surface equations of every surface type. Errors are set up in the depth map where the mask to locate the common surface shape overlaps two or more surfaces, the error tends to be enlarged thus, providing us a clue that a surface intersection appears in that general area. So we are using confirmation from more than one source of data.

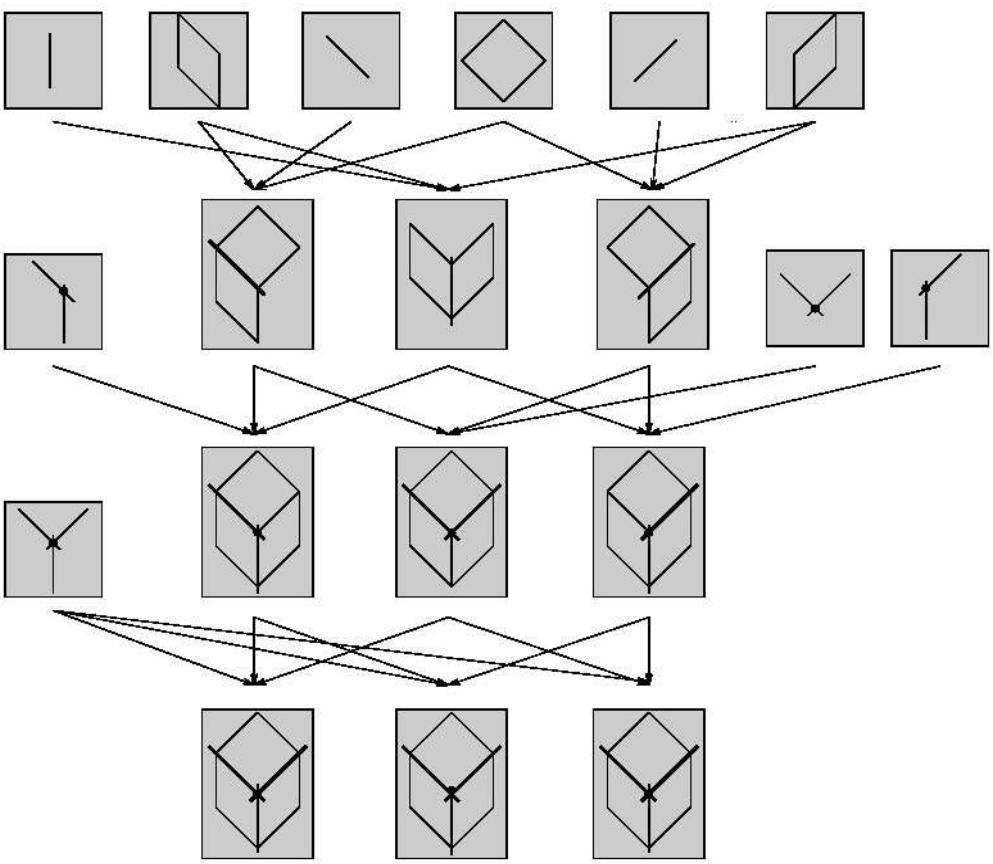
We continue by taking each of the surfaces in the surface list and a node is produced to represent it. We then take a pair of surfaces and try to intersect them. If they are perhaps intersecting then a 'feature group' node is produced referencing the surfaces and associated to the children surface nodes. This process is repetitive for each pair surfaces that we have removed. We now would like to connect a conditional probability to each of our new nodes. So we now know the surfaces that could *possibly* interact in the object. We now connect a probability to these connections. We do this by locating the equation of intersection, this will be a three dimensional line for two

Notes

planes or an ellipse for a plane and a sphere, and project this onto our focal plane. Now we have our hypothesized intersections in the same dimension as our extracted lines from the beginning stage. So we now locate, for every intersecting line a closest match line from our line list. Once we have located the closest matching line we produce a probability from the error. So a line that closely matches our intersection line then we have a high probability while two surfaces that don't intersect in the object are unlikely to correspond with a line from the line list thus providing us a low probability. The line that is found is also verified to see if it appears in an area of uncertainty. If it does then that is another strong evidence that the line that we have found is really where surfaces are attached.

So once we have produced this network with all the essential links *etc.* any more information that is given to the system can be added and the network will broadcast this information throughout the network in the form of probability updating. So for example say a new image was offered from say a colour image and this image increased the possibility of some edges and corners being present in the image then this would increase the probability of those traits that are linked to those edges and corners which would propagate all through the network. Figure 8.2 displays us a simple example of the network that would be produced from the input data of edges and planar faces of the cube. As can be observed, the feature group nodes can symbolize groups that vary from single features like line segments, surfaces or corners or the whole object is represented in the lower nodes which involves three surfaces, three line segments, three crosses and one corner.

Figure 8.2: A Bayesian Network for Segmentation of a Cube



Notes

Self Assessment

Fill in the blanks:

10. ..... networks are also known as *Belief* Networks or Probabilistic Inference Networks primarily generated by Pearl (1988).
11. A Bayesian Network is define as a ..... acyclic graph.
12. Conditional probabilities measure the power of ..... .

8.4 Fuzzy Logic

Fuzzy logic is a completely diverse approach to symbolizing uncertainty:

- It concentrates on ambiguities in illustrating events rather than uncertainty regarding the incidence of an event.
- Modifies the definitions of set theory and logic to permit this.
- Traditional set theory defines set memberships as a boolean predicate.

8.4.1 Fuzzy Set Theory

- *Fuzzy* set theory defines set membership as a *possibility distribution*.

  The general rule for this can displayed as:

$$f : [0, 1]^n \rightarrow [0, 1]$$

  where *n* some number of possibilities.

Did u know? *Fuzzy* set theory mainly specifies that we can take *n* possible events and use *f* to produce as single possible outcome.

- Once set membership has been redefined we can build up *new logics* depending on combining of sets etc. and reason efficiently.

Self Assessment

Fill in the blanks:

13. ..... concentrates on ambiguities in illustrating events rather than uncertainty regarding the incidence of an event.
14. Fuzzy set theory defines set membership as a ..... .
15. Fuzzy set theory mainly specifies that we can take n possible events and use *f* to produce as single possible ..... .

8.5 Summary

- Statistical methods give a method for showing principles that are not certain (or uncertain) but for which there may be some assisting (or contradictory) confirmation.

Notes

- Conditional probability, $P(A|B)$, signifies the probability of event A specified that we know event B has appeared.
- Bayes Theorem, that is $P(H_i|E) = \frac{P(E|H_i)P(H_i)}{\sum_{k=1}^{n} P(E|H_k)P(H_k)}$ signifies that specified some evidence E then probability that hypothesis is true is equal to the proportion of the probability that E will be true specified H_i times the a priori evidence on the probability of H_i and the sum of the probability of E over the set of all hypotheses times the probability of these hypotheses.
- Bayesian statistics lounge at the heart of many statistical reasoning systems.
- Bayesian networks are also known as Belief Networks or Probabilistic Inference Networks primarily generated by Pearl (1988).
- Certainty Factors do hold on to the rules of Bayesian statistics, but it can symbolize tractable knowledge systems.
- Fuzzy logic concentrates on ambiguities in illustrating events rather than uncertainty regarding the incidence of an event.
- Fuzzy set theory defines set membership as a possibility distribution.

8.6 Keywords

Bayesian networks: Bayesian networks are also known as Belief Networks or Probabilistic Inference Networks primarily generated by Pearl (1988).

Fuzzy Logic: Fuzzy logic concentrates on ambiguities in illustrating events rather than uncertainty regarding the incidence of an event.

***Fuzzy* Set:** Fuzzy set theory defines set membership as a possibility distribution.

Probabilities: Probabilities are (real) numbers in the range 0 to 1.

8.7 Review Questions

1. What is Bayes Theorem? Also illustrate Bayes rule-based systems.
2. Enlighten the probabilistic reasoning aspects of MYCIN.
3. Given that the first card dealt from a pack of cards was ace what is the probability that the next card will be an ace?
4. What are Bayesian networks? Illustrate the concept of implementation with example.
5. Given that
   (a) the probability that it will rain in Cardiff tomorrow is 0.8
   (b) the probability that there are Sea Gulls on Roath Park lake given that it will rain tomorrow is 0.1
   (c) the probability that there are Sea Gulls on Roath Park lake given that it will not rain tomorrow is 0.05

   What is the probability that it will rain tomorrow provided that there a Sea Gulls on Roath Park lake?

Notes

6. Initially, I gaze out of the window and saw at the clouds on the horizon. I conclude that I consider there is a 0.6 chance of rain. Half an hour later I make the same observation and now consider there is a 0.4 chance of rain.

   (a) What are my primary disbelief and certainty factors that it will rain?

   (b) What are the new estimates of belief, disbelief and certainty factors based on both observations?

   I then catch a climate forecast states that there 50am. What is my belief, disbelief and certainty factors depending on my observations and the weather forecast?

7. Peter, Paul and Mary are 3 suspects in a murder case. Only one of them committed the crime. There is some evidence that provides equal support for the murderer being male or female. There is also evidence that Peter was has an alibi for the time of the killing.

   (a) Combine the information above and represent it in the Dempster-Shafer formalism.

   (b) Calculate the Belief and Plausibility for each suspect.

8. Consider the following facts:

   (a) I saw my cat in the living room 3 hours ago.

   (b) Two hours ago my door blew open.

   (c) Three quarters of the time my door blows open my cat runs outside through the door.

   (d) On hour ago I thought I heard a cat-noise in my living. Assume I was half certain.

   (e) In any one hour period the probability that the cat will leave the room is 0.2. There is also a 0.2 probability that he may enter the room.

   What is the Certainty that the cat is in my living room? Use Bayesian networks to answer this.

9. Design a Bayesian network for each of the following problems:

   (a) Choosing a menu for a dinner party.

   (b) Planning a holiday.

   (c) Selecting final year computer science options based on second and first year prerequisites.

10. Describe how Fuzzy Logic is used in representing uncertainty.

Answers: Self Assessment

1.	Statistical	2.	Probabilities
3.	Conditional	4.	hypotheses
5.	$A \mid B$	6.	total number of outcomes
7.	rules	8.	disease
9.	independence	10.	Bayesian
11.	directed	12.	dependencies

13. Fuzzy Logic
14. possibility distribution
15. outcome

Notes

8.8 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

Notes

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.springer.com

Notes

Unit 9: Weak Slot and Filler Structures

CONTENTS

Objectives
Introduction
9.1 Semantic Nets
9.1.1 Representation in a Semantic Net
9.1.2 Inference in a Semantic Net
9.1.3 Extending Semantic Nets
9.2 Frames
9.2.1 Frame Knowledge Representation
9.2.2 Interpreting Frames
9.3 Summary
9.4 Keywords
9.5 Review Questions
9.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of weak slot and filler structures
- Discuss the concept of semantic nets
- Describe the frames

Introduction

Weak Slot and Filler Structures facilitates attribute values to be improved speedily where declarations are indexed by the entities and binary predicates are indexed by first argument. *E.g. team (Mike-Hall, Cardiff)*. Properties of relations are simple to illustrate. It permits ease of deliberation as it embraces features of object-oriented programming. It is so called for the reason that:

- A *slot* is an attribute value pair in its simplest outline.
- A *filler* is a value that a slot can take — could be a numeric, string (or any data type) value or a pointer to a different slot.
- A *weak* slot and filler structure does not consider the *content* of the representation.

We will consider here two types:

- Semantic Nets.
- Frames.

Notes

9.1 Semantic Nets

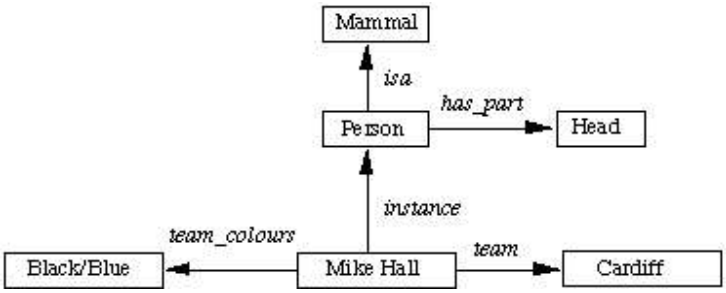
The main thought is that:

- The meaning of a notion occurs from its relationship to other notions, and that,
- The information is accumulated by interconnecting nodes with labeled arcs.

9.1.1 Representation in a Semantic Net

The physical attributes of a person can be displayed as in Figure 9.1.

Figure 9.1: A Semantic Network



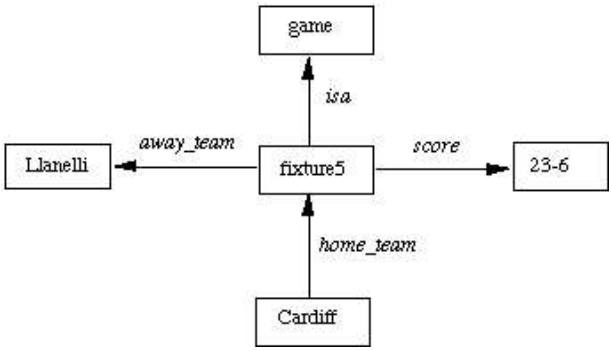
These values can also be displayed in logic as: *isa(person, mammal), instance(Mike-Hall, person) team(Mike-Hall, Cardiff)*

Remember that *isa* and *instance* symbolize inheritance and are well-liked in many knowledge representation systems. But we have a difficulty: *How we can have more than 2 place predicates in semantic nets?*

Example: score(Cardiff, Llanelli, 23-6) Solution:

- Produce new nodes to symbolize new objects either contained or alluded to in the knowledge, *game* and *fixture* in the existing example.
- Relate information to nodes and fill up slots (Figure 9.2).

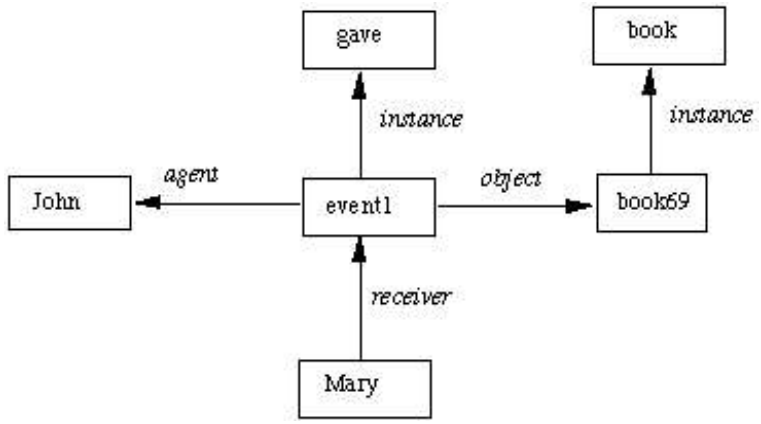
Figure 9.2: A Semantic Network for *n*-Place Predicate



Notes

As a more multifaceted instance consider the sentence: *John gave Mary the book*. Here we have numerous facets of an event.

Figure 9.3: A Semantic Network for a Sentence



Task Discuss the use of semantic net with example.

9.1.2 Inference in a Semantic Net

Fundamental inference method: *follow links between nodes*.

Two methods to perform this:

- Intersection search
  - The view that *spreading activation* out of two nodes and locating their intersection locates relationships between objects. This is attained by allocating a particular tag to every visited node.
  - Many benefits including entity-dependent organization and fast parallel implementation. However very structured questions necessitate highly structured networks.
- Inheritance
  - The *isa* and *instance* representation offer a method to implement this.

Did u know? Inheritance also offers a means of dealing with *default reasoning*.

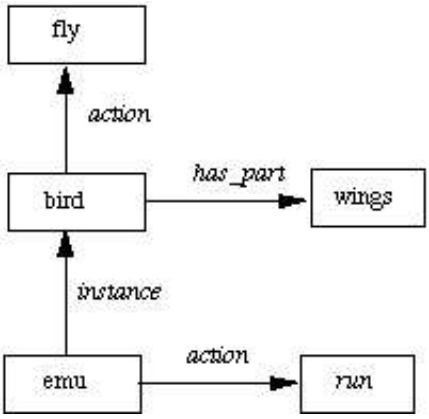
Example: We could represent:

- Emus are birds.
- Usually birds fly and have wings.
- Emus run.

Notes

in the following Semantic net:

Figure 9.4: A Semantic Network for a Default Reasoning



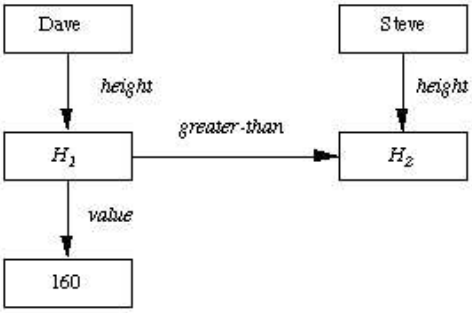
In creating certain inferences we will also necessitate to *differentiate between the link that defines a new entity and holds its value and the other type of link that associates two current entities*. Consider the example displayed where the height of two people is portrayed and we also desire to contrast them.

We require extra nodes for the notion in addition to its value.

Figure 9.5: Two Heights

Special procedures are required to process these nodes, but without this dissimilarity the analysis would be very restricted.

Figure 9.6: Comparison of Two Heights



9.1.3 Extending Semantic Nets

Here we will take into account some extensions to Semantic nets that conquer some problems or enlarge their appearance of knowledge.

Notes

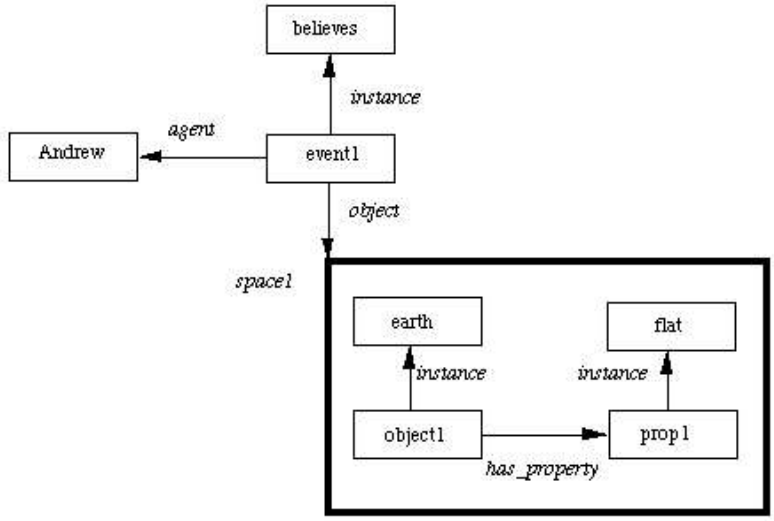
Partitioned Networks *Partitioned* Semantic Networks permit for:

- Propositions to be made without assurance to truth.
- Expressions to be measured.

Did u know? Basic idea: *Break network into* ***spaces*** *which comprise groups of nodes and arcs and consider each* ***space*** *as a node.*

Consider the following: *Symon thinks that the earth is flat.* We can predetermine the proposition *the earth is flat* in a *space* and inside it have nodes and arcs to symbolize the fact (Figure 9.7). We can the have nodes and arcs to relate this *space* the rest of the network to signify Symon's belief.

Figure 9.7: Partitioned Network



Now consider the measured expression: *Every parent loves their child* To symbolize this we:

- Produce a *general statement*, GS, special class.
- Make node g an instance of GS.
- Every element contains at least 2 attributes:
  - a *form* that specifies which relation is being emphasized.
  - one or more *forall* ($\forall$) or *exists* ($\exists$) connections — these symbolize universally quantifiable variables in such statements *e.g. x, y* in $\forall$x: *parent(x)* $\rightarrow \exists$*y:child(y)* $\wedge$ *loves(x,y)*

Now we have to create two *spaces one* for each *x,y*.

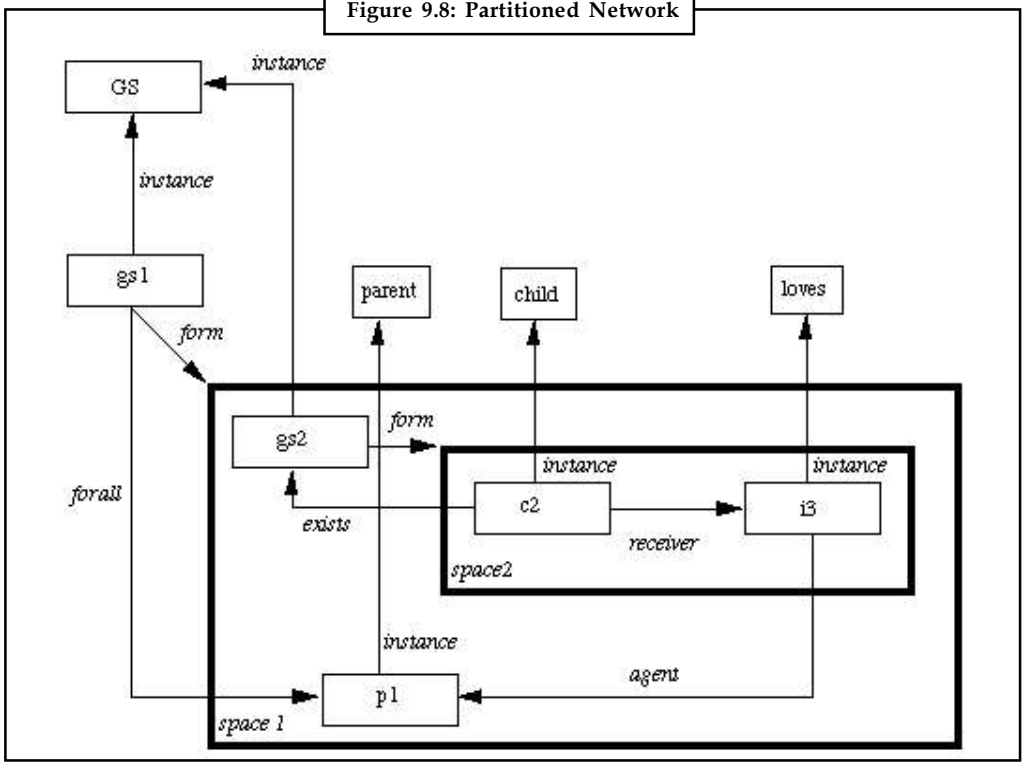
Notes We can articulate variables as *existentially qualified* variables and articulate the event of *love* having an agent *p* and receiver *b* for each parent *p* which could abridge the network.

Also If we modify the sentence to *Every parent loves* <u>*child*</u> then the node of the object being acted on (*the child*) occurs outside the form of the common statement. Thus it is not observed as an

Notes

existentially measured variable whose value may rely on the agent. So we could create a partitioned network as in Figure 9.8.

Figure 9.8: Partitioned Network



Self Assessment

Fill in the blanks:

1. A ..... is an attribute value pair in its simplest outline.
2. A ..... is a value that a slot can take — could be a numeric, string (or any data type) value or a pointer to a different slot.
3. A *weak* slot and filler structure does not consider the ..... of the representation.
4. In ..... method, the view that *spreading activation* out of two nodes and locating their intersection locates relationships between objects.
5. ..... Semantic Networks permit for propositions to be made without assurance to truth.
6. ..... Structures facilitates attribute values to be improved speedily where declarations are indexed by the entities and binary predicates are indexed by first argument.

9.2 Frames

Frames can also be considered as an expansion to Semantic nets. Certainly it is not apparent where the difference among a semantic net and a frame ends. Semantic nets initially we used to symbolize labeled connections among objects. As tasks became more multifaceted the representation requires to be more structured. A *frame* is a compilation of attributes or slots and related values that portray some actual world entity. Frames on their own are not mainly useful

Notes

but frame systems are a powerful manner of encoding information to sustain reasoning. Set theory offers a good foundation for understanding frame systems. Every frame symbolizes:

- a class (set), or
- an instance (an element of a class).

Caution The more structured the system it turns out to be more useful to use frames.

Task Illustrate the use of *Partitioned* Semantic Networks.

9.2.1 Frame Knowledge Representation

Take the example first discussed in Semantics Nets.

Figure 9.9: A Simple Frame System

```
Person
                    isa:                        Mammal
                    Cardinality:                …
Adult-Male
                    isa:                Person
                          Cardinality:  …
Rugby-Player
                    isa:                Adult-Male
                    Cardinality:        …
                    Height:
                          Weight:
                    Position:
                    Team:
                          Team-Colours:
Back
                    isa:                Rugby-Player
                    Cardinality:
                    Tries:
Mike-Hall
                    instance:                   Back
                    Height:                     6-0
                    Position:                   Centre
                    Team:               Cardiff-RFC
                    Team-Colours:               Black/Blue
```

Contd...

Notes

```
Rugby-Team
                    isa:                Team
                    Cardinality:                …
                    Team-size:                  15
                    Coach:
Cardiff-RFC
                    instance:           Rugby-Team
                    Team-size:          15
                    Coach: Terry Holmes
                    Players:            {Robert-Howley, Gwyn-Jones, ...}
```

Here the frames *Person, Adult-Male, Rugby-Player and Rugby-Team* are all **classes** and the frames *Robert-Howley* and *Cardiff-RFC* are instances.

Notes

- The *isa* relation is actually the subset relation.
- The *isa* attribute possesses a transitivity property. This shows: *Robert-Howley* is a *Back* and a *Back* is a *Rugby-Player* who in turn is an *Adult-Male* and also a *Person*.
- Both *isa* and *instance* have inverses which are known as subclasses or all instances.
- There are attributes that are connected with the class or set such as cardinality and on the other hand there are attributes that are owned by every member of the class or set.

Distinction between Sets and Instances

It is significant that this difference is evidently understood.

Cardiff-RFC can be considered of as a set of players or as an instance of a *Rugby-Team*.

If *Cardiff-RFC* were a *class* then

- its instances would be players
- it could not be a subclass of *Rugby-Team* or else its elements would be members of Rugby-Team which we do not desire.

Rather we make it a subclass of *Rugby-Player* and this permits the players to inherit the accurate properties facilitating us to allow the *Cardiff-RFC* to inherit information regarding teams.

This means that *Cardiff-RFC* is an instance of *Rugby-Team*.

But there is a difficulty here:

- A class is a set and its elements contain properties.
- We wish to use inheritance to confer values on its members.
- But there are properties that the set or class itself has like the manager of a team.

Notes

This is why we require to observe *Cardiff-RFC* as a subset of one class players and an instance of teams.

A metaclass is a particular class whose elements are themselves classes.

Now consider our rugby teams as:

Figure 9.10: A Metaclass Frame System

Class	
instance:	*Class*
isa:	*Class*
Cardinality:	...
Team	
instance:	*Class*
isa:	*Class*
Cardinality:	{ The number of teams }
Team-Size:	*15*
Rugby-Team	
isa:	*Team*
Cardinality:	{ The number of teams }
Team-size: 15	
Coach:	
Cardiff-RFC	
instance:	*Rugby-Team*
Team-size:	*15*
Coach:	*Terry Holmes*
Robert-Howley	
instance:	*Back*
Height:	*6-0*
Position:	*Scrum Half*
Team:	*Cardiff-RFC*
Team-Colours:	*Black/Blue*

The fundamental metaclass is *Class*, and this permits us to

- define classes which are instances of other classes, and (thus)
- inherit properties from this class.

Inheritance of default values appears when one element or class is an instance of a class.

Slots as Objects

How can we to symbolize the following properties in frames?

- Attributes like *weight, age* be attached and make sense.
- Constraints on values like *age* being less than a hundred
- Default values

Notes

- Rules for inheritance of values like children inheriting parent's names
- Rules for computing values
- Many values for a slot.

A slot is a relation that maps from its domain of classes to its range of values.

A relation is a set of ordered pairs so one relation is a subset of another.

As slot is a set the set of all slots can be symbolize by a metaclass known as *Slot,* say.

Consider the following:

```
SLOT
                isa:          Class
                instance:            Class
                domain:
                range:
                range-constraint:
                definition:
                default:
                to-compute:
                single-valued:
Coach
                instance:            SLOT
                domain:              Rugby-Team
                range:               Person
                range-constraint:         λz   (experience  x.manager)
                default:
                single-valued:              TRUE
Colour
                instance:            SLOT
                domain:              Physical-Object
                range:               Colour-Set
                single-valued:              FALSE
Team-Colours
                instance:            SLOT
                isa:          Colour
                domain:              team-player
                range:               Colour-Set
                range-constraint:           not  Pink
                single-valued:              FALSE
```

Notes

```
Position
                instance:            SLOT
                domain:              Rugby-Player
                range:               { Back, Forward, Reserve }
                to-compute:           λz x.position
                single-valued:              TRUE
```

The following points are to be noticed.:

- Instances of *SLOT* are slots
- Related with *SLOT* are attributes that every instance will inherit.
- Every slot has a domain and range.
- Range is divided into two parts one the class of the elements and the other is a restraint which is a logical expression if absent it is taken to be true.
- The *to-compute* attribute includes a procedure to calculate its value. *E.g.* in *Position* where we make use of the dot notation to allocate values to the slot of a frame.
- Transfers through lists other slots from which values can be derived from inheritance.

Caution If there is a value for default then it must be passed on unless an instance has its own value

9.2.2 Interpreting Frames

A frame system interpreter must be competent of the following so as to exploit the frame slot demonstration:

- Steadiness checking – when a slot value is added to the frame depending on the domain attribute and that the value is legal using range and range constraints.
- Broadcast of *definition* values along *isa* and *instance* links.
- Inheritance of default values along *isa* and *instance* links.
- Calculation of value of slot as required.
- Verifying that only correct number of values calculated.

Self Assessment

Fill in the blanks:

7. Frames can also be considered as an expansion to ..... .
8. A ..... is a compilation of attributes or slots and related values that portray some actual world entity.
9. Frames on their own are not mainly useful but frame systems are a powerful manner of encoding ..... to sustain reasoning.
10. The more ..... the system it turns out to be more useful to use frames.
11. A ..... is a set and its elements contain properties.

Notes

12. A ..... is a particular class whose elements are themselves classes.
13. A ..... is a set of ordered pairs so one relation is a subset of another.
14. ..... of default values appears when one element or class is an instance of a class.
15. Every slot has a domain and ..... .

9.3 Summary

- Weak Slot and Filler Structures facilitates attribute values to be improved speedily where declarations are indexed by the entities and binary predicates are indexed by first argument.
- Weak Slot and Filler Structure permits ease of deliberation as it embraces features of object oriented programming.
- In semantic nets, The information is accumulated by interconnecting nodes with labeled arcs.
- The view that spreading activation out of two nodes and locating their intersection locates relationships between objects.
- *Partitioned* Semantic Networks permit for propositions to be made without assurance to truth and expressions to be measured.
- Basic idea of extended semantic net is to break network into spaces which comprise groups of nodes and arcs and consider each space as a node.
- A frame is a compilation of attributes or slots and related values that portray some actual world entity.
- Frames on their own are not mainly useful but frame systems are a powerful manner of encoding information to sustain reasoning.

9.4 Keywords

Filler: A filler is a value that a slot can take — could be a numeric, string (or any data type) value or a pointer to a different slot.

Frame: A frame is a compilation of attributes or slots and related values that portray some actual world entity.

Slot: A slot is an attribute value pair in its simplest outline.

9.5 Review Questions

1. Construct Semantic Net representations of the following:
   (a) Dave is Welsh, Dave is a Lecturer.
   (b) Paul lent his new Frank Zappa CD to his best friend.
2. Represent the following in partitioned semantic networks:
   (a) Every player kicked a ball.
   (b) All players like the referee.
   (c) Andrew believes that there is a fish with lungs.

Notes

3. Pick a problem area and represent the knowledge in frame based system.
4. Devise algorithms that enable reasoning with frames. Discuss how:
   (a) Inference through inheritance can be achieved.
   (b) Matching can be achieved.
5. What are the benefits of a frame based knowledge representation?
6. What problems do you predict that a frame based knowledge representation having? Give examples of knowledge hard to symbolize in a frame. How could some difficulties be conquer?
7. What programming languages would be matched to put into practice a semantic network and frames?
8. Weak Slot and Filler Structures permits ease of deliberation as it embraces features of object oriented programming. Comment.
9. Illustrate the inference methods used in semantic nets.
10. Make distinction between sets and instances with examples.

Answers: Self Assessment

1.	Slot	2.	Filler
3.	Content	4.	Intersection Search
5.	Partitioned	6.	Weak Slot and Filler
7.	Semantic Nets	8.	Frame
9.	Information	10.	Structured
11.	Class	12.	Metaclass
13.	Relation	14.	Inheritance
15.	Range		

9.6 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Notes

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS: *A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link www.compmathsjournal.com/fulltext/v2i3/2.3.6.pdf

Notes

Unit 10: Strong Slot and Filler Structures

CONTENTS

Objectives
Introduction
10.1 Conceptual Dependency (CD)
10.1.1 Benefits of CD
10.1.2 Limitations of CD
10.1.3 Applications of CD
10.2 Scripts
10.2.1 Advantages of Scripts
10.2.2 Disadvantages
10.3 Summary
10.4 Keywords
10.5 Review Questions
10.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concepts of strong slot and filler structures
- Discuss the Conceptual Dependency (CD)
- Illustrate the use of scripts

Introduction

Strong Slot and Filler Structures usually symbolize links among objects according to more *firm* rules, particular notions of what types of object and associations among them are given, and symbolize knowledge regarding common situations. In this unit, you will understand the concept of conceptual dependency and scripts.

10.1 Conceptual Dependency (CD)

Conceptual Dependency initially generated to symbolize knowledge attained from natural language input.

Conceptual Dependency demonstration is utilized in natural language processing so as to represent them earning of the sentences in such a manner that inference we can be made from the sentences. It is autonomous of the language in which the sentences were initially stated.

Did u know? CD representations of a sentence is constructed out of primitives, which are not words belonging to the language but are abstract, these primitives are united to form the meanings of the words.

Notes

The objectives of this theory are:

- To aid in the sketching of inference from sentences.
- To be autonomous of the words utilized in the original input.

Caution For any 2 (or more) sentences that are equal in meaning there should be only one depiction of that meaning.

It has been utilized by many programs that indicate to recognize English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.*.

CD offers:

- a structure into which nodes displaying information can be positioned
- a particular set of primitives
- at a specified level of granularity.

Sentences are displayed as a series of diagrams portraying actions by means of both abstract and real physical situations.

- The agent and the objects are displayed
- The actions are constructed from a set of primitive acts which can be customized by tense.

Notes Conceptual dependency theory was dependent on two suppositions:

1. If two sentences have the similar meaning, they should be displayed the same, irrespective of the specific words used.
2. Information unreservedly specified in the sentence should be displayed overtly. That is, any information which can be inferred from what is overtly specified should be involved in the representation.

Example: Primitive Acts are:

ATRANS

— Transfer of an intangible relationship. *E.g. give*.

PTRANS

— Transfer of the physical position of an object. *E.g. go*.

PROPEL

— Application of a physical force to an object. *E.g. push*.

MTRANS

— Transfer of mental information. *E.g. tell*.

MBUILD

— Build new information from old. *E.g. decide*.

Notes

SPEAK

— Make a sound. *E.g. say*.

ATTEND

— Concentrate a sense on a stimulus. *E.g. listen, watch*.

MOVE

— Movement of a body part by owner. *E.g. punch, kick*.

GRASP

— Actor grasping an object. *E.g. clutch*.

INGEST

— Actor ingesting an object. *E.g. eat*.

EXPEL

— Actor getting clear of an object from body.

Six ancient conceptual categories offer *building blocks* which are the set of permissible dependencies in the concepts in a sentence:

PP

— Real world objects.

ACT

— Real world actions.

PA

— Attributes of objects.

AA

— Attributes of actions.

T

— Times.

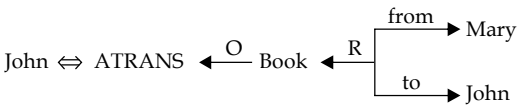
LOC

— Locations.

How do we bond these things together?

Example: Take the example as below:

John provides Mary a book



- Arrows specify the direction of dependency. Letters above specify some relationships:

  o

  — object.

Notes

R

– recipient-donor.

I

– instrument *e.g. eat with a spoon*.

D

– destination *e.g. going home*.

- Double arrows (⇔) designate *two-way* links among the actor (PP) and action (ACT).
- The actions are constructed from the set of primitive acts.
  - These can be customized by *tense etc.*

The use of tense and mood in illustrating events is tremendously significant and schank introduced the following modifiers:

p

– past

f

– future

t

– transition

t_3

– start transition

t_f

– finished transition

k

– continuing

?

– interrogative

/

– negative

delta

– timeless

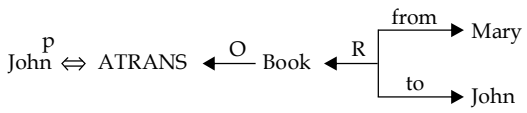
c

– conditional

the absence of any modifier points to the *present tense*.

So the *past tense* of the above example:

John gave Mary a book becomes:



Notes

The ⇔ has an object (actor), PP and action, ACT. *I.e.* PP ⇔ ACT. The triple arrow (⇔) is also a two link but among an object, PP, and its attribute, PA. *i.e., PP* ⇔ PA.

It symbolizes *isa* type dependencies. *E.g.*

Dave ⇔ lecturer Dave is a lecturer.

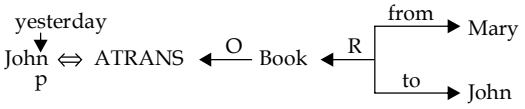
Did u know? Primitive states are used to depict many state descriptions like height, health, mental state, physical state.

There are many more physical states than primitive actions. They utilize a numeric scale.

Example: John height(+10) *John is the tallest* John height(< average) *John is short* Frank Zappa health(-10) *Frank Zappa is dead* Dave mental_state(-10) *Dave is sad* Vase physical_state(-10) *The vase is broken*

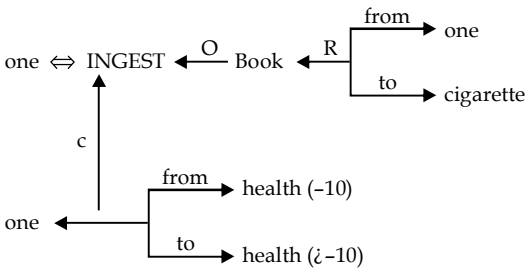
You can also identify things like the time of occurrence in the relationship.

Example: John gave Mary the book yesterday



Example: Now let us consider a more intricate sentence: *As smoking can kill you, I stopped* Let us have a look at how we symbolize the inference that *smoking can kill*:

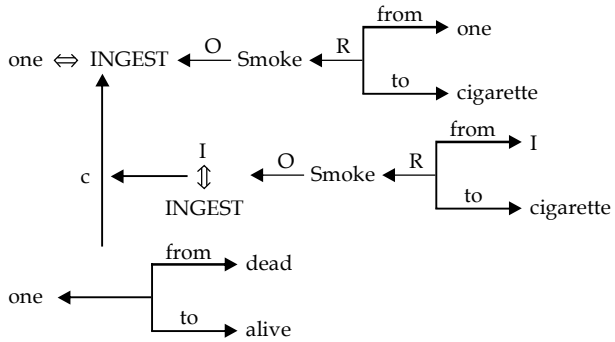
- Use the view of *one* to apply the knowledge to.
- Use the *primitive act* of INGESTing smoke from a cigarette to one.
- Killing is a changeover from being alive to dead. We use *triple arrow*s to designate a transition from one state to another.
- Have a conditional, *c* causality link. The *triple arrow* represents dependency of one concept on another.



To add the fact that *I stopped smoking*

- Use alike rules to imply that I smoke cigarettes.
- The qualification linked to this dependency represents that the instance Ingesting smoke has stopped.

Notes



Task Construct CD representation of the following: Drinking cocktail makes you drunk.

10.1.1 Benefits of CD

- Using these primitives includes fewer inference rules.
- Many inference rules are already displayed in CD structure.
- The holes in the initial structure aid to concentrate on the points still to be recognized.

10.1.2 Limitations of CD

- Knowledge must be converted into quite low level primitives.
- Impossible or hard to locate correct set of primitives.
- A lot of inference may still be needed.
- Representations can be multifaceted even for comparatively simple actions.

Consider

Dave bet Frank five pounds that Wales would win the Rugby World Cup.

Complex representations require a lot of storage.

10.1.3 Applications of CD

MARGIE (*Meaning Analysis, Response Generation and Inference on English*) — model natural language understanding.

SAM (*Script Applier Mechanism*) — Scripts to recognize stories.

PAM (*Plan Applier Mechanism*) — Scripts to recognize stories.

Schank *et al.* developed all of the above.

Self Assessment

Fill in the blanks:

1. ..... initially generated to symbolize knowledge attained from natural language input.

Notes

2. CD representations of a sentence is constructed out of ....., which are not words belonging to the language but are abstract , these primitives are united to form the meanings of the words.
3. For any 2 (or more) sentences that are equal in meaning there should be only ..... depiction of that meaning.
4. Sentences are displayed as a series of ..... portraying actions by means of both abstract and real physical situations.
5. Primitive Act ..... defines movement of a body part by owner.
6. Primitive Act ..... defines Application of a physical force to an object.
7. MARGIE (Meaning Analysis, Response Generation and Inference on English) is used to model ..... .

10.2 Scripts

A *script* is a structure that recommends a set of conditions which could be projected to follow on from one another.

It is alike to a thought sequence or a sequence of situations which could be anticipated.

It could be measured to comprise of a number of slots or frames but with more specialized roles.

Scripts are useful because:

- Events tend to happen in known runs or patterns.
- Causal relationships among events exist.
- Entry conditions exist which permit an event to happen.
- Prerequisites occur upon events taking place. *E.g.* when a student develops through a degree scheme or when a buyer buys a house.

The components of a script comprise:

Entry Conditions

— These must be pleased before events in the script can take place.

Results

— Conditions that will be true after events in script take place.

Props

— Slots displaying objects included in events.

Roles

— Persons involved in the events.

Track

— Dissimilarities on the script. Different tracks may share components of the same script.

Scenes

— The sequence of *events* that take place. *Events* are displayed in *conceptual dependency* form.

Notes

Scripts are functional in illustrating certain situations like robbing a bank. This might engage:

- Getting a gun.
- Hold up a bank.
- Escape with the money.

Here the *Props* might be

- Gun, *G*.
- Loot, *L*.
- Bag, *B*
- Get away car, *C*.

Figure 10.1: Simplified Bank Robbing Script

Script: ROBBERY	*Track: Successful Snatch*
Props: G = Gun, L = Loot, B= Bag, C = Get away car.	*Roles*: R = Robber, M = Cashier, O = Bank Manager, P = Policeman.
Entry Conditions: R is poor. R is destitute.	*Results*: R has more money. O is angry. M is in a state of shock. P is shot.
Scene 1: Getting a gun R PTRANS R into Gun Shop R MBUILD R choice of G R MTRANS choice. R ATRANS buys G (go to scene 2)	
Scene 2 Holding up the bank R PTRANS R into bank R ATTEND eyes M, O and P R MOVE R to M position R GRASP G R MOVE G to point to M R MTRANS "Give me the money or ELSE" to M P MTRANS "Hold it Hands Up" to R R PROPEL shoots G P INGEST bullet from G M ATRANS L to M M ATRANS L puts in bag, B M PTRANS exit O ATRANS raises the alarm (go to scene 3)	
Scene 3: The getaway M PTRANS C	

Notes

The *Roles* might be:

- Robber, *S*.
- Cashier, *M*.
- Bank Manager, *O*.
- Policeman, *P*.

The *Entry Conditions* might be:

- *S* is poor.
- *S* is destitute.

The *Results* might be:

- *S* has more money.
- *O* is angry.
- *M* is in a state of shock.
- *P* is shot.

There are 3 scenes: obtaining the gun, robbing the bank and the getaway.

The full Script could be illustrated in Figure 10.1.

Caution If a specific script is to be applied it must be triggered and the activating is based on its significance.

Notes Some more points to be noticed on Scripts:

1. If a topic is mentioned in passing then a pointer to that script could be held.
2. If the topic is significant then the script should be opened.
3. The danger lies in having too many lively scripts much as one might have too many windows open on the screen or too many recursive calls in a program.
4. Provided events follow a recognized trail we can use scripts to signify the actions included and use them to answer thorough questions.
5. Different trails may be permitted for dissimilar outcomes of Scripts (*e.g.* The bank robbery goes wrong).

Task Illustrate the functions of script.

10.2.1 Advantages of Scripts

- Aptitude to predict events.
- A single rational interpretation may be build up from a compilation of interpretations.

Notes

10.2.2 Disadvantages

- Less common than frames.
- May not be appropriate to symbolize all kinds of knowledge.

Self Assessment

Fill in the blanks:

8. A ..... is a structure that recommends a set of conditions which could be projected to follow on from one another.
9. Script is alike to a thought sequence or a sequence of ..... which could be anticipated.
10. Script could be measured to comprise of a number of ..... but with more specialized roles.
11. ..... component define the slots displaying objects included in events.
12. If a specific script is to be applied it must be triggered and the ..... is based on its significance.
13. Different trails may be permitted for dissimilar ..... of Scripts.
14. Scenes define the sequence of ..... that take place.
15. Different tracks may share components of the ..... script.

10.3 Summary

- Conceptual Dependency initially generated to symbolize knowledge attained from natural language input.
- For any 2 (or more) sentences that are equal in meaning there should be only one depiction of that meaning.
- CD offers a structure into which nodes displaying information can be positioned.
- Sentences are displayed as a series of diagrams portraying actions by means of both abstract and real physical situations.
- A script is a structure that recommends a set of conditions which could be projected to follow on from one another.
- It is alike to a thought sequence or a sequence of situations which could be anticipated.
- It could be measured to comprise of a number of slots or frames but with more specialized roles.
- If a specific script is to be applied it must be triggered and the activating is based on its significance.

10.4 Keywords

Conceptual Dependency: Conceptual Dependency initially generated to symbolize knowledge attained from natural language input.

Script: A script is a structure that recommends a set of conditions which could be projected to follow on from one another.

Notes

10.5 Review Questions

1. Illustrate the concept of conceptual dependency with example.
2. Explain the various objectives of conceptual dependency.
3. What are scripts? Illustrate the uses of scripts.
4. Discuss the advantages and disadvantages of conceptual dependency.
5. Construct CD representation of the following:
   (a) Johny request Mary for a pencil.
   (b) Jimmy stirred his coffee with a spoon.
   (c) David took the book off Jimmy.
   (d) On my way home, I stopped to fill my car with petrol.
   (e) I heard strange music in the woods.
   (f) Johny killed Mary by choking her.
6. Try capturing the differences between the following in CD:
   (a) Johny slapped David, Johny punched David.
   (b) Sue likes Prince, Sue adores Prince.
7. Rewrite the script specified in the unit so that the Bank robbery goes wrong.
8. Write a script to permit for both outcome of the Bank robbery: Getaway and going wrong and getting caught.
9. Write a script for enrolling as a student.
10. Find out how MARGIE, SAM and PAM are executed.

Answers: Self Assessment

1.	Conceptual Dependency	2.	Primitives
3.	One	4.	Diagrams
5.	MOVE	6.	PROPEL
7.	Natural Language Understanding	8.	Script
9.	Situations	10.	Slots or Frames
11.	Props	12.	Activating
13.	Outcomes	14.	Events
15.	Same		

10.6 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Notes

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.compmathsjournal.com/fulltext/v2i3/2.3.6.pdf

Notes

Unit 11: Natural Language Processing

CONTENTS

Objectives
Introduction
11.1 Natural Language Processing – Overview
11.1.1 Evaluation in NLP
11.1.2 Tasks and Limitations of NLP
11.1.3 Sub-problems of NLP
11.2 Steps in Natural Language Processing
11.2.1 Morphological Analysis
11.2.2 Syntactic Processing
11.2.3 Semantic Analysis
11.2.4 Discourse and Pragmatic Processing
11.3 Spell Checking
11.4 Summary
11.5 Keywords
11.6 Review Questions
11.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of natural language processing
- Identify the steps in natural language processing
- Discuss the spell checking

Introduction

Natural language processing is a field of computer science concerned with the interactions between computers and human (natural) languages. Natural language generation systems convert information from computer databases into readable human language. Natural language understanding systems convert samples of human language into more formal representations that are easier for computer programs to manipulate. Many problems within NLP apply to both generation and understanding; for example, a computer must be able to model morphology (the structure of words) in order to understand an English sentence, but a model of morphology is also needed for producing a grammatically correct English sentence. NLP has significant overlap with the field of computational linguistics, and is often considered a sub-field of artificial intelligence. The term natural language is used to distinguish human languages (such as Spanish, Swahili or Swedish) from formal or computer languages (such as C++, Java or LISP). Although NLP may encompass both text and speech, work on speech processing has evolved into a separate

Notes

field. The Redmond-based Natural Language Processing group is focused on developing efficient algorithms to process texts and to make their information accessible to computer applications. Since text can contain information at many different granularities, from simple word or token-based representations, to rich hierarchical syntactic representations, to high-level logical representations across document collections, the group seeks to work at the right level of analysis for the application concerned.

11.1 Natural Language Processing – Overview

The goal of the Natural Language Processing (NLP) group is to design and build software that will analyze, understand, and generate languages that humans use naturally, so that eventually you will be able to address your computer as though you were addressing another person.

This goal is not easy to reach. "Understanding" language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way. It's ironic that natural language, the symbol system that is easiest for humans to learn and use, is hardest for a computer to master. Long after machines have proven capable of inverting large matrices with speed and grace, they still fail to master the basics of our spoken and written languages.

The challenges we face stem from the highly ambiguous nature of natural language.

As an English speaker you effortlessly understand a sentence like "Flying planes can be dangerous". Yet this sentence presents difficulties to a software program that lacks both your knowledge of the world and your experience with linguistic structures. Is the more plausible interpretation that the pilot is at risk, or that the danger is to people on the ground? Should "can" be analyzed as a verb or as a noun? Which of the many possible meanings of "plane" is relevant? Depending on context, "plane" could refer to, among other things, an airplane, a geometric object, or a woodworking tool. How much and what sort of context needs to be brought to bear on these questions in order to adequately disambiguate the sentence?

We address these problems using a mix of knowledge-engineered and statistical/machine-learning techniques to disambiguate and respond to natural language input. Our work has implications for applications like text critiquing, information retrieval, question answering, summarization, gaming, and translation. The grammar checkers in Office for English, French, German, and Spanish are outgrowths of our research; Encarta uses our technology to retrieve answers to user questions; Intellishrink uses natural language technology to compress cellphone messages; Microsoft Product Support uses our machine translation software to translate the Microsoft Knowledge Base into other languages. As our work evolves, we expect it to enable any area where human users can benefit by communicating with their computers in a natural way.

11.1.1 Evaluation in NLP

The first evaluation campaign on written texts seems to be a campaign dedicated to message understanding in 1987 (Pallet 1998). Then, the Parseval/GEIG project compared phrase-structure grammars (Black 1991). A series of campaigns within Tipster project were realized on tasks like summarization, translation and searching (Hirshman 1998). In 1994, in Germany, the Morpholympics compared German taggers. Then, the Senseval and Romanseval campaigns were conducted with the objectives of semantic disambiguation. In 1996, the Sparkle campaign compared syntactic parsers in four different languages (English, French, German and Italian).

In France, the Grace project compared a set of 21 taggers for French in 1997 (Adda 1999). In 2004, during the Technolangue/Easy project, 13 parsers for French were compared. Large-scale

Notes

evaluation of dependency parsers were performed in the context of the CoNLL shared tasks in 2006 and 2007. In Italy, the evalita campaign was conducted in 2007 to compare various tools for Italian evalita web site. In France, within the ANR-Passage project (end of 2007), 10 parsers for French were compared passage web site. Adda G., Mariani J., Paroubek P., Rajman M. 1999 L'action GRACE d'évaluation de l'assignation des parties du discours pour le français. Langues vol-2 Black E., Abney S., Flickinger D., Gdaniec C., Grishman R., Harrison P., Hindle D., Ingria R., Jelinek F., Klavans J., Liberman M., Marcus M., Reukos S., Santoni B., Strzalkowski T. 1991 A procedure for quantitatively comparing the syntactic coverage of English grammars. DARPA Speech and Natural Language Workshop Hirshman L. 1998 Language understanding evaluation: lessons learned from MUC and ATIS. LREC Granada Pallet D.S. 1998 The NIST role in automatic speech recognition benchmark tests.

11.1.2 Tasks and Limitations of NLP

In theory, natural-language processing is a very attractive method of human-computer interaction. Early systems such as SHRDLU, working in restricted "blocks worlds" with restricted vocabularies, worked extremely well, leading researchers to excessive optimism, which was soon lost when the systems were extended to more realistic situations with real-world ambiguity and complexity. Natural-language understanding is sometimes referred to as an AI-complete problem, because natural-language recognition seems to require extensive knowledge about the outside world and the ability to manipulate it. The definition of "understanding" is one of the major problems in natural-language processing.

11.1.3 Sub-problems of NLP

Speech Segmentation

In most spoken languages, the sounds representing successive letters blend into each other, so the conversion of the analog signal to discrete characters can be a very difficult process. Also, in natural speech there are hardly any pauses between successive words; the location of those boundaries usually must take into account grammatical and semantic constraints, as well as the context.

Text Segmentation

Some written languages like Chinese, Japanese and Thai do not have single-word boundaries either, so any significant text parsing usually requires the identification of word boundaries, which is often a non-trivial task.

Part-of-speech Tagging

Word sense disambiguation: Many words have more than one meaning; we have to select the meaning which makes the most sense in context.

Syntactic Ambiguity

The grammar for natural languages is ambiguous, i.e. there are often multiple possible parse trees for a given sentence. Choosing the most appropriate one usually requires semantic and contextual information. Specific problem components of syntactic ambiguity include sentence boundary disambiguation.

Notes

Imperfect or Irregular Input

Foreign or regional accents and vocal impediments in speech; typing or grammatical errors, OCR errors in texts.

Speech Acts and Plans

A sentence can often be considered an action by the speaker. The sentence structure alone may not contain enough information to define this action. For instance, a question is actually the speaker requesting some sort of response from the listener. The desired response may be verbal, physical, or some combination.

Example: "Can you pass the class?" is a request for a simple yes-or-no answer, while "Can you pass the salt?" is requesting a physical action to be performed. It is not appropriate to respond with "Yes, I can pass the salt," without the accompanying action (although "No" or "I can't reach the salt" would explain a lack of action).

Task What are the disadvantages and problems of NLP? Discuss briefly.

Self Assessment

Fill in the blanks:

1. ..... is a field of computer science concerned with the interactions between computers and human (natural) languages.
2. "....." language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way.
3. Specific problem components of ..... ambiguity include sentence boundary disambiguation.
4. A ..... can often be considered an action by the speaker.
5. The goal of the Natural Language Processing (NLP) group is to design and build ..... that will analyze, understand, and generate languages that humans use naturally.

11.2 Steps in Natural Language Processing

The steps in Natural Language Processing are as follows:

1. ***Morphological Analysis:*** Individual worlds are scrutinized into their components and non word tokens, like punctuation are alienated from the words.
2. ***Syntactic Analysis:*** Linear sequences of words are malformed into structures that illustrate how the words associate to each other. Some word sequences may be discarded if they disobey the languages rules for how words may be united.
3. ***Semantic Analysis:*** The structures generated by the syntactic analyzer are allocated meanings.

Notes

4. ***Discourse Integration:*** The meaning of an individual sentences may rely on the sentences that head it and may affect the meanings of the sentence (may rely on the sentences that come first) that chase it.

5. ***Pragmatic Analysis:*** The structure displaying what was said is reinterpreted to verify that what was in fact meant.

Example: The sentence "Do you know what time it is?" should be deduced as a request to be informed the time.

These steps are discussed in detail as below.

11.2.1 Morphological Analysis

As a problem structuring and problem solving technique, morphological analysis was designed for multi-dimensional, non-quantifiable problems where causal modeling and simulation do not function well or at all. Zwicky developed this approach to address seemingly non-reducible complexity. Using the technique of Cross Consistency Assessment (CCA) (Ritchey, 2002), the system however does allow for reduction, not by reducing the number of variables involved, but by reducing the number of possible solutions through the elimination of the illogical solution combinations in a grid box. From Wikipedia, the free encyclopedia.

What is "Morphological Analyzer" and How it Related to Search Engines?

A morphological analyzer is a program for analyzing the morphology of an input word, the analyzer including a recognition engine, identifying suffixes and finding a stem within the input word algorithms. Morphological analyzers are using lexicon/thesaurus, keep/stop lists, and indexing engines for their process. Google using morphological analysis across all its products.

Example: Broad Match: This is the default option. If you include general keyword or keyword phrases – such as tennis shoes – in your keyword list, your ads will appear when a user's query contains tennis and shoes, in any order, and possibly along with other terms. Your ads will also automatically show for expanded matches, including plurals and relevant variations. Because broad matches are sometimes less targeted than exact or phrase matches, you should create keyword phrases containing at least two descriptive words each.

11.2.2 Syntactic Processing

This level concentrates on scrutinizing the words in a sentence so as to reveal the grammatical arrangement of the sentence. This needs both a grammar and a parser. The output of this level of processing is a (perhaps delinearized) demonstration of the sentence that discloses the structural dependency relationships among the words. There are numerous grammars that can be utilized, and which will, in turn, impact the choice of a parser. Not all NLP applications require a full parse of sentences, thus the remaining challenges in parsing of prepositional phrase attachment and conjunction scoping no longer stymie those applications for which phrasal and clausal dependencies are adequate.

Caution Syntax transmits meaning in most languages since order and dependency contribute to meaning.

Notes

Example: The two sentences: 'The dog followed the cat' and 'The cat followed the dog' varies only in terms of syntax, yet transmit quite diverse meanings.

11.2.3 Semantic Analysis

The development of object-oriented software starts from requirements expressed commonly as Use Cases. The requirements are then converted into a conceptual or analysis model. Analysis is a fundamental stage because the conceptual model can be shown to satisfy the requirements and becomes the skeleton on which the complete system is built. Most of the use of software patterns until now has been at the design stage and they are applied to provide extensibility and flexibility.

However, design patterns don't help avoid analysis errors or make analysis easier. Analysis patterns can contribute more to reusability and software quality than the other varieties. Also, their use contributes to simplifying the development of the analysis model. In particular, a new type of analysis pattern is proposed, called a Semantic Analysis Pattern (SAP), which is in essence a mini application, realizing a few Use Cases or a small set of requirements. Using SAPs, a methodology is developed to build the conceptual model in a systematic way. No good design or correct implementation is possible without good analysis; the best C++ or Java programmers cannot make up for conceptual errors.

Caution The correction of analysis errors becomes very expensive when these errors are caught in the code.

It is therefore surprising how poorly understood is this stage and how current industrial practice and publications show a large number of analysis errors. We have found that industrial software developers usually have trouble with analysis. What are worse, even serious journals and conferences publish papers or tutorials that contain clear analysis errors.

A possible improvement to this situation may come from the use of patterns. A pattern is a recurring combination of meaningful units that occurs in some context. Patterns have been used in building construction, enterprise management, and in several other fields. Their use in software is becoming very important because of their value for reusability and quality; they distill the knowledge and experience of many designers. Most of the use of patterns until now has been at the design stage. However, design patterns don't help to avoid analysis errors or to make analysis easier. We believe that we need analysis patterns to improve the quality of analysis and they can contribute more to reusability and software quality than the other varieties. We also intend to show that their use contributes to simplifying the development of the application analysis model. In particular, we propose a new type of analysis pattern, called a Semantic Analysis Pattern (SAP), which is in essence a mini application, realizing a few Use Cases or a small set of requirements. Using SAPs we develop a methodology to build the conceptual model in a systematic way. We use UML (Unified Modeling Language), as a language to describe our examples.

Analysis Patterns and their Use

The value of analysis is played down in practice. The majorities of the papers published about object-oriented design as well as the majority of textbooks concentrate on implementation. Books on Java, C++, and other languages outnumber by far the books on object-oriented analysis/design (OOA/OOD). On top of that, most books on OOA/OOD present very simple examples. To make things worse, professional programmers need to implement as soon as possible, there is pressure to show running code and they may skip the analysis stage altogether.

Notes

What is deceiving is that software may appear to work properly but may have errors, not be reusable or extensible, be unnecessarily complex. In fact, most of the software built without some model exhibits some or all of these defects. Most schools emphasize algorithms, not the development of software systems. There is a large literature on methods of system development that although oriented to other disciplines, is very applicable to software, but rarely used (In fact, design patterns originated from ideas about buildings).

Notes Some people believe that with components we don't need to understand what is inside each component. The result of all this is that analysis is skipped or done poorly.

We need to look for ways to make analysis more precise and easier for developers.

The use of patterns is a promising avenue. A Semantic Analysis Pattern is a pattern that describes a small set of coherent Use Cases that together describe a basic generic application.

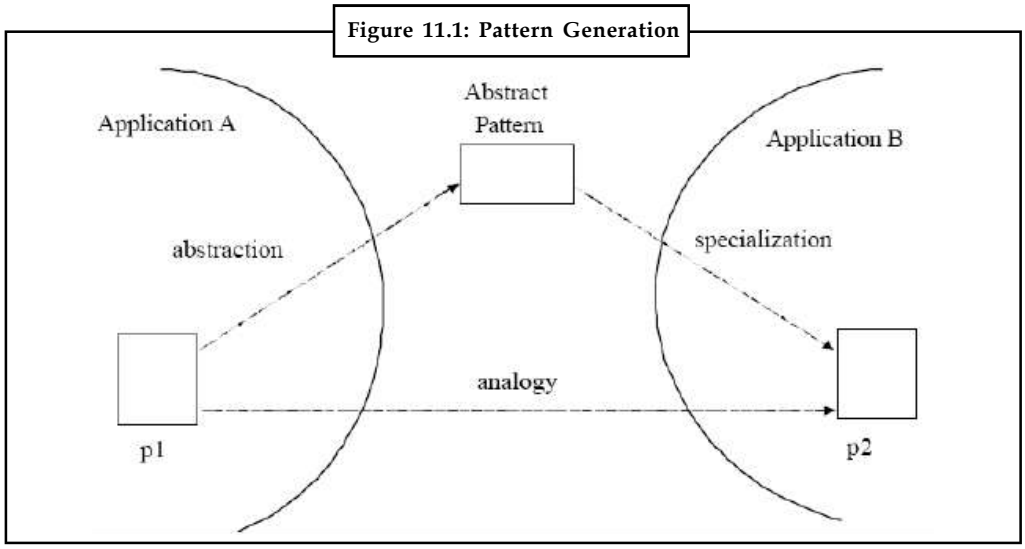
Did u know? The Use Cases are selected in such a way that the application can fit a variety of situations.

Semantic Analysis Patterns differ from design patterns in the following ways:

- Design patterns are closer to implementation, they focus on typical design aspects, e.g., user interfaces, creation of objects, basic structural properties.
- Design patterns apply to any application; for instance, all applications have user interfaces, they all need to create objects.
- Design patterns intend to increase the flexibility of a model by decoupling some aspects of a class.

An instance of a SAP is produced in the usual way: Use Cases, class and dynamic diagrams, etc. We select the Use Cases in such a way that they leave out aspects which may not be transportable to other applications. We can then generalize the original pattern by abstracting its components and later we derive new patterns from the abstract pattern by specializing it (Figure 11.1). We can also use analogy to directly apply the original pattern to a different situation.

Figure 11.1: Pattern Generation



Notes

We develop first a pattern from some basic use cases. We then use analogy to apply it to a different situation, then we generalize it and finally we produce another pattern for another application specializing the abstract pattern. We then show how to use these patterns in building conceptual models.

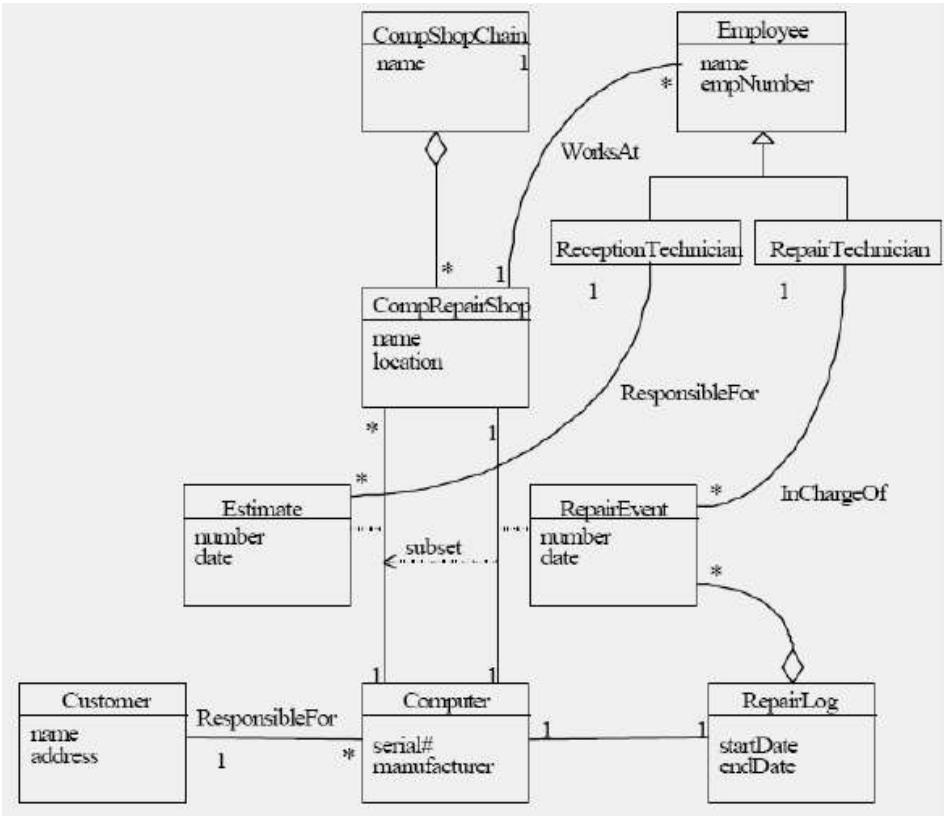
Example: Consider a design for a computer repair shop. The specifications for this application are: A computer repair shop fixes broken computers. The shop is part of a chain of similar shops. Customers bring computers to the shop for repair and a reception technician makes an estimate. If the customer agrees, the computer is assigned for repair to some repair technician, who keeps a Repair Event document. All the Repair Event documents for a computer are collected in its repair log. A repair event may be suspended because of a lack of parts or other reasons.

These requirements correspond to two basic Use Cases:

- Get an estimate for a repair
- Repair a computer

A class diagram for this system is shown in Figure 11.2, while Figure 11.3 shows a state diagram for Repair Event. Figure 11.4 shows a sequence diagram for assigning the repair of some computer to a technician. The class diagram reflects the facts that a computer can be estimated at different shops in the chain and that one of these estimates may become an actual repair. A computer that has been repaired at least once has a repair log that collects all its repair events. The collection of repair shops is described by the repair shops chain.

Figure 11.2: Class Diagram for the Computer Repair Shop



Notes

Figure 11.3: State Diagram for Repair Event

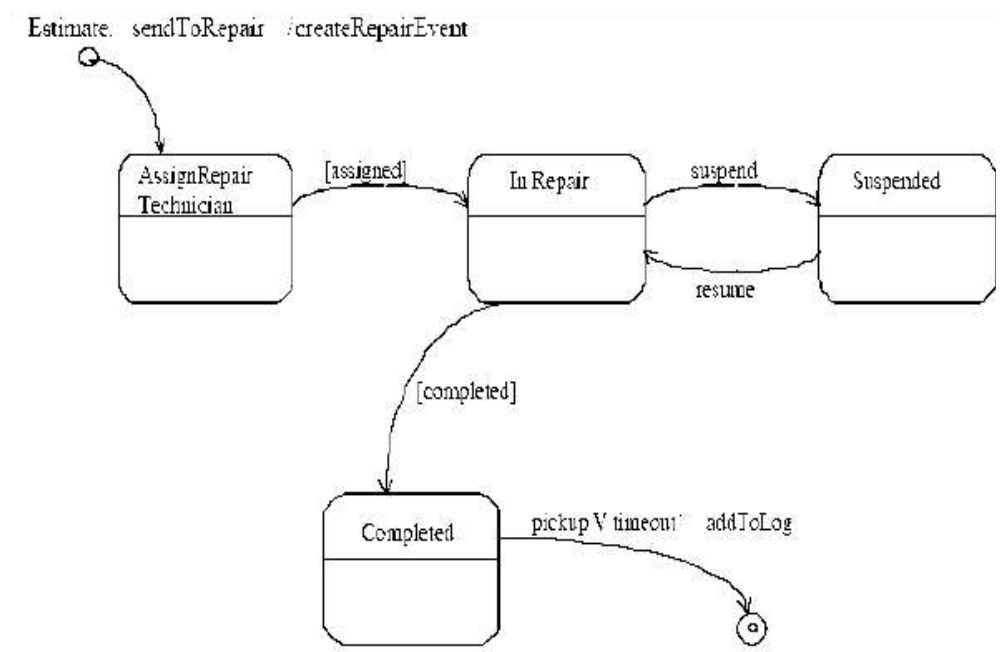
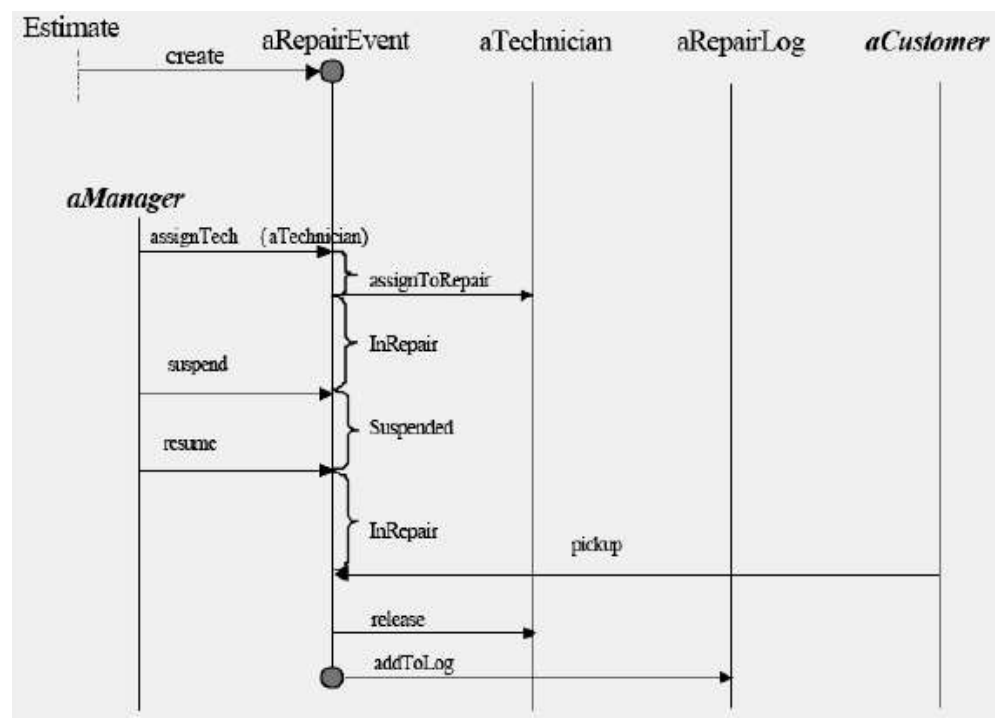


Figure 11.4: Sequence Diagram for Assigning Repair Jobs to Technicians



Notes

11.2.4 Discourse and Pragmatic Processing

In the recent decades, the need for an ability to treat discourse dependent factors such as anaphora resolution has increased in the field of NLP. This calls for a greater understanding of intersentential dependencies, in terms of how the discourse structure is built both in written narrative discourse and spoken dialogue. In the Survey of the State of the Art in Human Language Technology Barbara Grosz (1997) formulates two core questions in research on discourse: (1) What information is contained in extended sequences of utterances that goes beyond the meaning of the individual utterances? and (2) How does the context in which an utterance is used affect the meaning of the individual utterances, or parts of them? The first question thus concerns the issue of inferences, associations and implicit information, which in different ways can be derived from the utterances, and from combinations of utterances. Attempts to formalize these relations have been made in the extensive research on discourse relations (e.g. Hobbs, 1985, Mann & Thompson, 1988, Asher, 1993). Also the issue of knowledge representation is relevant here. The second question concerns the fuzzier field of domain dependence, i.e. what information is relevant for the specific discourse setting, what might be presupposed and what kind of conventions are to be applied in the specific discourse. Also the question of cognitive processing of discourse as well as the situation setting is relevant here.

When it comes to spoken discourses, there is a need to connect international features with discourse factors is present in e.g. automatic summarization of spoken language, or in spoken dialogue systems. The international features are affecting the discourse in dimensions as e.g. discourse segmenting (prosodic phrasing) and the assignment of discourse relations, such as e.g. contrast. Also salience, of focality, is affected by the prosody.

This will give an overview of factors relevant in discourse processing. These phenomena include the concept of salience/focus, anaphora resolution, cohesion, coherence discourse relations and discourse segments.

What is Discourse?

The term discourse includes both spoken and written forms, as well as both monologue and dialogue, i.e. "discourse" is taken to be the most super ordinate term. The difference between a discourse and an arbitrary collection of utterances lies in the phenomenon of coherence. A discourse is any stretch of coherent utterances.

This means that the utterances together constitute an intentionally meaningful message. The establishing of coherence can be made in many ways, but the result is always some kind of relations between items on different levels in the discourse.

Notes However, to assume relations in discourse implies also to assume units between which these relations holds, and this leads us in to the field of discourse segmenting and the (hierarchical) discourse structure.

Discourse Specific Issues

Some discourse specific issues are discussed as below:

1. ***Focus:*** The first phenomenon to discuss in trying to get hold of coherence in discourse is focus. Focus is often mentioned feature in research on discourse. The notion of focus is used for a number of aspects of salience or importance, and the most general description

Notes

is "the most important element". An important question, which complicates the picture, is then: important for whom/what? I.e. important to e.g. the speaker, the listener, the global message or the local utterance? Thus, focus is a concept simple to understand, but it is hopelessly difficult to pin down exactly. In this paper the term is used in the meaning salience or focus of attention. If it is used in some other meaning, this is indicated in the text. For a survey of the different uses of the term focus, see Gundel (1995). The feature focus is connected to cognitive processing, memory representation and linguistic representation, in that a more activated/Given/accessible concept is represented by a more reduced linguistic sign (Ariel, 1990, Gundel, Hedberg & Zacharski, 1993). Of course focus is also connected to the syntax of the utterance and the semantics (e.g. Sidner, 1983) as well as the prosody (Bruce, 1998, van Donzel, 1999). Van Donzel made extensive investigations on basis of Princes (1983) Given – New taxonomy, and her findings confirms the picture that Given elements are also acoustically more reduced than New elements. Thus, the intonation plays a great part for the information structure in spoken language, however, it is also possible to do without it, as in written language. Ability to predict focus should improve the assignment of prosodic features in speech synthesis, however, the task seems to be very difficult, (if not impossible) Bolinger (1972), and the result in synthesis are also today limited. Focality is further an often used feature in anaphora resolution.

2. ***Anaphora Resolution:*** Very closely connected to the research on focus and also to the research on discourse coherence is the research on anaphora resolution. This because anaphora is affected by focality and further context dependent. This means that anaphoric expressions are representing referents in focus, and they are at the same time establishing links back to the preceding context (the antecedent).

   Traditionally, the concept of anaphora was limited to co-referent NPs, this means that the only antecedents we have to keep available in the discourse record are the NP objects. However, this approach becomes to narrow if one want to give account for more complex, or implicit kind of anaphora, such as situation anaphora, abstract object anaphora or associative anaphora (e.g. Webber, 1990, Fraurud, 1992, Asher, 1993, Dahl & Hellman, 1995), or connected phenomena treated as anaphora such as presuppositions (van der Sandt, 1992) and referent coercion (e.g. Dahl & Hellman, 1995).

   It is clear, that to refer back to those more abstract or implicit objects, it is not enough to just keep NP objects available in the discourse record. Rather it is needed a discourse record with information about e.g. situations and associative connections (semantically related items). It is also worth noticing that while co-referent NP anaphora has a clearly anaphoric identity, the abstract object anaphora and the associative anaphora might be more easily understood as discourse relations, e.g. rhetorical relations (Mann & Thompson, 1988) or coherence relations (Hobbs, 1995). There is a large body of work on automatic anaphora resolution, however, often it is delimited to resolution of NP anaphora. Sidner (1983) makes explicitly use of the concept of focus, for resolving anaphora, i.e. her anaphora resolution algorithm is in the first step predicting focus, and in the second step choosing the antecedent on basis of the focus ranking. The use of Centering Theory (Grosz, Joshi & Weinstein, 1995) as an algorithm for anaphora resolution works in the same way, i.e. (a) predict focus, (b) choose antecedents from the ranked elements. Eckert and Strube (1998) has used the Centering theory algorithm, extended with speech act tagging, to develop an algorithm for resolution of abstract object anaphora in dialogue. Also Harabagiu (1999) have done work on automatic co-reference resolution.

   To investigate anaphora might be viewed as a way to investigate the "symptom" of coherence in discourse.

Notes

3. ***Cohesion and Coherence:*** Coherence is what makes a collections of sentences/utterances a discourse, but what exactly is it? Coherence might be defined as implicit relations between different parts of the discourse. Coherence is closely connected to the concept of cohesion, which means explicit markers of relations between different parts of the discourse.

Example: Coherence is in many cases established with cohesive devices, examples of such cohesive devices is e.g. discourse markers or anaphoric pronouns. E.g. Harabagiu (1999) stresses the importance of cohesive devices for coherence in discourse.

Discourse markers, or cue phrases, are said to indicate discourse relations, but they are also said to have the function of marking out discourse boundaries, i.e. to perform discourse segmenting.

Example: Some discourse markers are: "but", "anyway", "okay" etc.

Research aiming to give account for coherence in discourse has been conducted by e.g. Grosz & Sidner (1986), who have developed a discourse theory, which gives account for coherence on a global level of discourse, but with a limited inventory of discourse relations. The discourse relations by Grozs & Sidner (1986) are rather solely structure indicating (i.e. mother-daughter relation or sister relation).

Did u know? Discourse markers are primary seen as segmenting devices, and the specific relational information is not stressed.

Grosz, Joshi & Weinstein (1995) have with CT developed a way to give account for the degree of coherence in a discourse. Their approach to coherence is based on anaphoric relations, and does not pose any extra meaning in the relations between utterances.

The CT has been suggested many modifications, e.g. by 1998, Strube & Hahn, (1998, change of ranking criteria), Walker (1997, change of discourse segmenting), Passonneau, (1993, adding a more hierarchical structure), Beaver, (2000, rewriting CT in Optimality theory). Also evaluations have been made, e.g. by Byron & Stent (1998), Poesio et al. (2000), Tetrault, (forthc).

4. ***Discourse Relations:*** Discourse markers are often seen as establishing a relation between parts of the discourse, e.g. the discourse marker "but" can establish the relation contrast between two utterances or discourse segments. In this case we can see the relation as marked on the surface level. The relation "contrast" can furthermore be signaled prosodically.

The role of the meaning by discourse relations might be more stressed than by e.g. Grosz & Sidner (1986). In this case the discourse relations can be described as inferences drawn on basis of what is said in the discourse. The inferences thus enriches the discourse meaning, and might also function as a glue between utterances/segments, and the inferences then contributes to the meaning of the message, and not just to the structure. The inferences can be drawn on basis of adjacency, but are still mainly semantic to their nature, i.e. they are not dependent on the adjacency in the same way as the surface signaled discourse relations are. In the case of inferences, the inference machinery, as well as the knowledge representation is of course crucial.

There have been many attempts to give account for coherence in discourse, in terms of discourse relations, e.g. by Hobbs (1985) and by Mann & Thompson (1988). Specially Hobbs have been interested in how to compute the discourse relations, on basis of what

Notes

inference machinery and what knowledge representation to use. Mann and Thompson has focused more on the meaning of the relations, and has a more surface based approach (i.e. discourse markers). The treatment of discourse relations has also been integrated in DRT (Asher, 1993). Asher has used an inventory of discourse relations, much inspired by Mann & Thompson. Multiple functions by discourse markers have been investigated by e.g. Hovy (1997), but it is in place to note, that the issue of multiple discourse relations, still is an open question.

One weakness of the above mentioned approaches to discourse relations is that no distinction is made concerning on what kind of basis, or from what knowledge sources, the inferences are drawn. How can we make distinctions between inferences on basis of what source of knowledge they are computed from? Is the inference based fully on the semantic content in the utterance, is it based on world/domain knowledge, or is it based on socio-cultural knowledge. Of course these sources of knowledge might blend, but it is still the authors feeling, that different kind of knowledge should be separated or at least marked out in the knowledge representation, as is the attempt by e.g. Sanders *et al.* (1992). The inferences, or discourse relations, always holds between parts of the discourse. These parts are often called discourse segments, but the criteria and strategies in discourse segmenting is not very well established. However, the discourse segmenting is crucial in establishing discourse relations, and also in the description of coherence.

Discourse Segments

As mentioned, one way to perform discourse segmenting is on basis of cue phrases. In written language one can also make use of punctuation and in spoken language the prosodic phrasing.

To establish boundaries implies to make up groupings in between those boundaries.

It is worth to stress that the discourse boundaries do not have to be absolute, i.e. perhaps the boundary is rather a field than a specific point, and it should also be pointed out that discourse boundaries might be a function of the grouping. As mentioned earlier, discourse markers have the double function of dividing parts of the discourse, and at the same time connect those parts with some discourse or coherence relation. This is quite similar to the role of prosody; Prosody might also be regarded as a segmenting device, but this segmenting function is double, in that what defines a boundary also defines a non-boundary. It is an open issue whether we primarily shall focus on the boundary fields, or on the non boundary-fields. Polanyi (1988) builds discourse segments on basis of meaning clusters, i.e. she starts on phrase level and merges phrases which are semantically close to each other into a larger unit. The boundaries then become a result, when two segments cannot be merged, because they are not semantically close enough. This kind of discourse segments might thus be described as clusters of meaning. A similar approach is taken by Hearst (1997), who bases discourse segmenting on term repetition. The parallel to the prosodic grouping of speech can be made, where the prosodic boundary is not a feature of its own, but rather a result of the complete prosodic utterance contour. Bruce 1998 stresses this double function for prosody: Prosody has the function of both boundaries marking and grouping.

However, the textual surface approach with punctuation and cue phrases is more widely investigated in discourse segmenting, e.g. by Oberlander & Moore (1999), Albritton & Moore (1999), Flammia (1998). The base for discourse segments are further often claimed to be intention based (Grosz and Sidner, 1986), or information based (Mann & Thompson, 1988). The difference lies in whether the segmenting is made on basis of the intentions behind the message, or the information conveyed in the message. The intentions based approach is quite closely related to speech act theory and dialogue coding, as described by Carletta et al. (1997). Intentions based discourse segments and information based discourse segments have traditionally been viewed

Notes

as incompatible approaches, but Moser and Moore (1996), as well as Marcu, (1997) have suggested that this is not the case.

Now briefly mention some NLP-methods used for discourse processing.

Methods

The primary problem with the methods, is (at least in the authors eyes) not what method to choose, but what unit to manipulate with; What features, and in what relation to each other?

In discourse processing a mixture of NLP-methods can be used. This because in performing discourse processing, one need the whole arsenal of tools for the lower levels, e.g. tagging, parsing. The really difficult problem with discourse processing is to isolate the relevant features, and to make use of them in an efficient way, i.e. what kind of information is relevant to tag, what information is relevant to store, and what kinds of information is needed in different applications? How to categorise, how to remember, and what perspective to take. Technically we are free to choose any method that we might use for e.g. PoS - tagging, but we must first isolate what kind of units or categories that is relevant to mark up. This means that for a certain application, it might be the case that there is no need to give account for the full complexity in discourse processing, but a shallow analysis would do. In another application, a more fine grained analysis might be needed. Finite state methods have been used for discourse processing in terms of information extraction (Hobbs et al. 1997). The system FASTUS uses a cascaded non-deterministic finite-state automaton. The system is in five steps extracting (1) names and fixed expressions, (2) basic noun groups, verb groups prepositions and other particles, (3) complex noun groups and verb groups, (4) corresponding event structures, (5) distinct event structures that describe the same event are detected and merged. The "lean" finite-state method was claimed to be very successful for the task, as compared to the more complicated TACITUS system (*Hobbs et al.,* 1993), which included representation of discourse relations, based on abductive inferences.

Example: FASTUS seems to be an example of a successful limitation of steps to carry out of discourse processing in the task of information extraction.

Statistical methods for anaphora resolution have been reported by Mitkov & Schmidt (1996). The strategy used was an Uncertainty Reasoning approach, i.e. a scoring system was used, and in the end the candidate with the highest score was chosen. This strategy performed slightly worse, than one based on constraints and preferences, i.e. a more rule-based approach (Mitkov and Schmidt, 1996). *Machine learning* has been used for discourse segmenting by Passonneau & Litman (1997). On basis of human discourse segmenting the machine learning algorithm was trained. The results gave about the same accuracy as human annotators.

Applications

In this step the second question posed by Barbara Grosz (1997) becomes important, i.e. how to generally understand the issue of adoption? This issue is less investigated than the issues addressed earlier, try to avoid this field as much as possible.

1. ***Natural Language Understanding:*** In the NLU system TACITUS Hobbs *et al.* (1993) have made use of coherence relation, in aim to get a full representation of the message. The system is to a certain extent based on coherence relations, as described by Hobbs (1985), but the inference machinery is based on abduction. The knowledge base is an important factor.

2. ***Automatic Summarization:*** Automatic text summarization is an application where discourse understanding is crucial, i.e. it is important to be able to extract what is central

Notes

to the message. Summarization systems have been developed by e.g. Marcu, (1997). The approach was to parse the discourse relations (rhetorical relations, Mann & Thompson, 1988), and on basis of the parse be able to choose what discourse segments were most relevant. The approach relied to a large extent on surface cues, such as discourse markers.

3. ***Natural Language Generation:*** Marcu, 1997, also tested rhetorical relations for Natural Language Generation (NLG). On basis of rhetorical relations the ordering preferences of discourse segments was scored. The higher the score, the more likely that the discourse structure was coherent. Kibble & Power (1999) uses Centering theory for planning the most coherent stretch of utterances. These brief examples are of course just a very limited sample of applications which uses some kind of discourse related information. Still I will finish here and make some concluding remarks, and connect to my own dissertation subject.

Task Illustrate the term "discourse".

Self Assessment

Fill in the blanks:

6. In ..... *Analysis*, Individual worlds are scrutinized into their components and non word tokens, like punctuation are alienated from the words.
7. In ....., Linear sequences of words are malformed into structures that illustrate how the words associate to each other.
8. ..... concentrates on scrutinizing the words in a sentence so as to reveal the grammatical arrangement of the sentence.
9. Design patterns intend to increase the flexibility of a model by ..... some aspects of a class.
10. The term ..... includes both spoken and written forms, as well as both monologue and dialogue.
11. ..... is what makes a collections of sentences/utterances a discourse.

11.3 Spell Checking

The goal of spell checking is the detection and rectification of typographic and orthographic faults in the text at the level of word incidence measured out of its perspective.

No one can write without any faults. Even people well familiar with the rules of language can, just by misfortune, press a wrong key on the keyboard (maybe adjoining to the correct one) or miss out a letter. Moreover, when typing, one at times does not harmonize correctly the movements of the hands and fingers. All such errors are known as typos , or typographic errors. Alternatively, some people do not recognize the correct spelling of some words, particularly in a foreign language. Such errors are known as spelling errors.

Initially, a spell checker simply detects the strings that are not accurate words in a specified natural language. It is believed that most of the orthographic or typographic errors lead to strings that are impracticable as separate words in this language. Identifying the errors that exchange by accident one word into another obtainable word, like English 'then' into 'than' or Spanish 'czar' into 'Caesar', considers a task which needs much more influential tools.

Notes

After such impracticable string has been identified and highlighted by the program, the user can correct this string in any preferable way— manually or with the help of the program.

Example: If we try to insert into any English text the strings 3*groop, *greit, or *misanderstand, the spell checker will detect the error and stop at this string, emphasizing it for the user. Corresponding examples in Spanish can be * caió, * systema, *nesecitar.

The functions of a spell checker can be more adaptable. The program can also suggest a set of accessible words, which are alike enough to the specified corrupted word, and the user can then select one of them as the correct edition of the word, without retyping it in the line. In the preceding examples, Microsoft Word's spell checker provides, as possible candidates for substitution of the string caió, the existing Spanish words revealed in Figure 11.5. Mostly, especially for long strings, a spell checker provides only one or two candidates (or none).

Example: For the string* systema it provides only the correct Spanish word sistema.

The programs that carry out operations of both types are known as orthographic correctors, while in English they are generally known as spellcheckers. In daily practice, spell checkers are measured as very cooperative and are used by millions of users all through the world. The majority of modern text editors are supplied now with incorporated spell checkers.

Microsoft Word uses many spellcheckers, a particular one for each natural language used in the text.

Figure 11.5

The amount of linguistic information required for spell checkers is much superior than for hyphenation. A straightforward but very resource-consuming strategy operates with a list, or a dictionary, of all applicable words in a particular language. It is required to have also a criterion of resemblance of words, and some suppositions regarding the most general typographic and spelling errors. A deeper penetration into the correction problems needs a detailed knowledge of morphology, as it assists the creation of a more dense dictionary has a convenient size.

Spell checkers have been obtainable for more than 20 years, but some relatively apparent tasks of rectification of words, even taken separately, have not been yet solved.

Notes

Example: To put a particular example, let us take the ungrammatical string * teached in an English text. None of the spell checkers we have attempted recommended the correct form taught. In a corresponding way, if a foreigner inserts into a Spanish text such strings as * muestrar or * disponido, the Spanish spellcheckers we have attempted did not provide the forms mostrar and dispuesto as probable corrections.

Self Assessment

Fill in the blanks:

12. The goal of ..... is the detection and rectification of typographic and orthographic faults in the text at the level of word incidence measured out of its perspective.
13. Some people do not recognize the correct spelling of some words, particularly in a foreign language. Such errors are known as ..... .
14. The amount of linguistic information required for spell checkers is much superior than for ..... .
15. Initially, a spell checker simply detects the strings that are not accurate words in a specified ..... language.

11.4 Summary

- Natural language processing is a field of computer science concerned with the interactions between computers and human (natural) languages.
- The goal of the Natural Language Processing (NLP) group is to design and build software that will analyze, understand, and generate languages that humans use naturally, so that eventually you will be able to address your computer as though you were addressing another person.
- In Syntactic Analysis, Linear sequences of words are malformed into structures that illustrate how the words associate to each other.
- Syntactic Processing concentrates on scrutinizing the words in a sentence so as to reveal the grammatical arrangement of the sentence. This needs both a grammar and a parser.
- The development of object-oriented software starts from requirements expressed commonly as Use Cases.
- The term discourse includes both spoken and written forms, as well as both monologue and dialogue, i.e. "discourse" is taken to be the most super ordinate term.
- Coherence might be defined as implicit relations between different parts of the discourse. Coherence is closely connected to the concept of cohesion, which means explicit markers of relations between different parts of the discourse.
- The goal of spell checking is the detection and rectification of typographic and orthographic faults in the text at the level of word incidence measured out of its perspective.

11.5 Keywords

Discourse: The term discourse includes both spoken and written forms, as well as both monologue and dialogue, i.e. "discourse" is taken to be the most super ordinate term.

Notes

Natural Language Processing: Natural language processing is a field of computer science concerned with the interactions between computers and human (natural) languages.

Pragmatic Analysis: The structure displaying what was said is reinterpreted to verify that what was in fact meant.

Spell Checking: The goal of spell checking is the detection and rectification of typographic and orthographic faults in the text at the level of word incidence measured out of its perspective.

11.6 Review Questions

1. What is Natural Language Processing? Illustrate the concept of evaluation in NLP.
2. Elucidate the sub-problems used in NLP.
3. Explain the steps followed in Natural Language Processing.
4. What are the Major tasks in NLP? Illustrate.
5. Syntactic Processing concentrates on scrutinizing the words in a sentence so as to reveal the grammatical arrangement of the sentence. Comment.
6. How does analysis pattern differ from design patterns. Exemplify.
7. Explain the process of Discourse and Pragmatic Processing in NLP.
8. Illustrate the process of performing spell checking.
9. What are the various specific issues concerning Discourse? Explicate.
10. Illuminate some NLP-methods used for discourse processing.

Answers: Self Assessment

1.	Natural Language Processing	2.	Understanding
3.	Syntactic	4.	Sentence
5.	Software	6.	Morphological
7.	Syntactic Analysis	8.	Syntactic Processing
9.	Decoupling	10.	Discourse
11.	Coherence	12.	Spell Checking
13.	Spelling Errors	14.	Hyphenation
15.	Natural		

11.7 Further Readings

Books

Arsanjani, A., Service provider: *A Domain Pattern and its Business Framework Implementation*, Procs. of PloP'99.

Booch, G., Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide*, Addison-Wesley 1998.

Braga, R.T.V., Germano, F.S.R., Masiero, P.C.: *A Confederation of Patterns for Resource Management*, Procs. of PLoP'98.

Notes

Braga, R.T.V., Germano, F.S.R., Masiero, P.C.: *A pattern language for Business Resource management*, Procs. of PloP'99.

Coad, P.: *Object Models – Strategies, Patterns and Applications (2nd. Edition)*, Prentice-Hall 1997.

Fernandez, E. B., Stock manager: *An Analysis Pattern for Inventories,* Procs. of PloP 2000.

Fernandez, E.B, Yuan, X.: *An Analysis Pattern for Reservation and Use of Entities*, Procs. Of PLoP99 , http://st-www.cs.uiuc.edu/~plop/plop99.

Fernandez, E.B., *Good Analysis as the basis for Good Design and Implementation, Report* TRCSE-97-45, Dept. of Computer Science and Eng., Florida Atlantic University, September 1997. Presented at OOPSLAÊ97.

Fernandez, E.B., Hawkins, J.: *"Determining role rights from use cases",* Procs. 2nd *ACM Workshop on Role-Based Access Control*, 1997, 121-125.

Fernandez, E.B.: *"Building systems using analysis patterns"*, Procs. 3rd Int. *Soft. Architecture Workshop (ISAW3*), Orlando, FL , November 1998, 37-40.

Online link www.pcai.com/web/ai_info/natural_lang_proc.html

Notes

Unit 12: Learning

CONTENTS

Objectives

Introduction

12.1 Meaning of Learning

12.2 Rote Learning

12.3 Learning by Taking Advice

12.4 Learning from Examples

12.4.1 Induction

12.5 Explanation-based Learning

12.6 Learning by Parameter Adjustment

12.7 Learning with Macro-operators

12.8 Learning by Chunking

12.9 Discovery-based Learning

12.10 Learning by Correcting Mistakes

12.11 Learning by Recording Cases

12.12 Learning by Managing Multiple Models

12.13 Summary

12.14 Keywords

12.15 Review Questions

12.16 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the meaning of learning
- Illustrate the rote learning
- Discuss the learning from examples
- Understand the explanation based learning, etc.

Introduction

Learning is defined as modifications in the system that are adaptive in the sense that they facilitate the system to do the similar task or tasks taken from the similar population more competently and more successfully the next time.

Notes

Learning includes a broad range of phenomenon:

- ***Skill refinement:*** Practice makes abilities perk up. More you play, better you obtain
- ***Knowledge acquisition:*** Knowledge is usually attained via experience

12.1 Meaning of Learning

Learning is distinguished into a number of different forms. The simplest is learning by trial-and-error. For example, a simple program for solving mate-in-one chess problems might try out moves at random until one is found that achieves mate. The program remembers the successful move and next time the computer is given the same problem it is able to produce the answer immediately. The simple memorizing of individual items – solutions to problems, words of vocabulary, etc. – is known as rote learning.

Rote learning is relatively easy to implement on a computer. More challenging is the problem of implementing what is called generalization. Learning that involves generalization leaves the learner able to perform better in situations not previously encountered. A program that learns past tenses of regular English verbs by rote will not be able to produce the past tense of e.g. "jump" until presented at least once with "jumped", whereas a program that is able to generalize from examples can learn the "add-ed" rule, and so form the past tense of "jump" in the absence of any previous encounter with this verb. Sophisticated modern techniques enable programs to generalize complex rules from data.

Self Assessment

Fill in the blanks:

1. ..... is defined as modifications in the system that are adaptive in the sense that they facilitate the system to do the similar task or tasks taken from the similar population more competently and more successfully the next time.
2. Learning that involves ..... leaves the learner able to perform better in situations not previously encountered.

12.2 Rote Learning

Rote learning is the fundamental learning activity. It is also known as memorization since the knowledge, without any alteration is, just copied into the knowledge base. As calculated values are accumulated, this technique can save a important amount of time.

Rote learning method can also be utilized in multifaceted learning systems provided sophisticated techniques are engaged to use the accumulated values faster and there is a simplification to keep the number of amassed information down to a handy level.

Example: Checkers-playing program accesses this technique to study the board positions it assesses in its look-ahead hunt.

Example: A simple example of rote learning is *caching*

- Amass computed values (or large piece of data)
- Recall this information when needed by computation.

Notes

- Important time savings can be attained.
- Many AI programs (in addition to more general ones) have used caching very successfully.

Notes Memorization is a key requirement for learning:

- It is a fundamental requirement for any intelligent program — is it a separate learning process?
- Memorization can be a multifaceted subject — how best to amass knowledge?

Rote learning is basically a straightforward process. However it does demonstrate some issues that are pertinent to more intricate learning issues.

- ***Organization***
  - access of the accumulated value must be quicker than it would be to recompute it. Methods like hashing, indexing and sorting can be engaged to enable this.
  - *E.g.* Samuel's program indexed board positions by observing the number of pieces.
- ***Generalisation***
  - The number of potentially accumulated objects can be very huge. We may require to simplify some information to make the problem controllable.

Example: E.g. Samuel's program accumulated game positions only for white to move. Also rotations along diagonals are united.

- ***Stability of the Environment***
  - Rote learning is not very efficient in a speedily varying environment. If the environment does modify then we must identify and record exactly what has altered *the frame problem.*

Self Assessment

Fill in the blanks:

3. ..... is also known as memorization since the knowledge, without any alteration is, just copied into the knowledge base.
4. ..... is a key requirement for learning.

12.3 Learning by Taking Advice

This is a straightforward form of learning. Assume a programmer writes a set of directions to instruct the computer what to accomplish, the programmer is a educator and the computer is a learner. Once learned (i.e. programmed), the system will be in a situation to perform new things.

The advice may come from many sources: human experts, internet to name a few. This type of learning requires more inference than rote learning. The knowledge must be transformed into an operational form before stored in the knowledge base. Moreover the reliability of the source of knowledge should be considered.

Notes

Caution The system must guarantee that the new knowledge is differing with the offered knowledge.

Example: FOO (First Operational Operationaliser) is a learning system which is accessed to learn the game of Hearts. It transforms the advice which is in the form of principles, problems, and methods into efficient executable (LISP) procedures (or knowledge). Now this knowledge is prepared to apply.

Self Assessment

Fill in the blank:

5. The knowledge must be transformed into an ..... form before stored in the knowledge base.

12.4 Learning from Examples

As humans, we can learn by being informed. For intelligent agents, we are interested in learning by example, or else, we are back to programming yet again.

12.4.1 Induction

Classification is the procedure of allocating, to a specific input, the name of a class to which it associates. The classes from which the classification process can select can be illustrated in a variety of manners. Their definition will rely on the use to which they are put. Classification is an imperative component of many problem solving duties. Before classification can be achieved, the classes it will utilize must be defined:

- Separate a set of traits that are pertinent to the task domain. Define every class by a weighted sum of values of these traits.

Example: Task is weather prediction, the parameters can be measurements such as rainfall, location of cold fronts etc.

- Separate a set of traits that are pertinent to the task domain. Define every class as a structure composed of these traits. Ex: classifying animals, diverse traits can be such things as color, length of neck etc.

Did u know? The thought of generating a classification program that can develop its own class definitions is known as concept learning or induction.

Winston's Learning Program

- It is an early structural concept learning program.
- This program functions in a simple blocks world domain.
- Its objective was to build representations of the definitions of concepts in blocks domain.

Notes

Example: It learned the concepts House, Tent and Arch.

- A near miss is an object that is not a case of the concept in query but that is very alike to such instances.

Basic Approach of Winston's Program

1. Start with a structural description of one recognized instance of the concept. Call that description the concept definition.
2. Inspect descriptions of other recognized instances of the concepts. Generalize the definition to include them.
3. Inspect the descriptions of near misses of the concept. Restrict the definition to exclude these.

Version Spaces

- The objective of version spaces is to generate a description that is reliable with all positive examples but no negative examples in the training set.
- This is another strategy to concept learning.
- Version spaces function by sustaining a set of possible descriptions and evolving that set as new examples and near misses are displayed.
- The version space is just a set of descriptions, so an early idea is to maintain an explicit list of those descriptions.
- Version space includes two subsets of the concept space.
- One subset known as G includes most general descriptions reliable with the training examples The other subset includes the most particular descriptions reliable with the training examples.
- The algorithm for narrowing the version space is known as the Candidate elimination algorithm.

Algorithm: Candidate Elimination

- ***Given:*** A demonstration language and a set of positive and negative examples articulated in that language.
- ***Compute:*** A concept description that is reliable with all the positive examples and none of the negative examples.

1. Initialize G to enclose one element.
2. Initialize S to enclose one element: the first positive element.
3. Accept new training example. If it is a positive example, first remove from G any descriptions that do not cover the example. Then modernize the set S to enclose most particular set of descriptions in the version space that cover the example and the current elements of the S set. Inverse actions for negative example.
4. If S and G are both singleton sets, then if they are indistinguishable, output their values and halt.

Notes

Decision Trees

- This is a third strategy to concept learning.
- To categorize a specific input, we begin at the top of the tree and reply questions until we reach a leaf, where the specification is stored.
- ID3 is a program example for Decision Trees.
- ID3 utilizes iterative method to construct decision trees, favoring simple trees over complex ones, on the theory that simple trees are more precise classifiers of future inputs.
- It starts by selecting a random subset of the training examples.
- This subset is known as the window.
- The algorithm constructs a decision tree that accurately categorizes all examples in the window.

Task Illustrate the steps used in Winston's Program.

Self Assessment

Fill in the blanks:

6. The thought of generating a classification program that can develop its own class definitions is known as ..... .
7. The objective of ..... is to generate a description that is reliable with all positive examples but no negative examples in the training set.

12.5 Explanation-based Learning

An Explanation-based Learning (EBL) system accepts an example (i.e. a training example) and illustrates what it learns from the example. The EBL system takes only the pertinent features of the training. This clarification is converted into specific form that a problem solving program can understand. The explanation is generalized so that it can be utilized to solve other problems.

PRODIGY is a system that incorporates problem solving, planning, and learning methods in a single design. It was formerly envisioned by Jaime Carbonell and Steven Minton, as an AI system to test and build up ideas on the role that machine learning plays in planning and problem solving. PRODIGY utilizes the EBL to obtain control rules.

The EBL module utilizes the results from the problem-solving trace (i.e. Steps in solving problems) that were produced by the central problem solver (a search engine that searches over a problem space). It builds explanations by means of an axiomatized theory that illustrates both the domain and the architecture of the problem solver. The results are then converted as control rules and added to the knowledge base. The control knowledge that comprises control rules is utilized to direct the search process efficiently.

Notes

Self Assessment

Fill in the blanks:

8. An ..... system accepts an example (i.e. a training example) and illustrates what it learns from the example.

9. ..... is a system that incorporates problem solving, planning, and learning methods in a single design.

12.6 Learning by Parameter Adjustment

Here the learning system depends on evaluation procedure that merges information from numerous sources into a single summary static.

Example: The factors like demand and production capacity may be merged into a single score to signify the likelihood for increase of production. But it is hard to know a priori how much weight should be associated to each factor.

The accurate weight can be located by taking some approximation of the correct settings and then permit the program alter its settings based on its experience. This type of learning systems is functional when little knowledge is obtainable.

Example: In game programs, the factors like piece benefit and mobility are merged into a score to decide whether a specific board position is desirable. This single score is nothing but a knowledge which the program collected through calculation.

Self Assessment

Fill in the blank:

10. In case of learning by....., learning system depends on evaluation procedure that merges information from numerous sources into a single summary static.

12.7 Learning with Macro-operators

Series of actions that can be considered as a whole are known as macro-operators. Once a problem is solved, the learning component takes the calculated plan and accumulates it as a macro-operator. The requirements are the initial conditions of the problem just solved, and its post conditions matches to the goal just attained.

The problem solver competently accesses the knowledge base it gained from its preceding experiences. By generalizing macro-operators the problem solver can even crack diverse problems.

Caution Generalization is performed by substituting all the constants in the macro-operators with variables.

Example: The STRIPS, is a planning algorithm that engaged macro-operators in it's learning segment. It constructs a macro operator MACROP, that comprises preconditions,

Notes

post-conditions and the series of actions. The macro operator will be accessed in the upcoming operation.

Self Assessment

Fill in the blanks:

11. Series of actions that can be considered as a whole are known as ..... .
12. Generalization is performed by substituting all the ..... in the macro-operators with variables.

12.8 Learning by Chunking

Chunking is comparable to learning by means of macro-operators. Usually, it is accessed by problem solver systems that utilize production systems.

A production system comprise of a set of rules that are in if-then outline. That is specified a specific situation, what are the actions to be achieved.

Example: If it is raining then take umbrella.

Production system also comprises knowledge base, control strategy and a rule applier. To solve a problem, a system will contrast the present condition with the left hand side of the rules. If there is a match then the system will achieve the actions illustrated in the right hand side of the corresponding rule.

Problem solvers explain problems by applying the rules. Some of these rules may be more functional than others and the results are accumulated as a chunk. Chunking can be utilized to study common search control knowledge. Numerous chunks may encode a single macro-operator and one chunk may contribute in a number of macro sequences. Chunks learned in the starting of problem solving, may be used in the afterward stage. The system maintains the chunk to utilize it in solving other problems.

Soar is a common cognitive architecture for producing intelligent systems. Soar needs knowledge to solve several problems. It obtains knowledge by means of chunking mechanism. The system learns reflexively when impasses have been resolved. An impasse occurs when the system does not have enough knowledge. Thus, Soar selects a new problem space (set of states and the operators that influence the states) in an offer to resolve the standoff. While resolving the impasse, the individual steps of the task plan are collected into bigger steps called chunks. The chunks reduce the problem space search and so augment the efficiency of achieving the task.

In Soar, the knowledge is accumulated in long-term memory. Soar utilizes the chunking mechanism to generate productions that are accumulated in long-term memory. A chunk is nothing but a great production that does the work of the whole sequence of smaller ones. The productions have a set of conditions or patterns to be matched to functioning memory which comprise of existing goals, problem spaces, states and operators and a set of actions to carry out when the production fires. Chunks are generalized before accumulating. When the same impasse appears again, the chunks so gathered can be utilized to resolve it.

Self Assessment

Fill in the blanks:

13. A ..... system comprise of a set of rules that are in if-then outline.

Notes

14. A ..... is nothing but a great production that does the work of the whole sequence of smaller ones.

12.9 Discovery-based Learning

Discovery is a limited form of learning. The knowledge acquisition is performed without obtaining any support from an educator. Discovery Learning is an inquiry-dependent learning method.

In discovery learning, the learner utilizes his own experience and prior knowledge to learn the truths that are to be studied. The learner builds his own knowledge by researching with a domain, and inferring rules from the outcomes of these experiments. In addition to domain information the learner require some support in selecting and interpreting the information to construct his knowledge base.

A cluster is a compilation of objects which are alike in some manner. Clustering groups data items into resemblance classes. The properties of these classes can then be utilized to understand problem traits or to discover similar groups of data items. Clustering can be defined as the process of decreasing a huge set of unlabeled data to controllable piles involving similar items. The similarity measures rely on the assumptions and preferred handling one brings to the data.

Clustering starts by doing trait withdrawal on data items and compute the values of the selected feature set. Then the clustering model chooses and compares two sets of data items and outputs the similarity measure among them. Clustering algorithms that utilize specific similarity measures as subroutines are engaged to construct clusters.

The clustering algorithms are usually classified as Exclusive Clustering, Overlapping Clustering, Hierarchical Clustering and Probabilistic Clustering. The collection of clustering algorithms depends on different criteria like time and space complexity. The outcomes are verified to observe if they meet the standard or else some or all of the above steps have to be frequent.

Some applications of clustering are data compression, hypothesis generation and hypothesis testing. The theoretical clustering system accepts a set of object descriptions in the form of events, observations, facts and then generates a classification method over the observations.

Notes COBWEB is an incremental abstract clustering system. It incrementally adds the objects into a categorization tree. The striking trait of incremental systems is that the knowledge is modernized with every new observation. In COBWEB system, learning is incremental and the knowledge it learned in the form of classification trees raise the inference capabilities.

Task Illustrate the concept of clustering system.

Self Assessment

Fill in the blanks:

15. In ..... learning, the learner utilizes his own experience and prior knowledge to learn the truths that are to be studied.

Notes

16. ..... starts by doing trait withdrawal on data items and compute the values of the selected feature set.

12.10 Learning by Correcting Mistakes

When learning new things, there is a likelihood that the learning system may make errors. Similar to human beings, learning system can correct itself by determining reasons for its breakdown, isolate it, elucidate how the particular supposition causes failure, and alters its knowledge base.

Example: When playing chess a learning system may make an incorrect move and finishes up with failure. Now the learning system considers the reasons for the breakdown and corrects its knowledge base. So when it plays again it will not replicate the similar mistake.

In his work, Active Learning with Multiple Views, Ion Muslea has utilized this technique to label the data. He produces a technique called Co-EMT which is a amalgamation of two techniques: Co-testing and Co-EM. The Co-testing technique communicates with the user to label the data. If it does any mistake in labeling, it learns from the mistakes and enhances. After learning, the system labels the unlabeled data removed from a source proficiently.

Did u know? The labeled data comprises what is known as knowledge.

Self Assessment

Fill in the blank:

17. When learning new things, there is a likelihood that the learning system may make ..... .

12.11 Learning by Recording Cases

A program that learns by recording cases usually use constancy heuristic. As per constancy heuristic, a property of something can be estimated by locating the most similar cases from a specified set of cases.

Example: A computer is given the images of different types of insects, birds, and animals. If the computer is asked to recognize a living thing which is not in the recorded list, it will contrast the specified image with already recorded ones, and will at least tell whether the specified image is insect, bird or animal.

Learning by recoding cases method is chiefly used in natural language learning tasks.

For the period of the training phase, a set of cases that illustrate vagueness resolution episodes for a specific problem in text analysis is gathered. Every case includes a set of traits or attribute-value pairs that instruct the context in which the vagueness was encountered.

Furthermore, every case is annotated with solution traits that enlighten how the ambiguity was resolved in the present example. The cases which are formed are then accumulated in a case base. Once the training is over, the system can make use of the case base to resolve ambiguities in new sentences. This manner, the system obtains the linguistic knowledge.

Notes

Self Assessment

Fill in the blank:

18. Learning by recoding cases method is chiefly used in ..... learning tasks.

12.12 Learning by Managing Multiple Models

Learning can be performed by handling a set of common models and a set of particular models kept in a version space. A version space is nothing but a demonstration that is used to get pertinent information from a set of learning examples.

A version space portrayal includes two trees which harmonize to each other: one symbolize general model and the other symbolize specific model. Both positive and negative examples are used to get the two set of models converge on one just-right model. That is, positive examples simplify specific models and negative examples specialize common models. Eventually, a correct model that matches only the observed positive examples is attained. Query By Committee (QBC) is an algorithm which executes this technique so as to obtain knowledge.

Self Assessment

Fill in the blanks:

19. That is, ... examples simplify specific models and negative examples specialize common models.
20. A version space portrayal includes two..... which harmonize to each other: one symbolize general model and the other symbolize specific model.

12.13 Summary

- Learning is distinguished into a number of different forms. The simplest is learning by trial-and-error.
- Rote learning is the fundamental learning activity which is also known as memorization since the knowledge, without any alteration is, just copied into the knowledge base.
- Learning by taking advice is a straightforward form of learning where knowledge must be transformed into an operational form before stored in the knowledge base.
- Classification is the procedure of allocating, to a specific input, the name of a class to which it associates.
- The thought of generating a classification program that can develop its own class definitions is known as concept learning or induction.
- Winston's Learning Program is an early structural concept learning program which functions in a simple blocks world domain.
- The objective of version spaces is to generate a description that is reliable with all positive examples but no negative examples in the training set.
- In case of decision tree, to categorize a specific input, we begin at the top of the tree and reply questions until we reach a leaf, where the specification is stored.
- An Explanation-based Learning (EBL) system accepts an example (i.e. a training example) and illustrates what it learns from the example.

Notes

- In discovery learning, the learner utilizes his own experience and prior knowledge to learn the truths that are to be studied.

12.14 Keywords

Classification: It is the procedure of allocating, to a specific input, the name of a class to which it associates.

Explanation-based Learning (EBL): An Explanation-based Learning (EBL) system accepts an example (i.e. a training example) and illustrates what it learns from the example.

Induction: The thought of generating a classification program that can develop its own class definitions is known as concept learning or induction.

Learning: Learning is defined as modifications in the system that are adaptive in the sense that they facilitate the system to do the similar task or tasks taken from the similar population more competently and more successfully the next time.

Rote Learning: Rote learning is the fundamental learning activity which is also known as memorization since the knowledge, without any alteration is, just copied into the knowledge base.

Winston's Learning Program: It is an early structural concept learning program which functions in a simple blocks world domain.

12.15 Review Questions

1. What is rote learning? Illustrate with example.
2. Enlighten some issues that are relevant to more complex learning issues.
3. Discuss the process of Learning by taking Advice. Give example.
4. Explicate the process of induction with example.
5. What is Winston's Learning Program? Illustrate the basic approach of Winston's Program.
6. What is the main goal of version spaces? Explain with the help of algorithm.
7. To categorize a specific input, we begin at the top of the tree and reply questions until we reach a leaf, where the specification is stored. Comment.
8. Describe the concept of Explanation-based Learning (EBL).
9. Make distinction between learning with macro-operators and learning by chunking.
10. What is a cluster? Illustrate the process of beginning clustering.

Answers: Self Assessment

1.	Learning	2.	Generalization
3.	Rote Learning	4.	Memorization
5.	Operational	6.	Concept Learning or Induction
7.	Version Spaces	8.	Explanation-based Learning (EBL)
9.	PRODIGY	10.	Parameter Adjustment
11.	Macro-operators	12.	Constants

Notes

13.	Production	14.	Chunk
15.	Discovery	16.	Clustering
17.	Errors	18.	Natural Language
19.	Positive	20.	Trees

12.16 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Notes

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.springer.com

Notes

Unit 13: Expert Systems and its Architecture

CONTENTS

Objectives

Introduction

13.1 Part of Expert System

13.1.1 Main ES Components

13.2 General Concepts and Characteristics of ES

13.2.1 Chaining

13.2.2 Certainty Factors

13.3 Expert System Architecture

13.3.1 End User

13.3.2 Explanation System (Another Part of Expert System)

13.3.3 Expert Systems versus Problem-solving Systems

13.3.4 Procedure Node Interface

13.3.5 User Interface

13.4 Application of Expert System

13.5 Advantages and Disadvantages

13.5.1 Advantages

13.5.2 Disadvantages

13.6 Types of Problems Solved by Expert Systems

13.7 ES-Shells

13.7.1 Shells

13.7.2 Expert System Programming Environments

13.7.3 Advantages and Disadvantages of Expert System Shells

13.8 Dealing with Uncertainty

13.8.1 Domain Knowledge Representation

13.8.2 User Knowledge Representation

13.8.3 Knowledge Determination and Updating

13.8.4 Model Initialization

13.8.5 Model Updating

13.8.6 Knowledge Value Propagation

13.8.7 Qualitative and Quantitative Analysis

Contd...

13.9 Speech Recognition
13.10 Summary
13.11 Keywords
13.12 Review Questions
13.13 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Understand the concept of expert system
- Discuss the components of expert system
- Illustrate the characteristics of ES
- Understand the expert system architecture
- Discuss the applications of expert system
- Explain the concept of ES shells
- Understand the concept of dealing with uncertainty

Introduction

An expert system is software that attempts to reproduce the performance of one or more human experts, most commonly in a specific problem domain, and is a traditional application and/or subfield of artificial intelligence. A wide variety of methods can be used to simulate the performance of the expert however common to most or all are (1) the creation of a so-called "knowledge base" which uses some knowledge representation formalism to capture the Subject Matter Experts (SME) knowledge and (2) a process of gathering that knowledge from the SME and codifying it according to the formalism, which is called knowledge engineering. Expert systems may or may not have learning components but a third common element is that once the system is developed it is proven by being placed in the same real world problem solving situation as the human SME, typically as an aid to human workers or a supplement to some information system. As a premiere application of computing and artificial intelligence, the topic of expert systems has many points of contact with general systems theory, operations research, business process reengineering and various topics in applied mathematics and management science.

13.1 Part of Expert System

13.1.1 Main ES Components

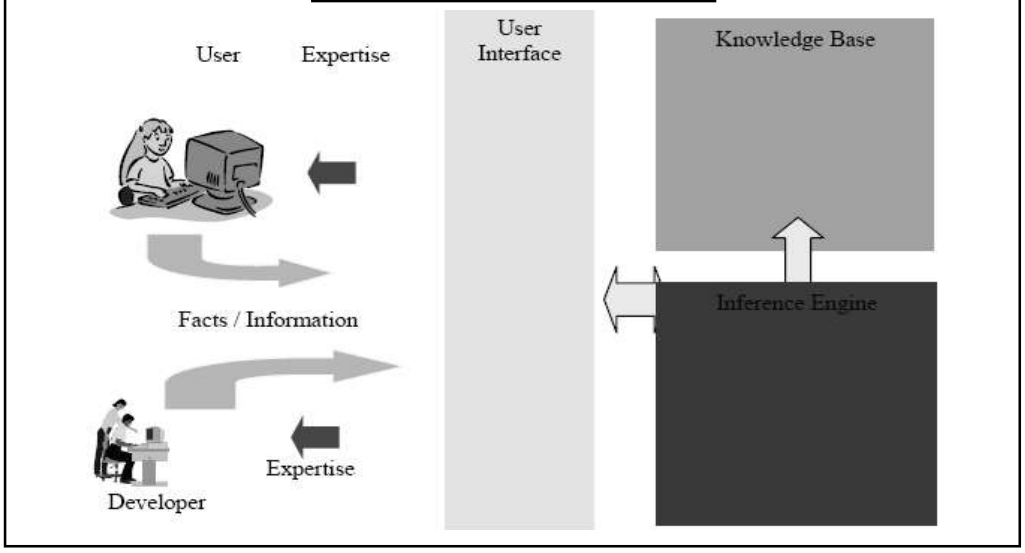
- *Knowledge base*
  - contains essential information about the problem domain
  - often represented as facts and rules
- *Inference engine*
  - mechanism to derive new knowledge from the knowledge base and the

Notes

- information provided by the user
- often based on the use of rules

- *User interface*
  - interaction with end users
  - development and maintenance of the knowledge base

Figure 13.1: Parts of Expert System



Self Assessment

Fill in the blanks:

1. An ..... is software that attempts to reproduce the performance of one or more human experts, most commonly in a specific problem domain, and is a traditional application and/or subfield of artificial intelligence.
2. Knowledge base contains essential information about the problem ..... .

13.2 General Concepts and Characteristics of ES

There are the following parts of Expert System:

13.2.1 Chaining

There are two main methods of reasoning when using inference rules: backward chaining and forward chaining. Forward chaining starts with the data available and uses the inference rules to conclude more data until a desired goal is reached. An inference engine using forward chaining searches the inference rules until it finds one in which the if-clause is known to be true. It then concludes the then-clause and adds this information to its data. It would continue to do this until a goal is reached. Because the data available determines which inference rules are used, this method is also called data driven. Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a

Notes

then-clause that matches a desired goal. If the if-clause of that inference rule is not known to be true, then it is added to the list of goals.

Example: Suppose a rulebase contains:

(1) If Fritz is green then Fritz is a frog.

(2) If Fritz is a frog then Fritz hops.

Suppose a goal is to conclude that Fritz hops. The rulebase would be searched and rule (2) would be selected because its conclusion (the then clause) matches the goal. It is not known that Fritz is a frog, so this "if" statement is added to the goal list. The rulebase is again searched and this time rule (1) is selected because its then clause matches the new goal just added to the list. This time, the if-clause (Fritz is green) is known to be true and the goal that Fritz hops is concluded. Because the list of goals determines which rules are selected and used, this method is called goal driven.

Task Make distinction between backward chaining and forward chaining.

13.2.2 Certainty Factors

One advantage of expert systems over traditional methods of programming is that they allow the use of "confidences" (or "certainty factors"). When a human reasons he does not always conclude things with 100% confidence. He might say, "If Fritz is green, then he is probably a frog" (after all, he might be a chameleon). This type of reasoning can be imitated by using numeric values called confidences.

Example: If it is known that Fritz is green, it might be concluded with 0.85 confidence that he is a frog; or, if it is known that he is a frog, it might be concluded with 0.95 confidence that he hops. These numbers are similar in nature to probabilities, but they are not the same. They are meant to imitate the confidences humans use in reasoning rather than to follow the mathematical definitions used in calculating probabilities.

Self Assessment

Fill in the blanks:

3. ..... chaining starts with the data available and uses the inference rules to conclude more data until a desired goal is reached.

4. ..... chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals.

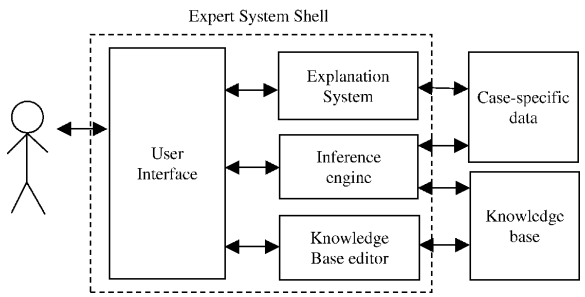
13.3 Expert System Architecture

Figure 13.2 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

Notes

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.

Figure 13.2: Important Module of ES



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.

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 13.2, 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.

Did u know? Expert systems work in industry includes both writing expert system shells and writing expert systems using shells.

Notes Using shells to write expert systems generally greatly reduces the cost and time of development (compared with writing the expert system from scratch).

The following general points about expert systems and their architecture have been illustrated:

1. The sequence of steps taken to reach a conclusion is dynamically synthesized with each new case. It is not explicitly programmed when the system is built.
2. Expert systems can process multiple values for any problem parameter. This permits more than one line of reasoning to be pursued and the results of incomplete (not fully determined) reasoning to be presented.
3. Problem solving is accomplished by applying specific knowledge rather than specific technique. This is a key idea in expert systems technology. It reflects the belief that human experts do not process their knowledge differently from others, but they do possess different knowledge. With this philosophy, when one finds that their expert system does not produce the desired results, work begins to expand the knowledge base, not to reprogram the procedures.

Notes

There are various expert systems in which a rulebase and an inference engine cooperate to simulate the reasoning process that a human expert pursues in analyzing a problem and arriving at a conclusion. In these systems, in order to simulate the human reasoning process, a vast amount of knowledge needed to be stored in the knowledge base. Generally, the knowledge base of such an expert system consisted of a relatively large number of "if then" type of statements that were interrelated in a manner that, in theory at least, resembled the sequence of mental steps that were involved in the human reasoning process.

Because of the need for large storage capacities and related programs to store the rulebase, most expert systems have, in the past, been run only on large information handling systems. Recently, the storage capacity of personal computers has increased to a point where it is becoming possible to consider running some types of simple expert systems on personal computers.

In some applications of expert systems, the nature of the application and the amount of stored information necessary to simulate the human reasoning process for that application is just too vast to store in the active memory of a computer. In other applications of expert systems, the nature of the application is such that not all of the information is always needed in the reasoning process.

Example: An example of this latter type application would be the use of an expert system to diagnose a data processing system comprising many separate components, some of which are optional.

When that type of expert system employs a single integrated rulebase to diagnose the minimum system configuration of the data processing system, much of the rulebase is not required since many of the components which are optional units of the system will not be present in the system. Nevertheless, earlier expert systems require the entire rulebase to be stored since all the rules were, in effect, chained or linked together by the structure of the rulebase.

When the rulebase is segmented, preferably into contextual segments or units, it is then possible to eliminate portions of the Rulebase containing data or knowledge that is not needed in a particular application. The segmenting of the rulebase also allows the expert system to be run with systems or on systems having much smaller memory capacities than was possible with earlier arrangements since each segment of the rulebase can be paged into and out of the system as needed. The segmenting of the rulebase into contextual segments requires that the expert system manage various intersegment relationships as segments are paged into and out of memory during execution of the program. Since the system permits a rulebase segment to be called and executed at any time during the processing of the first rulebase, provision must be made to store the data that has been accumulated up to that point so that at some time later in the process, when the system returns to the first segment, it can proceed from the last point or rule node that was processed. Also, provision must be made so that data that has been collected by the system up to that point can be passed to the second segment of the rulebase after it has been paged into the system and data collected during the processing of the second segment can be passed to the first segment when the system returns to complete processing that segment. The user interface and the procedure interface are two important functions in the information collection process.

13.3.1 End User

The end-user usually sees an expert system through an interactive dialog,

Examples:

Q. Do you know which restaurant you want to go to?

Notes

A. No

Q. Is there any kind of food you would particularly like?

A. No

Q. Do you like spicy food?

A. No

Q. Do you usually drink wine with meals?

A. Yes

Q. When you drink wine, is it French wine?

A. Yes

As can be seen from this dialog, the system is leading the user through a set of questions, the purpose of which is to determine a suitable set of restaurants to recommend. This dialog begins with the system asking if the user already knows the restaurant choice (a common feature of expert systems) and immediately illustrates a characteristic of expert systems; users may choose not to respond to any question.

In expert systems, dialogs are not pre-planned. There is no fixed control structure. Dialogs are synthesized from the current information and the contents of the knowledge base. Because of this, not being able to supply the answer to a particular question does not stop the consultation.

13.3.2 Explanation System (Another Part of Expert System)

Another major distinction between expert systems and traditional systems is illustrated by the following answer given by the system when the user answers a question with another question, "Why", as occurred in the above example. The answer is:

A. I am trying to determine the type of restaurant to suggest. So far Chinese is not a likely choice. It is possible that French is a likely choice. I know that if the diner is a wine drinker, and the preferred wine is French, then there is strong evidence that the restaurant choice should include French. It is very difficult to implement a general explanation system (answering questions like "Why" and "How") in a traditional computer program. An expert system can generate an explanation by retracing the steps of its reasoning. The response of the expert system to the question WHY is an exposure of the underlying knowledge structure. It is a rule; a set of antecedent conditions which, if true, allow the assertion of a consequent. The rule references values, and tests them against various constraints or asserts constraints onto them. This, in fact, is a significant part of the knowledge structure. There are values, which may be associated with some organizing entity.

Example: The individual diner is an entity with various attributes (values) including whether they drink wine and the kind of wine.

There are also rules, which associate the currently known values of some attributes with assertions that can be made about other attributes. It is the orderly processing of these rules that dictates the dialog itself.

13.3.3 Expert Systems versus Problem-solving Systems

The principal distinction between expert systems and traditional problem solving programs is the way in which the problem related expertise is coded. In traditional applications, problem

Notes

expertise is encoded in both program and data structures. In the expert system approach all of the problem related expertise is encoded in data structures only; none is in programs. This organization has several benefits.

An example may help contrast the traditional problem solving program with the expert system approach.

Example: The example is the problem of tax advice. In the traditional approach data structures describe the taxpayer and tax tables, and a program in which there are statements representing an expert tax consultant's knowledge, such as statements which relate information about the taxpayer to tax table choices. It is this representation of the tax expert's knowledge that is difficult for the tax expert to understand or modify. In the expert system approach, the information about taxpayers and tax computations is again found in data structures, but now the knowledge describing the relationships between them is encoded in data structures as well. The programs of an expert system are independent of the problem domain (taxes) and serve to process the data structures without regard to the nature of the problem area they describe. For example, there are programs to acquire the described data values through user interaction, programs to represent and process special organizations of description, and programs to process the declarations that represent semantic relationships within the problem domain and an algorithm to control the processing sequence and focus.

The general architecture of an expert system involves two principal components: a problem dependent set of data declarations called the knowledge base or rule base, and a problem independent (although highly data structure dependent) program which is called the inference engine.

Individuals Involved with Expert Systems

There are generally three individuals having an interaction with expert systems. Primary among these is the end-user; the individual who uses the system for its problem solving assistance. In the building and maintenance of the system there are two other roles: the problem domain expert who builds and supplies the knowledge base providing the domain expertise, and a knowledge engineer who assists the experts in determining the representation of their knowledge, enters this knowledge into an explanation module and who defines the inference technique required to obtain useful problem solving activity. Usually, the knowledge engineer will represent the problem solving activity in the form of rules which is referred to as a rule-based expert system. When these rules are created from the domain expertise, the knowledge base stores the rules of the expert system.

Inference Rule

An understanding of the "inference rule" concept is important to understand expert systems. An inference rule is a statement that has two parts, an if-clause and a then-clause. This rule is what gives expert systems the ability to find solutions to diagnostic and prescriptive problems.

Example: An example of an inference rule is: If the restaurant choice includes French, and the occasion is romantic, then the restaurant choice is definitely Paul Bocuse.

An expert system's rulebase is made up of many such inference rules. They are entered as separate rules and it is the inference engine that uses them together to draw conclusions. Because each rule is a unit, rules may be deleted or added without affecting other rules (though it should affect which conclusions are reached). One advantage of inference rules over traditional programming is that inference rules use reasoning which more closely resemble human

Notes

reasoning. Thus, when a conclusion is drawn, it is possible to understand how this conclusion was reached. Furthermore, because the expert system uses knowledge in a form similar to the expert, it may be easier to retrieve this information from the expert.

13.3.4 Procedure Node Interface

The function of the procedure node interface is to receive information from the procedures coordinator and create the appropriate procedure call. The ability to call a procedure and receive information from that procedure can be viewed as simply a generalization of input from the external world. While in some earlier expert system external information has been obtained, that information was obtained only in a predetermined manner so only certain information could actually be acquired. This expert system, disclosed in the cross-referenced application, through the knowledge base, is permitted to invoke any procedure allowed on its host system. This makes the expert system useful in a much wider class of knowledge domains than if it had no external access or only limited external access.

In the area of machine diagnostics using expert systems, particularly self-diagnostic applications, it is not possible to conclude the current state of "health" of a machine without some information. The best source of information is the machine itself, for it contains much detailed information that could not reasonably be provided by the operator. The knowledge that is represented in the system appears in the rulebase. In the rulebase described in the cross-referenced applications, there are basically four different types of objects, with associated information present.

1. ***Classes***: These are questions asked to the user.
2. ***Parameters:*** A parameter is a place holder for a character string which may be a variable that can be inserted into a class question at the point in the question where the parameter is positioned.
3. ***Procedures:*** These are definitions of calls to external procedures.
4. ***Rule Nodes:*** The inferencing in the system is done by a tree structure which indicates the rules or logic which mimics human reasoning. The nodes of these trees are called rule nodes. There are several different types of rule nodes.

The rulebase comprises a forest of many trees. The top node of the tree is called the goal node, in that it contains the conclusion. Each tree in the forest has a different goal node. The leaves of the tree are also referred to as rule nodes, or one of the types of rule nodes. A leaf may be an evidence node, an external node, or a reference node.

An evidence node functions to obtain information from the operator by asking a specific question. In responding to a question presented by an evidence node, the operator is generally instructed to answer "yes" or "no" represented by numeric values 1 and 0 or provide a value of between 0 and 1, represented by a "maybe." Questions which require a response from the operator other than yes or no or a value between 0 and 1 are handled in a different manner. A leaf that is an external node indicates that data will be used which was obtained from a procedure call.

A reference node functions to refer to another tree or subtree. A tree may also contain intermediate or minor nodes between the goal node and the leaf node. An intermediate node can represent logical operations like And or Or. The inference logic has two functions. It selects a tree to trace and then it traces that tree. Once a tree has been selected, that tree is traced, depth-first, left to right. The word "tracing" refers to the action the system takes as it traverses the tree, asking classes (questions), calling procedures, and calculating confidences as it proceeds.

As explained in the cross-referenced applications, the selection of a tree depends on the ordering of the trees. The original ordering of the trees is the order in which they appear in the rulebase. This order can be changed, however, by assigning an evidence node an attribute "initial" which

Notes

is described in detail in these applications. The first action taken is to obtain values for all evidence nodes which have been assigned an "initial" attribute. Using only the answers to these initial evidences, the rules are ordered so that the most likely to succeed is evaluated first. The trees can be further reordered since they are constantly being updated as a selected tree is being traced.

It has been found that the type of information that is solicited by the system from the user by means of questions or classes should be tailored to the level of knowledge of the user. In many applications, the group of prospective uses is nicely defined and the knowledge level can be estimated so that the questions can be presented at a level which corresponds generally to the average user. However, in other applications, knowledge of the specific domain of the expert system might vary considerably among the group of prospective users.

One application where this is particularly true involves the use of an expert system, operating in a self-diagnostic mode on a personal computer to assist the operator of the personal computer to diagnose the cause of a fault or error in either the hardware or software. In general, asking the operator for information is the most straightforward way for the expert system to gather information assuming, of course, that the information is or should be within the operator's understanding.

Example: In diagnosing a personal computer, the expert system must know the major functional components of the system. It could ask the operator, for instance, if the display is a monochrome or color display. The operator should, in all probability, be able to provide the correct answer 100% of the time. The expert system could, on the other hand, cause a test unit to be run to determine the type of display. The accuracy of the data collected by either approach in this instance probably would not be that different so the knowledge engineer could employ either approach without affecting the accuracy of the diagnosis.

However, in many instances, because of the nature of the information being solicited, it is better to obtain the information from the system rather than asking the operator, because the accuracy of the data supplied by the operator is so low that the system could not effectively process it to a meaningful conclusion.

In many situations the information is already in the system, in a form of which permits the correct answer to a question to be obtained through a process of inductive or deductive reasoning. The data previously collected by the system could be answers provided by the user to less complex questions that were asked for a different reason or results returned from test units that were previously run.

Task Illustrate the concept of Procedure Node Interface.

13.3.5 User Interface

The function of the user interface is to present questions and information to the user and supply the user's responses to the inference engine. Some responses are restricted to a set of possible legal answers, others are not. The user interface checks all responses to insure that they are of the correct data type. Any responses that are restricted to a legal set of answers are compared against these legal answers. Whenever the user enters an illegal answer, the user interface informs the user that his answer was invalid and prompts him to correct it.

Notes

Caution Any values entered by the user must be received and interpreted by the user interface.

Self Assessment

Fill in the blanks:

5. The ..... usually sees an expert system through an interactive dialog.
6. An ..... is a statement that has two parts, an if-clause and a then-clause.

13.4 Application of Expert System

Expert systems are designed and created to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, human resources etc. Indeed, the foundation of a successful expert system depends on a series of technical procedures and development that may be designed by certain technicians and related experts.

Example: A good example of application of expert systems in banking area is expert systems for mortgages. Loan departments are interested in expert systems for mortgages because of the growing cost of labour which makes the handling and acceptance of relatively small loans less profitable.

They also see in the application of expert systems a possibility for standardized, efficient handling of mortgage loan, and appreciate that for the acceptance of mortgages there are hard and fast rules which do not always exist with other types of loans. While expert systems have distinguished themselves in AI research in finding practical application, their application has been limited. Expert systems are notoriously narrow in their domain of knowledge – as an amusing example, a researcher used the "skin disease" expert system to diagnose his rust bucket car as likely to have developed measles and the systems were thus prone to making errors that humans would easily spot. Additionally, once some of the mystique had worn off, most programmers realized that simple expert systems were essentially just slightly more elaborate versions of the decision logic they had already been using.

Therefore, some of the techniques of expert systems can now be found in most complex programs without any fuss about them.

Example: An example, and a good demonstration of the limitations of, an expert system used by many people is the Microsoft Windows operating system troubleshooting software located in the "help" section in the taskbar menu.

Obtaining expert/technical operating system support is often difficult for individuals not closely involved with the development of the operating system. Microsoft has designed their expert system to provide solutions, advice, and suggestions to common errors encountered throughout using the operating systems.

Example: For example, the computer baseball games Earl Weaver Baseball and Tony La Russa Baseball each had highly detailed simulations of the game strategies of those two baseball managers. When a human played the game against the computer, the computer queried the Earl Weaver or Tony La Russa Expert System for a decision on what strategy to follow. Even those

Notes

choices where some randomness was part of the natural system (such as when to throw a surprise pitch-out to try to trick a runner trying to steal a base) were decided based on probabilities supplied by Weaver or La Russa.

Today we would simply say that "the game's AI provided the opposing manager's strategy."

Self Assessment

Fill in the blank:

7. Microsoft has designed their expert system to provide solutions, advice, and suggestions to common errors encountered throughout using the ..... .

13.5 Advantages and Disadvantages

13.5.1 Advantages

- Provides consistent answers for repetitive decisions, processes and tasks
- Holds and maintains significant levels of information
- Encourages organizations to clarify the logic of their decision-making
- Never "forgets" to ask a question, as a human might

13.5.2 Disadvantages

- Lacks common sense needed in some decision making
- Cannot make creative responses as human expert would in unusual circumstances
- Domain experts not always able to explain their logic and reasoning
- Errors may occur in the knowledge base, and lead to wrong decisions
- Cannot adapt to changing environments, unless knowledge base is changed

Self Assessment

Fill in the blanks:

8. Expert system encourages organizations to clarify the logic of their ..... .
9. Expert System lacks ..... needed in some decision making.

13.6 Types of Problems Solved by Expert Systems

Expert systems are most valuable to organizations that have a high-level of know-how experience and expertise that cannot be easily transferred to other members. They are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem-solving purposes. Typically, the problems to be solved are of the sort that would normally be tackled by a medical or other professional. Real experts in the problem domain (which will typically be very narrow, for instance "diagnosing skin conditions in human teenagers") are asked to provide "rules of thumb" on how they evaluate the problems, either explicitly with the aid of experienced systems developers, or sometimes implicitly, by getting such experts to evaluate test cases and using computer programs

Notes

to examine the test data and (in a strictly limited manner) derive rules from that. Generally, expert systems are used for problems for which there is no single "correct" solution which can be encoded in a conventional algorithm. One would not write an expert system to find shortest paths through graphs, or sort data, as there are simply easier ways to do these tasks. Simple systems use simple true/false logic to evaluate data. More sophisticated systems are capable of performing at least some evaluation, taking into account real-world uncertainties, using such methods as fuzzy logic. Such sophistication is difficult to develop and still highly imperfect.

Self Assessment

Fill in the blanks:

10. Typically, the ..... to be solved are of the sort that would normally be tackled by a medical or other professional.
11. Generally, expert systems are used for problems for which there is no single "correct" solution which can be encoded in a ..... algorithm.

13.7 ES-Shells

A shell is a complete development environment for building and maintaining knowledge-based applications. It provides a step-by-step methodology for a knowledge engineer that allows the domain experts themselves to be directly involved in structuring and encoding the knowledge. Many commercial shells are available. The **ES shell** is a command line interpreter developed by Byron Rakitzis and Paul Haahr, that uses a scripting language similar to the RC shell of the Plan 9 operating system. It is intended to provide a fully functional programming language as a Unix shell. The bulk of es' development occurred in the early 1990s. A paper on an early version of the ES shell was presented at the Winter 1993 USENIX conference in San Diego. A patched version of es-0.9-beta1 which includes job control features, a precompiled binary, additional documentation, a basic emacs editing mode and other contributed programs is available: es-0.9-beta1job-control.tar.bz2. Expert Systems Shells include Software Development Packages such as:

- Corvd (Exsys) InstantTea
- K-Vision
- KnowledgePro

If one wishes to build an expert system, one has several choices of software tool:

(1) Conventional programming languages (e.g. Pascal, C++, Java)

(2) Artificial intelligence programming languages (particularly LISP and Prolog)

(3) Expert system shells

- The first choice is almost certainly a bad idea.
- Conventional programming languages are not designed for this sort of job, and too much work is required to make the program perform in the way required.
- However, if it is important to have highly efficient software, this might be a suitable choice.
- Choice (2) has the advantage:

A flexible system can be built, accurately reflecting the peculiarities of the knowledge domain and system task.

Notes

Choice (2) has the disadvantages: Programming skills in these languages are not common. It may be necessary to hire specialist programmers, or retrain the programming staff.

Programming the system will always be a larger (and hence longer and more expensive) task than using a shell. Choice (3) has been the most frequent choice for commercial systems in recent years. You will remember that an expert system shell is a ready-made expert system, with the knowledge base missing, together with instructions for building a knowledge base in the customer's chosen domain.

Figure 13.3: Idea of ES Shell

Figure 13.4: Idea of ES Shell 2

13.7.1 Shells

Some organisations avoid using shells for building complete expert systems; but even they frequently use them for:

- Training
- Building prototypes

13.7.2 Expert System Programming Environments

1. Some people make a distinction between ES shells and ES programming environments (or "hybrid systems"). For instance, Efraim Turban does in his book (Turban, 1992).
2. Historically, this has been important because, in the 1980s, most expert systems projects in the UK used shells, and most expert systems projects in the USA used environments.
3. Environments were so called because they provided several different forms of knowledge representation, for instance,
   (a) rules
   (b) metarules
   (c) frames
   (d) semantic nets
4. Several different forms of inference, e.g.
   (a) forward chaining
   (b) backward chaining
   (c) bidirectional chaining
   (d) non-monotonic reasoning

Notes

5. They needed more powerful hardware than a microcomputer – usually, a workstation. Historically, ES shells have been more constrained, perhaps offering only a single kind of knowledge representation. They would usually be designed to run on a PC.
6. However, in recent years, ES shells have become more sophisticated, and added multiple forms of knowledge representation and of inference strategy. PCs have become more powerful, and PC versions of ES environment software have been released.

Notes It is probably not useful to make the distinction any more. One could simply speak of "simple shells" and "sophisticated shells".

13.7.3 Advantages and Disadvantages of Expert System Shells

Advantages

The programming effort that has gone into building the user interface and inference engine is reused:

1. The level of programming skill needed to produce the finished system is much lower than it would be if the system was programmed from scratch using a language.
2. This means that, if an appropriate shell is chosen, the project can be completed faster, and cheaper.

Disadvantages

1. ES tools are "end-user tools". Compared with systems programmed from a language, such software packages tend to produce systems that have:
   (a) poor documentation
   (b) weak security
   (c) difficult maintenance problems
2. If the shell is a poor match for the type of knowledge in the domain concerned, it is liable to produce a system which simply doesn't correspond to the expertise of the original domain expert.
3. In an attempt to model a non-standard piece of reasoning, the system builders by side. Such a "system" is bound to be unsatisfactory, and to lead to problems of use, maintenance and training.

Self Assessment

Fill in the blanks:

12. A ..... is a complete development environment for building and maintaining knowledge-based applications.
13. The ..... is a command line interpreter developed by Byron Rakitzis and Paul Haahr, that uses a scripting language similar to the rc shell of the Plan 9 operating system.

Notes

13.8 Dealing with Uncertainty

The description of knowledge is quite vague and imprecise, and includes a great deal of Uncertainty. There are many mathematical theories for expressing uncertainty and which could be used to deal with the uncertainty in the description of the user knowledge. We decided to use fuzzy set theory (Zadeh, 1965): the user model representation is based on fuzzy sets and model updating on fuzzy rules (Kavcic, 2001).

13.8.1 Domain Knowledge Representation

There are two types of relations between domain concepts. The essential prerequisite relation can exist between two concepts only to some extend; therefore we describe it as a fuzzy relation (Kavcic, 2001). The supportive prerequisite relation is always fully present (if it is present at all) and is therefore described as a normal crisp relation. Since the essential prerequisite relation is defined as a fuzzy relation, the domain concept graph is also a fuzzy structure.

13.8.2 User Knowledge Representation

The domain knowledge representation is also used for describing user knowledge, which is a subset of the domain knowledge. An overlay over the domain model is used, so the user model can be regarded as a sub graph of the domain concept graph. Each concept of the sub graph has some additional properties attached, which explain the user knowledge of this concept: a triple of membership functions for three fuzzy sets of unknown, known and learned concepts. As a result, a fuzzy graph with fuzzy relations and fuzzy nodes is a base for user knowledge representation (Kavcic, 2001).

13.8.3 Knowledge Determination and Updating

The user knowledge of domain concepts changes (increases) during the interaction with the system. Consequently, the user model also changes to reflect the current user understanding of the teaching domain. The main principle for gathering information about the user knowledge is checking tests results and analyzing learning units that the user visits. If the user knowledge changes through user actions, it always increases. Even when the user performs poorly, it never decreases.

13.8.4 Model Initialization

The user model is initialized using the results of a quick pre-test, which each new user is required to solve. During this initialization, each domain concept becomes either fully learned or remains completely unknown.

13.8.5 Model Updating

The most significant changes in user knowledge of the domain can be recorded when the user answers the test questions corresponding to a certain learning unit. Tests are used for checking how well a particular concept is learned and a set of corresponding test questions is provided for each learning unit. After the user successfully solves the test on one domain concept, the membership value for a set of learned concepts is increased for this concept, regardless of the previous knowledge level of this particular concept. If the test questions are not answered satisfactory, the value does not change. We also update the user model after each visit of a learning unit; especially first visits of units and visits of still not learned units have the biggest

Notes

influence to the user knowledge. In this case, the value of membership function for a set of known concepts is increased for the concept that is explained in the unit. This can be applied only to still unknown concepts, for which the value of membership function for a set of unknown concepts is greater than zero.

Did u know? The actual increase in value also depends on the values of all prerequisite concepts.

13.8.6 Knowledge Value Propagation

Because the domain concepts are interrelated, we can also infer knowledge values of some concepts. This way, the knowledge of essential prerequisite concepts is inferred on the basis of demonstrated concept knowledge. After every change of concept knowledge values, an inferring mechanism (knowledge value propagation) is triggered that updates the values of all essential prerequisite concepts. This propagation algorithm is based on six fuzzy rules and works recursively on all essential prerequisite concepts, until it reaches the basic concepts that have no prerequisites. This way, the changes of one concept values are reflected in values of all concepts that are essential prerequisites to this concept. Its difficult to understand the uncertainty in ES for this we take the help from the result of analysis in this field. For this we will see the following analysis.

13.8.7 Qualitative and Quantitative Analysis

Qualitative analysis can be defined as identifying qualitative structures, identifying the states of those qualitative structures, and the pattern of changes (transformations) in those states. Quantitative methods can sometimes be used to aid this process, but usually qualitative methods are exclusively used for the analysis of qualitative data and structures for which quantities proper are difficult to define. Thom (1975) argues that all quantitative analysis assumes a firm qualitative foundation. Before they measure, people must agree that there is something to be measured, and that is a qualitative judgment.

Caution People must agree that the measure (metric) they use is appropriate, and applicable to other phenomena.

Example: As an example consider per capita income. It is apparently easy enough to agree on the structure, but the metric is another issue. If currency is used as a metric, a poor family in the United States would be a wealthy one in Pakistan. The metric can be further adjusted by considering cost of living, but an acceptable level of living in the United States is not equivalent to one in Pakistan. The problem is not difficult to understand qualitatively: there are different standards in the two places. The two countries' per capita income can be compared quantitatively, but the interpretation of the comparison is qualitative. The quantitative analysis is more difficult to reconcile, and indeed is undecidable without reference to qualitative structures in the two societies.

In most cases quantitative analysis depends on continuity. To quantify a phenomenon meaningfully it is usually necessary to assume that the relation between phenomena and metric can be described by a continuous function, since a primary goal of quantification is to provide a basis for comparison. For phenomena where the analytic focus is on states this is often

Notes

misleading or impossible. In most social phenomena there is no continuous function that can adequately describe the important qualitative relationships.

Example: As an example consider income and education. These are variables that are often given a quantitative definition in social research. They are relatively easy to define, and people generally measure income in currency, education in years. But they often assume linearity and usually there will be a good correlation between them. But it will not be a perfect correlation, as one unit change in the independent variable will not result in some regular linear unit change in the dependent variable. Now this is not terribly shocking, since people do not expect that all the variation in one variable is to be explained by the other, but there is benefit in understanding the relationship between the variables by breaking the relationship into stages, and examining the conditions for moving from one stage to the next. For instance, in the USA 11 years of education is minimally better than 10 years, but 12 years is far better than 11. This is because of the local structure of American education: 11 years is pregraduation, and 12 years is post-graduation. The graduating student has a qualitatively changed educational status, the pre-graduating student has not significantly changed status. This type of analysis helps to give a better account of interactions.

Self Assessment

Fill in the blanks:

14. The description of knowledge is quite vague and imprecise, and includes a great deal of ..... .
15. ..... can be defined as identifying qualitative structures, identifying the states of those qualitative structures, and the pattern of changes (transformations) in those states.

13.9 Speech Recognition

Automatic speech recognition is the course by which a computer plots an auditory speech signal to text. Automatic speech perceptive is the process by which a computer plots an auditory speech signal to some form of conceptual meaning of the speech. Speech synthesis is the job of converting written input to spoken output. The input can either be offered in a graphemic/orthographic or a phonemic script, based on its source. As a result of its reliance on phonology, linguistics, signal processing, statistics, computer science, acoustics, connectionist networks, psychology and other fields, there are numerous technologies included in speech technology.

Self Assessment

Fill in the blanks:

16. Automatic ..... perceptive is the process by which a computer plots an auditory speech signal to some form of conceptual meaning of the speech.
17. ..... is the job of converting written input to spoken output.

13.10 Summary

- An expert system is software that attempts to reproduce the performance of one or more human experts, most commonly in a specific problem domain, and is a traditional application and/or subfield of artificial intelligence.

Notes

- Forward chaining starts with the data available and uses the inference rules to conclude more data until a desired goal is reached.
- Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals.
- The user interacts with the system through a user interface which may use menus, natural language or any other style of interaction).
- The end-user usually sees an expert system through an interactive dialog.
- The general architecture of an expert system involves two principal components: a problem dependent set of data declarations called the knowledge base or rule base, and a problem independent (although highly data structure dependent) program which is called the inference engine.
- An inference rule is a statement that has two parts, an if-clause and a then-clause.
- The function of the procedure node interface is to receive information from the procedures coordinator and create the appropriate procedure call.
- The function of the user interface is to present questions and information to the user and supply the user's responses to the inference engine.
- A shell is a complete development environment for building and maintaining knowledge-based applications.

13.11 Keywords

Backward Chaining: Backward Chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals.

End User: The end-user usually sees an expert system through an interactive dialog.

Expert System: An expert system is software that attempts to reproduce the performance of one or more human experts, most commonly in a specific problem domain, and is a traditional application and/or subfield of artificial intelligence.

Forward Chaining: Forward chaining starts with the data available and uses the inference rules to conclude more data until a desired goal is reached.

Inference Rule: An inference rule is a statement that has two parts, an if-clause and a then-clause.

Shell: A shell is a complete development environment for building and maintaining knowledge-based applications.

User Interface: The user interacts with the system through a user interface which may use menus, natural language or any other style of interaction.

13.12 Review Questions

1. What is Expert System? Illustrate the main components of expert system.
2. Explain the concept of chaining in expert system.
3. What is the important module of ES?
4. Make distinction between Procedure Node Interface and User Interface.
5. Make distinction between expert systems and problem solving system.

Notes

6. Explain the general architecture of an expert system.
7. What do you understand by Dealing with uncertainty?
8. What is ES Shell? Identify the advantages and disadvantages of Expert System Shells.
9. Explain the applications of Expert system.
10. Describe the concept of Programming Environments of expert system.

Answers: Self Assessment

1.	Expert System	2.	Domain
3.	Forward	4.	Backward
5.	End-user	6.	Inference Rule
7.	Operating Systems	8.	Decision-making
9.	Common Sense	10.	Problems
11.	Conventional	12.	Shell
13.	Es Shell	14.	Uncertainty
15.	Qualitative Analysis	16.	Speech
17.	Speech Synthesis		

13.13 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS: *A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Notes

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

onlinemca.com/.../artificialintelligence/architecture_of_expert_system...

Notes

Unit 14: Prolog

CONTENTS

Objectives
Introduction
14.1 AI Programming Languages
14.1.1 AI Language Prolog
14.2 Converting English to Prolog Facts and Rules
14.2.1 Facts
14.2.2 Rules
14.2.3 Conversion from English to Prolog Facts and Rules
14.3 Goals
14.4 Prolog Terminology
14.4.1 Atom
14.4.2 Numbers
14.4.3 Variables
14.4.4 Compound Term
14.5 Control Structures
14.5.1 Conjunction, Disjunction, Fail and True
14.5.2 Cuts
14.5.3 If-then-else
14.5.4 Variable Goals and Calls
14.5.5 Repeat
14.5.6 Once
14.5.7 Negation
14.6 Arithmetic Operators
14.7 Matching
14.8 Backtracking
14.9 Lists
14.10 Input/Output and Streams
14.10.1 Input/Output using Current Streams
14.10.2 Explicitly Specifying Streams for Input/Output
14.11 Summary
14.12 Keywords
14.13 Review Questions
14.14 Further Readings

Dinesh Kumar, Lovely Professional University

Notes

Objectives

After studying this unit, you will be able to:

- Understand the concerting English to prolog facts
- Discuss the goals execution and prolog terminology
- Illustrate the variables, control structures and arithmetic operators
- Understand the concept of matching, backtracking and lists
- Discuss the input/output and streams

Introduction

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. 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 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.

14.1 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 subsymbolic 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 proto-typing). 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). Rather, 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

Notes

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

14.1.1 AI Language Prolog

Prolog is a logical and a declarative programming language. The name itself, Prolog, is short for PROgramming in LOGic. Prolog's heritage includes the research on theorem provers and other automated deduction systems developed in the 1960s and 1970s. The inference mechanism of Prolog is based upon Robinson's resolution principle (1965) together with mechanisms for extracting answers proposed by Green (1968). These ideas came together forcefully with the advent of linear resolution procedures. Explicit goal-directed linear resolution procedures, such as those of Kowalski and Kuehner (1971) and Kowalski (1974), gave impetus to the development of a general purpose logic programming system. The "first" Prolog was "Marseille Prolog" based on work by Colmerauer (1970). The first detailed description of the Prolog language was the manual for the Marseille Prolog interpreter (Roussel, 1975).

The other major influence on the nature of this first Prolog was that it was designed to facilitate natural language processing. Prolog is the major example of a fourth generation programming language supporting the declarative programming paradigm. The Japanese Fifth-Generation Computer Project, announced in 1981, adopted Prolog as a development language, and thereby focused considerable attention on the language and its capabilities.

Properties of Prolog as a Programming language:

- There are no explicit types or classes in this
- They are rule-based, founded on first-order logic
- There is high expressibility: functionality per program line
- Interactive, experimental programming

Background for Prolog

Prolog can be understand as PROgramming in LOGic:

- ***Syntax:*** subset of 1.-order logic
- ***Declarative semantics:*** Logical consequence
- ***Procedural semantics:*** Resolution, proof rule with unification; Robinson, 1965
- ***A. Colmerauer & co. (Marseille), ca. 1970:*** "Prolog" D.H.D. Warren: Efficient compiler, abstract machine „WAM‰, 1975,
- Language made known by R. Kowalski "Logic for Problem solving", 1979.

Notes

In Prolog program is a description of data

parent(pam, bob). % Pam is a parent of Bob

parent(tom, bob).

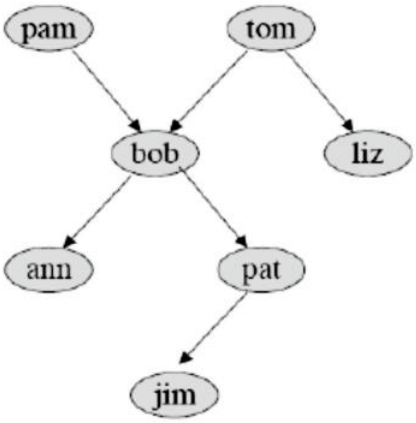
parent(tom, liz).

parent(bob, ann).

parent(bob, pat).

parent(pat, jim)

Figure 14.1: Program in Prolog



Basic Notions

There are following notions present in prolog:

- Predicates: **parent**
  - describes a relation
  - defined by facts, rules, collectively called clauses
- Constant (symbol)s: **tom, bob, x, y**
- Variables: **X, Y, Tom**
- Atoms (simple goals): **parent (A, a)**
- Queries....

Did u know? In Prolog literature, constants are called atoms.

Self Assessment

Fill in the blanks:

1. Prolog is a logical and a declarative ..... language.
2. Prolog is based upon ..... principle (1965) together with mechanisms for extracting answers proposed by Green (1968).

Notes

14.2 Converting English to Prolog Facts and Rules

14.2.1 Facts

In ProLog, facts describe the relationships between different objects and are independent of each other. Facts are also called ground clauses and are the basis to derive further information.

You can declare facts using predicates or functions. Predicates can take a number of arguments, which are enclosed within parentheses and are separated from each other by commas. The number of arguments that a predicate takes is known as its arity.

Example: The ProLog statement for the fact that NeoRage is an employee is:

employee(neorage).

In the above example, employee is the predicate and NeoRage is a data object and an argument to the predicate. The 'arity' of the predicate is one because it has only one argument.

In the following ProLog statement, the arity of the predicate is two because it has two arguments for the fact that Sam is the father of Jack:

father(sam, jack).

In the above example, father is the predicate and Sam and Jack are the two arguments. ProLog evaluates facts from left to right so the above statement cannot be interpreted in reverse order and mean that Jack is the father of Sam.

Notes A fact begins with a lowercase letter and a statement ends with a period. All data objects, such as sam and jack, are atoms and cannot begin with an uppercase.

14.2.2 Rules

In ProLog, rules are used in the process of decision-making and can deduce new facts from existing ones.

Example: Suppose there are two facts such as: trope likes mary bob likes sam

The rule says:

jim likes X if bob likes X.

Prolog can deduce that:

jim likes sam (jim lives in San Fran lol)

You can give a ProLog program a goal that is a problem it needs to find a solution for.

Example: find every person who likes sam**:** ProLog will use its deductive ability to find all solutions to the problem.

A rule consists of two parts: a conditional part, if, and a conclusion or action part, then.

Notes

The if and then parts of a rule are separated by the: symbol, referred to as the infix operator. The conditional part of the rule is written on the right of the infix operator and conclusion part on its left. A rule can be declared with a condition. For example, you can declare that if the weather is sunny, the day is to be selected for a picnic. The ProLog statement for declaring this rule and its condition is:

picnic(day) :- weather(day,sunny).

14.2.3 Conversion from English to Prolog Facts and Rules

The following program includes a number of predicates that portray a family's genealogical relations.

```
female(amy).
female(merry).

male(john).
male(bruce).
male(ogden).

parentof(amy,merry).
parentof(amy,john).
parentof(amy,bruce).
parentof(ogden,merry).
parentof(ogden,john).
parentof(ogden,bruce).
```

The above program includes the three straightforward predicates: female; male; and parentof. They are parameterized with what are known as 'atoms.' There are other family relations which could also be defined as facts, but this is a deadly process. Presuming customary marriage and child-bearing practices, we could define some rules which would reduce the tediousness of determining and listing all the probable family relations.

Example: Suppose you are required to know if merry had any siblings, the first question you must request is "what does it mean to be a sibling?" To be someone's sibling you must have the same parent. This last sentence can be written in Prolog as

```
siblingof(X,Y) :-
        parentof(Z,X),
        parentof(Z,Y).
```

A conversion of the above Prolog rule into English would be "X is the sibling of Y given that Z is a parent of X, and Z is a parent of Y." X, Y, and Z are variables. This rule though, also defines a child to be its own sibling. To correct this we must add that X and Y are not similar. The exact version is:

siblingof(X,Y) :-

Notes

```
    parentof(Z,X),
    parentof(Z,Y),
    X  Y.
```

The relation brotherof is alike but adds the condition that X must be a male.

```
brotherof(X,Y) :-
    parentof(Z,X),
    male(X),
    parentof(Z,Y),
    X  Y.
```

Self Assessment

Fill in the blanks:

3. In ProLog, ... describe the relationships between different objects and are independent of each other.
4. In ProLog, ... are used in the process of decision-making and can deduce new facts from existing ones.

14.3 Goals

Proposing a query means demanding Prolog to attempt to prove that the statement(s) implied by the query can be prepared true given the right variable instantiations are prepared. The search for such a proof is generally pointed to as goal execution. Every predicate in the query comprises a (sub)goal, which Prolog attempts to please one after the other. If variables are shared among numerous subgoals their instantiations have to be similar throughout the whole expression.

If a goal goes with the head of a rule, the particular variable instantiations are made within the rule's body, which then turns out to be the new goal to be satisfied. If the body includes numerous predicates the goal is again split into subgoals to be executed in turn.

Alternatively, the head of a rule is measured provably true, if the conjunction of all its body predicates are provably true. If a goal goes with a fact in our program the proof for that goal is absolute and the variable instantiations made throughout matching are conversed back to the surface.

Notes The order in which facts and rules emerge in our program is significant here. Prolog will for all time attempt to match its existing goal with the first possible fact or rule-head it can find.

If the principal function of a goal is an incorporated predicate the connected action is executed whilst the goal is being satisfied.

Example: As far as goal execution is regarded the predicate write ('Hello World!') will just succeed, but simultaneously it will also print the words Hello World! on the screen.

Notes

The incorporated predicate true will always do well (without any further side-effects), while fail will at all times fail. At times there is more than one method of satisfying the existing goal. Prolog selects the first option (as determined by the order of clauses in a program), but the fact that there are choices is recorded. If at some point Prolog fails to confirm a certain subgoal, the system can go back and attempt a substitute way of executing the earlier goal. This process is called backtracking. We shall demonstrate the process of goal execution by using the following well-known argument:

All men are mortal.

Socrates is a man.

Hence, Socrates is mortal.

In Prolog terms, the first statement shows a rule: X is mortal, if X is a man (for all X). The second one comprises a fact: Socrates is a man. This can be executed in Prolog as below:

mortal(X) :- man(X).

man(socrates).

Note that X is a variable, while socrates is an atom. The conclusion of the argument, "Socrates is mortal", can be displayed via the predicate mortal(socrates). After having conferred with the above program we can propose this predicate to Prolog as a query, which will cause the following reaction:

?- mortal(socrates).

Yes

Prolog consents with our own rational reasoning, which is nice. But how did it reach to its conclusion? Let's follow the goal execution step by step.

1. The query mortal(socrates) is made the initial goal.
2. Scanning via the clauses of our program, Prolog attempts to match mortal(socrates) with the first probable fact or head of rule. It finds mortal(X), the head of the first (and only) rule. When matching the two terms the instantiation X = socrates requires to be made.
3. The variable instantiation is extended to the body of the rule, i.e. man(X) turns out to be man(socrates).
4. The newly instantiated body turns out to be our new goal: man(socrates).
5. Prolog implements the new goal by again attempting to match it with a rule-head or a fact. Clearly, the goal man(socrates) matches the fact man(socrates), since they are identical. This means the current goal succeeds.
6. This, again, means that also the initial goal succeeds.

Self Assessment

Fill in the blank:

5. Proposing a ..... means demanding Prolog to attempt to prove that the statement(s) implied by the query can be prepared true given the right variable instantiations are prepared.

Notes

14.4 Prolog Terminology

Prolog's single data type is the term. Terms are either *atoms, numbers, variables* or *compound terms*.

14.4.1 Atom

An atom is a general-purpose name with no inherent meaning. It is composed of a sequence of characters that is parsed by the Prolog reader as a single unit. Atoms are usually bare words in Prolog code, written with no special syntax. However, atoms containing spaces or certain other special characters must be surrounded by single quotes. Atoms beginning with a capital letter must also be quoted, to distinguish them from variables. The empty list, written [], is also an atom.

Example: Other examples of atoms include x, blue, Taco, and some atom.

An atom is a data object in ProLog and is also used as a name of an individual or a predicate. An atom is a word-like entity and has the following characteristics:

- It begins with a lowercase letter and contains letters, digits, and an underscore.
- It can be enclosed in single quotes that contain any character, such as a space.
- It does not have a length limit.

Example: Some examples of atoms are:

sam: An atom with only alphabet characters

a_long_word: An atom with embedded underscores

'New Jersey': An atom enclosed in parentheses because it contains an embedded space

13$$56: An atom starting with digits

Use atoms to construct more complex entities, such as structures and lists

14.4.2 Numbers

Numbers can be floats or integers. Many Prolog implementations also provide unbounded integers and rational numbers. All standard ProLog implementations have numbers that are positive, negative, or floating-point integers. Some implementations handle the exponential format.

The knowledge base in ProLog is written in free format because there is no on the number of free spaces that a program can have. A new line is allowed at any point in the program but there are two restrictions: the atom or variable name cannot have embedded spaces and there cannot be anything between the function and the opening parentheses.

Listing 1 shows a ProLog program that has a knowledge base of facts and rules related to hardware equipments:

Listing 1: Identifying Hardware Equipment

```
/* Program to identify hardware */

/* Rule to check for hardware */

hardware(X) :- equipment(X).
```

Notes

```
/* An item qualifies to be a hardware if it qualified to be an equipment */
/*List of facts */
equipment(mouse).
equipment(keyboard).
equipment(modem).
equipment('web camera').
equipment(monitor).
equipment('hard disk').
/* End of program */
```

14.4.3 Variables

Variables are denoted by a **string** consisting of letters, numbers and underscore characters, and beginning with an upper-case letter or underscore. Variables closely resemble variables in logic in that they are placeholders for arbitrary terms. A variable can become instantiated (bound to equal a specific term) via unification. A single underscore (_) denotes an anonymous variable and means "any term". Unlike other variables, the underscore does not represent the same value everywhere it occurs within a predicate definition. A variable name contains letters, digits, and underscores. It begins with a capital letter or an underscore mark.

Example: Some examples of variables are Result, _total, and _a123.

There are two types of variables:

- ***Free variable:*** A variable whose value is unknown to the Prolog program
- ***Bound variable:*** A variable whose value is set

To understand the different variable types, consider the example of the following set of facts:

```
likes(trope, reading).
likes(bb, swimming).
likes(trope, football).
likes(Saj, reading).
```

Consider that the rule for deducing from the above facts is:

likes(X, reading) and likes(X, football).

ProLog searches from left to right, so the first subgoal is:

likes(X, reading)

In the above statement of subgoal the first argument is X, which is a free variable and the value in second argument is reading. ProLog attempts to satisfy the subgoal with facts that contain reading as the second argument. In the example the match found is:

likes(trope, reading).

The X variable is now bound to the value, trope; the ProLog execution then shifts to the next subgoal, which is:

likes(trope, swimming).

The above statement shows that the variable X has been substituted with the value, trope.

Notes

14.4.4 Compound Term

A compound term is composed of an atom called a functor and a number of "arguments", which are again terms. Compound terms are ordinarily written as a function followed by a comma-separated list of argument terms, which is contained in parentheses. The number of arguments is called the term's arity. An atom can be regarded as a compound term with arity zero.

Example: Examples of compound terms are truck_year ('Mazda', 1986) and Person_Friends' (zelda,[tom,jim]).

Compound terms with functors that are declared as operators can be written in prefix or infix notation.

Example: The terms -(z), +(a,b) and =(X,Y) can also be written as -z, a + b and X = Y, respectively.

Users can declare arbitrary functors as operators with different precedences to allow for domain-specific notations. The notation *f/n* is commonly used to denote a term with function f and arity n.

Special cases of compound terms:

- ***Lists are defined inductively:*** The atom [] is a list. A compound term with functor . (dot) and arity 2, whose second argument is a list, is itself a list. There exists special syntax for denoting lists: .(A, B) is equivalent to [A | B].
- ***Strings:*** A sequence of characters surrounded by quotes is equivalent to a list of (numeric) character codes, generally in the local character encoding or Unicode if the system supports Unicode.

Task Make distinction between numbers, atoms and variables.

Self Assessment

Fill in the blanks:

6. ..... is composed of a sequence of characters that is parsed by the Prolog reader as a single unit.
7. A ..... is composed of an atom called a functor and a number of "arguments", which are again terms.

14.5 Control Structures

14.5.1 Conjunction, Disjunction, Fail and True

As in practically all Prologs, the comma (,) means "and," the semicolon (;) means "or," fail always fails, and true always succeeds with no other action.

Notes

14.5.2 Cuts

The cut (!) functions in the conventional manner. When implemented, it succeeds and throws away all backtrack points among itself and its CUTPARENT. Usually, the cutparent is the query that caused execution to enter the existing clause. Though, if the cut is in an environment that is OPAQUE TO CUTS, the cutparent is the commencement of that environment.

Example: Examples of surroundings that are opaque to cuts are:

_ The argument of the negation operator (\+).

_ The argument of call, which can certainly be a compound goal, such as call((this,!, that)).

_ The left–hand argument of '->' (see below).

The objectives that are arguments of once, catch, findall, bagof, and setof (and, usually, any other goals that are arguments of predicates).

14.5.3 If-then-else

The "if–then–else" build (Goal1 -> Goal2 ; Goal3) tries to implement Goal1, and, if successful, proceeds to Goal2; or else, it proceeds to Goal3. The semicolon and Goal3 can be omitted.

Observe that:

_ Only the first solution to Goal1 is established; any backtrack points produced while executing Goal1 are thrown away.

_ If Goal1 succeeds, execution proceeds to Goal2, and then:

If Goal2 fails, the whole construct fails.

- If Goal2 succeeds, the whole construct succeeds.
- If Goal2 has multiple solutions, the whole construct has multiple solutions.

_ If Goal1 fails, execution proceeds to Goal3, and then:

If Goal3 fails, the whole construct fails.

- If Goal3 succeeds, the whole construct succeeds.
- If Goal3 has multiple solutions, the whole construct has multiple solutions.

_ If Goal1 fails and there is no Goal3, the whole construct fails.

_ Either Goal2 or Goal3 will be executed, but not both (not even upon backtracking).

_ If Goal1 contains a cut, that cut only has scope over Goal1, not the whole clause. That is, Goal1 is opaque to cuts.

_ The whole if–then–else structure has numerous solutions if Goal1 succeeds and Goal2 has multiple solutions, or if Goal1 fails and Goal3 has multiple solutions. That is, backtrack points in Goal2 and Goal3 behave normally.

_ Cuts in Goal2 and Goal3 have scope over the entire clause (i.e., behave normally).

Notes

Notes Observe that the semicolon in Goal1 -> Goal2 ; Goal3 is not the normal disjunction operator; if it were, you would be capable to obtain solutions to Goal1 -> Goal2 and then, upon backtracking, also obtain solutions to Goal3. But this never happens. Rather, -> and; have to be interpreted as a unit.

Caution We do not suggest mixing cuts with if–then or if–then–else structures.

14.5.4 Variable Goals and Calls

Variables can be utilized as goals. A term G which is a variable appearing representing a goal is transformed to the goal call(G). Note that call is not clear to cuts.

14.5.5 Repeat

The predicate repeat functions in the traditional manner, i.e., whenever backtracking reaches it, execution continues forward again via the same clauses as if another substitute had been found.

14.5.6 Once

The query once(Goal) locates exactly one solution to Goal. It is comparable to call((Goal,!)) and is opaque to cuts.

14.5.7 Negation

The negation predicate is written \+ and is obscure to cuts. That is, \+ Goal is like call(Goal) apart from its success or failure is the opposite.

Note that extra parentheses are needed around compound goals (e.g., \+ (this, that)).

Self Assessment

Fill in the blanks:

8. The predicate ..... functions in the traditional manner, i.e., whenever backtracking reaches it, execution continues forward again via the same clauses as if another substitute had been found.
9. The query ..... locates exactly one solution to Goal. It is comparable to call((Goal,!)) and is opaque to cuts.

14.6 Arithmetic Operators

Prolog has the usual range of Arithmetic Operators (+, -, =, * ..)

Prolog must be able to manage arithmetic in order to be a functional general purpose programming language. Though, arithmetic does not fit well into the logical plan of things.

Notes

Specifically, the thought of evaluating an arithmetic expression is in difference to the straight pattern matching we have observed so far. For this reason, Prolog offers the incorporated predicate 'is' that assesses arithmetic expressions. Its syntax calls for the utilization of operators.

X is <arithmetic expression>

The variable X is fixed to the value of the arithmetic expression. On backtracking it is unallocated.

The arithmetic expression appears like an arithmetic expression in any other programming language.

Here is how to make use of Prolog as a calculator.

?- X is 2 + 2.

X = 4

?- X is 3 * 4 + 2.

X = 14

Parentheses clarify precedence.

?- X is 3 * (4 + 2).

X = 18

?- X is (8 / 4) / 2.

X = 1

In addition to 'is,' Prolog offers a number of operators that contrast two numbers. These comprise 'greater than', 'less than', 'greater or equal than', and 'less or equal than.' They perform more logically, and succeed or fail as per whether the contrast is true or false.

Notice the order of the symbols in the greater or equal than and less than or equal operators. They are particularly constructed not to appear like an arrow, so that you can use arrow symbols in your programs without confusion.

X > Y

X < Y

X >= Y

X =< Y

Example: Here are a few examples of their use.

?- 4 > 3.

yes

?- 4 < 3.

no

?- X is 2 + 2, X > 3.

X = 4

Notes

```
?- X is 2 + 2, 3 >= X.
no

?- 3+4 > 3*2.
yes
```

They can be used in rules also.

Example: Here are two example predicates. One transforms centigrade temperatures to Fahrenheit, the other verifies if a temperature is below freezing.

```
c_to_f(C,F) :-
  F is C * 9 / 5 + 32.

freezing(F) :-
  F =< 32.
```

Here are some examples of their use.

```
?- c_to_f(100,X).
X = 212
yes

?- freezing(15).
yes

?- freezing(45).
no
```

Self Assessment

Fill in the blank:

10. Prolog must be able to manage arithmetic in order to be a ..... general purpose programming language.

14.7 Matching

Two terms are defined to match if they are either indistinguishable or if they can be made indistinguishable by variable instantiation. Instantiating a variable means allocating it a xed value. Two free variables also match, since they could be instantiated with the similar ground term.

Notes

Did u know? The similar variable has to be instantiated with the similar value all through an expression. The only exception to this rule is the unidentified variable _, which is measured to be unique whenever it appears.

Example: We provide some examples. The terms is_bigger(X, dog) and is_bigger(elephant, dog) match, since the variable X can be instantiated with the atom elephant. We could check this in the Prolog interpreter by submitting the equivalent query to which Prolog would respond by listing the suitable variable instantiations:

?- is_bigger(X, dog) = is_bigger(elephant, dog).

X = elephant

Yes

The following is an example for a query that doesn't succeed, because X cannot match with 1 and 2 at the same time.

?- p(X, 2, 2) = p(1, Y, X).

No

If, however, rather than X we use the anonymous variable _, matching is possible, since every occurrence of _ represents a different variable. All through matching Y is instantiated with 2:

?- p(_, 2, 2) = p(1, Y, _).

Y = 2

Yes

Another example for matching:

?- f(a, g(X, Y)) = f(X, Z), Z = g(W, h(X)).

X = a

Y = h(a)

Z = g(a, h(a))

W = a

Yes

Until now so good but what happens, if matching is probable even though no specific variable instantiation has to be imposed. Consider the following query:

?- X = my_functor(Y).

X = my_functor(_G177)

Y = _G177

Yes

In this example matching succeeds, since X could be a compound term with the function my_function and a non-specified single argument. Y could be any valid Prolog term, but it has

Notes

to be the similar term as the argument inside X. In Prolog's output this is indicated through the use of the variable _G177. This variable has been produced by Prolog during execution time. Its particular name, _G177 here, may be different each time the query is submitted.

Self Assessment

Fill in the blanks:

11. Two terms are defined to ..... if they are either indistinguishable or if they can be made indistinguishable by variable instantiation.
12. The similar variable has to be instantiated with the similar ..... all through an expression.

14.8 Backtracking

The backtracking predicate provides an alternative path for a program when the current path fails.

The response to a query is generated after executing and evaluating various conditional statements in the program. During execution, the control goes to the first conditional statement and tests it with a variable. If the condition is FALSE, the execution moves back and tries to prove the condition with another variable. This looping back to the previous statement is called backtracking.

Listing 3 shows a sample ProLog program that uses the backtracking predicate:

Listing 3: Program with Backtracking

```
/* The program specifies that the weather can be sunny on Friday, Saturday, or Sunday and the
weekend is on Saturday and Sunday.*/

/* Facts */

weather(friday, sunny).

weather(saturday, sunny).

weather(sunday, sunny).

weekend(saturday).

weekend(sunday).

/* The rule states that outings are possible on weekends, when the weather is sunny */

/* Write is a built-in predicate to display on screen */

outing(Day) :- weather(Day, sunny),

weekend(Day),

write(Day),

nl.

/* End of Program */
```

Backtracking allows Prolog to find all alternative solutions to a given query.

Notes

Self Assessment

Fill in the blanks:

13. The ..... predicate provides an alternative path for a program when the current path fails.
14. Backtracking allows Prolog to nd all ..... solutions to a given query.

14.9 Lists

A list is an ordered sequence of zero or more terms that has the following characteristics:

- It is written between square brackets and separated by commas.
- Every nonempty list is divided into a head and tail using the pipe (|) symbol. The tail of a list is always a list and the head of a list is an element.
- An empty list is written as [] and does not have a term.

The size of the list is not declared and can be increased or decreased dynamically within the program.

Each element in the list is accompanied by a pointer, which indicates the location of the next element in the list.

Example: Some examples of list are [jack, jill, jim], [1,2,3,5], [(a, sublist, in, list), main, list], and [a|[b,c,d]].

Self Assessment

Fill in the blanks:

15. A ..... is an ordered sequence of zero or more terms.
16. Each element in the list is accompanied by a ....., which indicates the location of the next element in the list.

14.10 Input/Output and Streams

Two groups of input/output predicates are obtainable for input and output: that is, those which refer completely to a current stream, and those which need the stream to be explicitly specified.

Current Streams

There is one current input stream, one current output stream and one current error stream at any specified time. At system startup these are the typical streams. If necessary, you can alter this default setting. You should redefine a current stream only if you desire to consequently use it frequently for input or output. The input/output predicates in this group implicitly address the current streams, because a stream is not explicitly specified.

Explicitly Specified Stream

The specified stream must already be present when the equivalent predicate is called. If the stream is a file, it must be open for reading or writing.

Notes

In Prolog, streams for input and output are pointed to by alias names or via a system-defined terms which the Prolog system supplies with the predicateopen/3/4. The alias names are atoms. The following table offers an outline of the streams which can be addressed. For addressing, either the device name or the alias name can be used. The device name, or driver name, is used only on opening a stream. The alias name displays a logical reference to a stream and discovers a link among a stream and a name. The name is liberally selectable.

Stream	**Alias name**
Standard input	user_input
Standard output	user_output
Standard error output	user_error
Keyboard	keyboard
Screen	screen

The following defaults are fixed for the three standard streams:

Stream	**Default**
Standard input	Keyboard
Standard output	Screen or window in which Prolog was started
Standard error output	Screen or window in which Prolog was started and error message window

You can also redefine the typical streams when calling Prolog.

Example: The file *goals.pro* is presumed to enclose the following goals:

```
Y is 3 + 4.
append(A, B, [a,b,c,d]).
atom(A).
```

If you then call Prolog and state the file as the input stream, these goals are developed sequentially. The following output occurs on the screen:

```
$ prolog < goals.pro
Y     = 7
yes

A     = []
B     = [a,b,c,d]
yes

no
$
```

When the end of the file is reached, Prolog is finished.

You can utilize the predicate stream_copy/2 to modify the standard stream assignments throughout a Prolog session.

Notes

14.10.1 Input/Output using Current Streams

Input/output predicates in which no stream is mentioned address the current streams. The predicate listing/0/1 also sends its outputs to the existing output stream. The Prolog system concerns error messages on the existing error stream.

When the Prolog system is begin, the corresponding standard streams are first opened, i.e.

Stream	Alias	Default
Current input	current_input	user_input (standard input)
Current output	current_output	user_output (standard output)
Current error output	current_error	user_error (standard error output)

You can point to the current streams by their alias names in input/output predicates which need a stream to be mentioned.

You can query or modify the settings for the current streams with the following predicates:

Predicate	Purpose
current_input(?*Stream*)	Query current input stream
set_input(@*Stream*)	Set current input stream
see(@*Stream*)	Set and if required, open current input stream
seeing(?*Stream*)	Query current input stream
seen	Reset current input stream to user_input and if required, close the first stream
current_output(?*Stream*)	Query current output stream
set_output(@*Stream*)	Set current output stream
tell(@*Stream*)	Set and if required, open current output stream
telling(?*Stream*)	Query current output stream
told	Reset current output stream to user_output and if required, close the first stream
current_error(?*Stream*)	Query current error stream
set_error(@*Stream*)	Set current error stream
error_tell(@*Stream*)	Set and if required, open current error stream
error_telling(?*Stream*)	Query current error stream
error_told	Reset current error stream to user_error and if required, close the first stream

When you announce a file as the current input/output stream, you should primarily open it with the predicate open/3/4 and then describe it as the current input/output stream with one of the above predicates. If you desire to state a pipe as the current input/output stream, you should first produce it with unix_make_pipe/1 and open it with open/3/4. Then you define the pipe as the existing input/output stream.

With the following predicates, input is utilizing the current input stream and output is utilizing the current output stream.

Notes

Predicate	Purpose
get_byte(?*ByteCode*)	Read a byte
get_char(?*Char*)	Read a character
get_code(?*CharCode*)	Read a character
get_until(+*SearchChar*, ?*Text*, ?*EndChar*)	Read up to a specified character
nl	Write new line
print(@*Term*)	Write a term
put_byte(+*ByteCode*)	Write a byte
put_char(+*Char*)	Write a character
put_code(+*CharCode*)	Write a character
read(?*Term*)	Read a term
read_term(?*Term*, +*Options*)	Read a term with control options
skip_line	Skip input line
write(@*Term*)	Write a term
write_canonical(@*Term*)	Write a term in normal structure representation and in single quotes
write_formatted(+*Format*, @*TermList*)	Write terms with formatting
write_term(@*Term*, @*Options*)	Write a term using operator notation
writeq(@*Term*)	Write a term in single quotes, using operator notation

14.10.2 Explicitly Specifying Streams for Input/Output

Input and output can be executed also from and to an explicitly specified stream. It is not required to redefine current streams only for a single input or output operation. Rather, a predicate with an extra argument for the stream can be utilized.

If you desire to implement the input/output from or to a stream, you must first open this stream previous to the first read or write operation. If the stream is to be a pipe, you should first create it with the predicate unix_make_pipe(-*Pipename*). *Pipename* is produced by the Prolog system when the pipe is generated.

Open for	file by calling	pipe by calling
Reading	open(file(*Filename*), read, X)	open(pipe*(Pipename*), read, Y)
Writing	open(file(*Filename*), write, X)	open(pipe*(Pipename*), write, Y)
Appending	open(file(*Filename*), append, X)	–

1cm**i** [t]13cmYou can also open a stream (typically a file) by calling see/1, tell/1 or error_tell/1.

1cm**i** [t]13cmFile names which enclose special or Prolog-specific characters, or which start with an uppercase letter, must be contained in single quotes.

1cm**i** [t]13cmYou can open a stream either for reading or for writing, but not for both concurrently.

Notes

The following predicates need the input/output stream to be specified explicitly. The stream is addressed by a name (alias name or term of open/3/4). You can also address the standard streams by their synonyms.

Predicate	**Purpose**
get_byte(@*Stream*, *?ByteCode*)	Read a byte
get_char(@*Stream*, *?Char*)	Read a character
get_code(@*Stream*, *?CharCode*)	Read a character
get_until(@*Stream*, +*SearchChar*, ?*Text*, ?*EndChar*)	Read up to a specific character
nl(@*Stream*)	Write new line
print(@*Stream*, @*Term*)	Write a term
put_byte(@*Stream*, +*ByteCode*)	Write a byte
put_char(@*Stream*, +*Char*)	Write a character
put_code(@*Stream*, *CharCode*)	Write a character
read(@*Stream*, ?*Term*)	Read a term
read_term(@*Stream*, ?*Term*, @*Options*)	Read a term with control options
skip_line(@*Stream*)	Skip input line
write(@*Stream*, @*Term*)	Write a term
write_canonical(@*Stream*, @*Term*)	Write a term in normal structure representation and in single quotes
write_formatted(@*Stream*, +*Format*, @*TermList*)	Write terms with formatting
write_term(@*Stream*, @*Term*, @*Options*)	Write a term using operator notation
writeq(@*Stream*, @*Term*)	Write a term in single quotes, using operator notation

1cm**i** [t]13cmKeyboard inputs are usually buffered one line at a time and are passed to the Prolog system when is pressed. Output to files and pipes is usually also buffered. If necessary, you can clear the output buffers with flush_output/0/1. You can manage buffering with open/3/4.

You can utilize the predicate close/1 to close any stream despite the standard streams. Should you effort to close a standard stream with close/1, it will have no result.

The predicate reset_streams/0 reinstates the defaults for the standard streams

Task Make distinction between Input/output using current streams and explicitly specified streams.

Self Assessment

Fill in the blanks:

17. Input/output predicates in which no stream is mentioned address the ..... streams.

Notes

18. You should ..... a current stream only if you desire to consequently use it frequently for input or output.

14.11 Summary

- Prolog is a logical and a declarative programming language. The name itself, Prolog, is short for PROgramming in LOGic.
- In ProLog, facts describe the relationships between different objects and are independent of each other.
- In ProLog, rules are used in the process of decision-making and can deduce new facts from existing ones.
- Proposing a query means demanding Prolog to attempt to prove that the statement(s) implied by the query can be prepared true given the right variable instantiations are prepared.
- An atom is a general-purpose name with no inherent meaning. It is composed of a sequence of characters that is parsed by the Prolog reader as a single unit.
- Variables are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an uppercase letter or underscore.
- A compound term is composed of an atom called a functor and a number of “arguments”, which are again terms.
- The predicate repeat functions in the traditional manner, i.e., whenever backtracking reaches it, execution continues forward again via the same clauses as if another substitute had been found.
- Two terms are defined to match if they are either indistinguishable or if they can be made indistinguishable by variable instantiation.
- The backtracking predicate provides an alternative path for a program when the current path fails.

14.12 Keywords

Atom: An atom is a general-purpose name with no inherent meaning. It is composed of a sequence of characters that is parsed by the Prolog reader as a single unit.

Backtracking: The backtracking predicate provides an alternative path for a program when the current path fails.

Compound Term: A compound term is composed of an atom called a functor and a number of "arguments", which are again terms.

Facts: In ProLog, facts describe the relationships between different objects and are independent of each other.

Matching: Two terms are defined to match if they are either indistinguishable or if they can be made indistinguishable by variable instantiation.

Prolog: Prolog is a logical and a declarative programming language. The name itself, Prolog, is short for PROgramming in LOGic.

Repeat: The predicate repeat functions in the traditional manner, i.e., whenever backtracking reaches it, execution continues forward again via the same clauses as if another substitute had been found.

Notes

Rules: In ProLog, rules are used in the process of decision-making and can deduce new facts from existing ones.

Variable: Variables are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an uppercase letter or underscore.

14.13 Review Questions

1. What is Prolog? Illustrate the Properties of Prolog as a Programming language.
2. Enlighten how a prolog is considered as an AI Programming Language.
3. Illustrate the process of Converting English to Prolog Facts and Rules.
4. Elucidate the process of achieving goal execution.
5. Describe various terms related to prolog.
6. Demonstrate the working of If-then-else loop.
7. Make distinguish between repeat and once predicate.
8. Describe the concept of matching two terms if they are either identical or if they can be made identical by variable instantiation. Give example.
9. Illustrate the function of backtracking predicate with examples.
10. Explain the concept of Input/Output and Streams with examples.

Answers: Self Assessment

1.	Programming	2.	Robinson's Resolution
3.	Facts	4.	Rules
5.	Query	6.	Atom
7.	Compound Term	8.	Repeat
9.	Once	10.	Functional
11.	Match	12.	Value
13.	Backtracking	14.	Alternative
15.	List	16.	Pointer
17.	Current	18.	Redefine

14.14 Further Readings

Books

Antonelli, D. 1983. *The application of artificial intelligence to a maintenance and diagnostic information system (MDIS).* Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance. Boulder, CO.

Boose, J.H. 1984. *Personal construct theory and the transfer of human expertise.* Proceedings of the National Conference on Artificial Intelligence (AAAI-84), p. 27-33, Austin, Texas.

Notes

Boose, J.H. 1985. *A knowledge acquisition program for expert systems based on personal construct psychology.* International Journal of Man-Machine Studies, 23, 495-525.

Boose, J.H. 1986a. *Expertise Transfer for Expert System Design*, New York; Elsevier.

Boose, J.H. 1986b. *Rapid acquisition and combination of knowledge from multiple experts in the same domain*. Future Computing Systems Journal, 1, 191-216.

Boose, J.H. 1988. *Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies, 29, 287-310.

Boose, J.H. 1989. *A survey of knowledge acquisition techniques and tools.* Knowledge acquisition: An international journal of knowledge acquisition for knowledge-based systems, in press. Vol. 1, No. 1.

Boose, J.H., and Bradshaw, J.M. 1987b. AQUINAS:*A knowledge acquisition workbench for building knowledge based systems*. Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (pp. A6.1-6). Reading University.

Boose, J.H., Bradshaw, J.M., Shema, D.B. 1988. *Recent progress in Aquinas: A knowledge acquisition workbench*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88). p. 2.1-15, Bonn.

Bradshaw, J. 1988. *Shared causal knowledge as a basis for communication between expert and knowledge acquisition system*. Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88) pp. 12.1-6.

Bradshaw, J.M. 1988. *Strategies for selecting and interviewing experts*. Boeing Computer Services Technical report in preparation.

Bradshaw, J.M. and Boose, J.H. 1989. *Decision analytic techniques for knowledge acquisition: Combining situation and preference models using Aquinas*. Special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, International Journal of Man- Machine Studies. in press.

Clancey. W. 1986. Heuristic classification. In J. Kowalik (Ed.). *Knowledge-based problem solving*. New York: Prentice-Hall.

Davis, R., and Lenat, D.B. 1982. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.

DeJong, K. 1987. *Knowledge acquisition for fault isolation expert systems*. Special issue on the 1st AAA1 Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, International/ Journal of Man-Machine Studies, Vol. 27, No. 2.

Online link

www.solanum.org/prolog.htm